

258321 DKE Projekt

**Gruppe 2**

**Teammitglieder:**

k01607605, Aistleithner Andrea

k01256561, Dusanic Maja

k01356577, Teuchtmann Alexander

k01356229, Tomic Milos

KONzeptueller Entwurf

Inhaltsverzeichnis

[1. Systemkomponenten 2](#_Toc531025087)

[1.1. Evaluierungsprogramm 2](#_Toc531025088)

[1.2. Datengenerator Rule Model Inheritance 3](#_Toc531025089)

[1.3. Datengenerator CBR 3](#_Toc531025090)

[1.4. Externer Speicher 5](#_Toc531025091)

[2. Schnittstellen 6](#_Toc531025092)

[2.1. User Schnittstelle 7](#_Toc531025093)

[2.1.1. CBR 8](#_Toc531025094)

[2.1.2. Rule Model Inheritance 8](#_Toc531025095)

[2.2. Rule Model Inheritance Datengenerierung Schnittstelle 10](#_Toc531025096)

[2.3. CBR Datengenerierung Schnittstelle 10](#_Toc531025097)

[2.4. Vadalog Schnittstelle 10](#_Toc531025098)

[2.5. Speicherung Schnittstelle 11](#_Toc531025099)

[3. Umsetzung 12](#_Toc531025100)

[3.1. Klassenstruktur 13](#_Toc531025101)

[3.1.1. DataGenerators 13](#_Toc531025102)

[3.1.2. EvaluationFramework 14](#_Toc531025103)

[3.1.3. Exceptions 14](#_Toc531025104)

[3.1.4. Tests 14](#_Toc531025105)

[3.1.5. Vadalog 14](#_Toc531025106)

[4. Abbildungsverzeichnis 15](#_Toc531025107)

# Systemkomponenten

Das System des Evaluierungsframeworks wird in Abbildung 1 graphisch dargestellt. Die Komponenten im grün umrahmten Bereich, sind die zu implementierenden Komponenten.



Abbildung 1: Systemkomponenten

Im folgenden Abschnitt werden die in der Abbildung 1 dargestellten Komponenten genauer beschrieben. Ebenso wird die geplante Implementierung der Komponenten erläutert und schriftlich dargestellt.

## Evaluierungsprogramm

Die Komponente des Evaluierungsprogramms steht im Mittelpunkt des Systems und agiert als User Interface mit dem Benutzer. Die Implementierung einer grafischen Darstellung vom User Interface wird als unwichtig eingestuft, weil dies nicht zum Ziel dieses Projekts gehört und wird nicht durchgeführt.

Die Eingaben der Daten vom User wird über die Konsole erfolgen. Der User agiert mit der Komponente des Evaluierungsprogrammes über die User Schnittstelle, welche im Abschnitt 2.1 genauer erläutert wird.

Die vom User eingegeben Daten, über die gewünschten Testfälle, werden im Evaluierungsprogramm erfasst und über eine der Datengeneratoren Schnittstellen an den jeweils betroffenen Datengenerator weitergegeben.

Die vom Datengenerator erzeugten Daten werden zurück an das Evaluierungsprogramm übermittelt, über die betroffene Datengenerator Schnittstelle.

Erhält das Evaluierungsprogramm die generierten Testdaten vom Datengenerator wird ein Zeitobjekt erstellt. Dieses Zeitobjekt enthält Datum und die Uhrzeit, wann der Test gestartet wurde.

Anschließend übergibt das Evaluierungsprogramm den Code (bzw. die Testfälle), der vom Datengenerator erzeugte wurde, über die Vadalog Schnittstelle, welche im Abschnitt 2.4 genauer beschrieben wird, an die von der Universität zur Verfügung gestellte Vadalog Ausführung, damit diese den Code (bzw. die Testfälle) ausführt und die Performanceergebnisse zurückliefert.

Die vom Benutzer eingegeben Daten und auch die Performanceergebnisse des Testfalles werden vom Evaluierungsprogramm an die Schnittstelle zur Datenspeicherung übergeben. Diese Schnittstelle wird in Abschnitt 2.5 genauer erläutert.

## Datengenerator CBR

Wünscht der Benutzer Testfälle betreffend CBR, erhält der Datengenerator vom Evaluierungsprogramm über seine zugehörige Schnittstelle die Informationen, wie viele Testdaten er generieren soll. Diese geforderten Testdaten werden vom Datengenerator generiert.

Die Generierung der CBR Testdaten erfolgt mithilfe eines selbst implementierten Random Wort Generator.

Die konkrete Implementierung dieses Wort Generators sieht folgendermaßen aus:

**public** **class** GeneratorRandomString {

**public** **static** String getRandomString(**int** length) {

String ALPHABET = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

**final** SecureRandom RANDOM = **new** SecureRandom();

StringBuilder sb = **new** StringBuilder();

**for** (**int** i = 0; i < length; ++i) {

sb.append(ALPHABET.charAt(RANDOM.nextInt(ALPHABET.length())));

}

**return** sb.toString();

}

}

Die oben gezeigte Implementierung kann bei der Implementierung des gesamten Evaluierungsprogrammes noch abgeändert werden.

Diese Implementierung liefert ein Wort, bzw eine zufällig aneinander gereihte Variation von Buchstaben, mit der durch den Parameter bestimmten Länge.

Für die Implementierung der Testdaten wird eine Zufallszahl zwischen 4 und (exklusive) 8 ausgewählt, welche die Länge der Testdaten bestimmen soll, um zu vermeiden, dass alle generierten Wörter dieselbe Länge haben.

Diese Zufallszahl wird folgendermaßen, mithilfe einer Java Bibliothek, ThreadLocalRandom, generiert:

**int** randomNumber = ThreadLocalRandom.*current*().nextInt(4, 8);

Der CBR Testcode wird durch eine Methode generiert. Diese Methode ruft weitere Hilfsmethoden auf, um eine Übersichtlichkeit und Strukturiertheit des Codes zu gewährleisten. Jede Hilfsmethode generiert einen Abschnitt des gesamten CBR Testcodes.

Die Hauptmethode kann daher wie folgt aussehen:

**private** **static** String *CBRCode*;

**public** **static** String generateCBRCode(**int** parameters) {

*CBRCode* = "";

*CBRCode* += *generateContextClass*();

*CBRCode* += *generateParameters*(parameters);

*CBRCode* += *generateTransitiveAndTransitiveReflexiveCovers*();

*CBRCode* += *generateContextHierachy*();

*CBRCode* += *generateParameterValuesAndHierachys*();

*CBRCode* += *generateContextClass*();

*CBRCode* += *generateDetermineParemeterValues*();

*CBRCode* += *generateDetermineRelevantContextsAndMostSpecificRelevantContext*();

*CBRCode* += *generateBusinessCases*();

**return** *CBRCode*;

}

Der CBR Testcode ist daher vom Objekttyp ein String, welcher aus den String Rückgabewerten der Hilfsmethoden zusammengesetzt wird.

Die einzelnen Code Teile werden anhand von definierten Bestandteilen des Codes in Verbindung mit zufällig generierten Worten, als Parameter z.B., generiert.

Dies lässt sich am Beispiel von der Generierung der Parameter zeigen, welches wie folgt aussehen kann und in der endgültigen Implementierung noch abgeändert werden kann.

**private** **static** String generateParameters(**int** count) {

List<String> parameters = **new** ArrayList<String>();

**int** paramCount = count;

String generatedParameters = "";

**for** (**int** i = 0; i < paramCount; i++) {

// to prevent to have all parameters at the same length, a random Number generator is used.

**int** randomNumber = ThreadLocalRandom.*current*().nextInt(4, 8);

parameters.add(GeneratorRandomString.*getRandomString*(randomNumber));

}

**for** (**int** i = 0; i < parameters.size(); i++) {

generatedParameters += "hasParameter(\"aimCtx,\"" + parameters.get(i) + "\"). ";

}

generatedParameters += "\n";

**for** (**int** i = 0; i < parameters.size(); i++) {

generatedParameters += "parameter(\"" + parameters.get(i) + "\").\n";

}

generatedParameters += "\n";

**return** generatedParameters;

}

Mit dieser Methode werden auch die anderen Bestandteile des CBR Codes generiert. Die statischen Teile des Codes werden unverändert übernommen und zu dem Return String hinzugefügt.

Die Testdaten werden also vom Generator erzeugt und der fertige Test Code wird über die Schnittstelle wieder zurück an das Evaluierungsprogramm übergeben.

## Datengenerator Rule Model Inheritance

Wünscht der Benutzer Testfälle betreffend der Rule Model Inheritance, erhält der Datengenerator vom Evaluierungsprogramm über seine zugehörige Schnittstelle die Informationen, wie viele Testdaten er generieren soll. Diese geforderten Testdaten werden vom Datengenerator generiert.

Die Testdaten werden vom Generator erzeugt und der fertige Test Code wird über die Schnittstelle wieder zurück an das Evaluierungsprogramm übergeben.

Die Generierung der RMI Testdaten erfolgt mithilfe eines selbst implementierten Random Wort Generator, der im Punkt 1.3 näher beschrieben wurde.

Die Hauptmethode kann daher wie folgt aussehen:

**public** **static** String generateRMICode(**int** rules, int facts) {

*RMICode* = "";

*RMICode* += *generateProgram();*

*RMICode* += *generateRules(int rules);*

*RMICode += generateRelationalAtoms(int facts);*

*RMICode += generateNonRelationalAtoms(int facts);*

*RMICode += generateAnnotaions();*

*RMICode += generateTerms();*

**return** *RMIcode*;

}

Durch die Methode „generateProgram();“ wird der Name des Programms, mit Hilfe vom Random Wort Generator, erstellt. Dieser Name wird in Form eines Strings an den bestehenden String RMICode angehängt. In die Methode gehen keine Paramater hinein.

Durch die Methode „generateRules(int rules);“ werden die Regeln, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Die Anzahl dieser Regeln bestimmt ein Integer, der als der einzige Paramater in die Methode hineingeht. Anschließend wird diese Liste in Kombination mit dem Namen des Programms als ein String zurückgegeben.

Durch die Methode „generateRelationalAtoms(int facts)“ werden Atome, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Die Anzahl dieser Atome bestimmt ein Integer, der als der einzige Parameter in die Methode hineingeht. Anschließend wird diese Liste in Kombination mit seinem Namen (hasName), einer Regel (hasPositive/NegativeHeadAtom) (hasPositive/NegativeBodyAtom) und einem Term mit Zahl (hasArgument) als String zurückgegeben.

Durch die Methode „generateNonRelationalAtoms(int facts)“ werden Atome, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Die Anzahl dieser Atome bestimmt ein Integer, der als der einzige Parameter in die Methode hineingeht. Anschließend wird diese Liste in Kombination mit seinem Namen (hasName), einer Regel (hasPositive/NegativeHeadAtom) (hasPositive/NegativeBodyAtom), einem Term mit Zahl (hasArgument) und einem String (hasSerialisation) als String zurückgegeben.

Durch die Methode „generateAnnotations();“ werden die Annotationen, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Anschließend wird diese Liste in Kombination mit einer Regel (hasAnnotation), ihrem Namen (hasName) und einem Term mit einer Zahl (hasArgument) als String zurückgegeben.

Durch die Methode „generateTerms();“ werden die Terms, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Anschließend wird diese Liste in Kombination mit einem Literal (hasSerialization) als String zurückgegeben.

## Externer Speicher

Im externen Speicher werden die Ergebnisse der Performanceevaluierung gespeichert. Zusätzlich werden die vom Benutzer geforderten Testdaten hier persistiert, um die Tests nachvollziehen zu können und gegebenenfalls die Testergebnisse für spätere Evaluierungsarbeiten wiederverwenden zu können. Somit besteht auch die Möglichkeit, nicht nur ein Testergebnis mehrerer Durchläufe zu analysieren, sondern auch zeitlich unabhängige Tests untereinander vergleichen zu können.

Bei dem externen Speicher handelt es sich um eine mySQL Datenbank (Version 5,7), welche von einem externen Hoster zur Verfügung gestellt wird, und somit grundsätzlich dauerhaft verfügbar (99,9% Verfügbarkeit) ist. Dadurch ist es möglich, mithilfe des Evaluierungsprogramms die gewünschten Daten extern zu speichern, um sie persistent zu erhalten, wenn das Programm geschlossen wird. Durch die Datenbank ist ebenso der Mehrbenutzerbetrieb, sowie eine programmunabhängige Speicherung der Daten möglich. Die Einrichtung und Bearbeitung der Datenbank und Tabellen erfolgt über das phpMyAdmin Interface (Version 4.8.3) des Hosters.

Adresse: <https://e42776-phpmyadmin.services.easyname.eu>

Server: e42776-mysql.services.easyname.eu

Benutzer: u48005db23

Datenbank: u48005db23

Passwort: prdke2018

Die Daten werden über die Speicherung Schnittstelle, genauer behandelt in Abschnitt 2.5, vom Evaluierungsprogramm an den Speicher übergeben. Nachfolgend wird die Struktur der Tabelle, genauer der Spalten, erklärt:

|  |  |  |  |
| --- | --- | --- | --- |
| Name der Spalte | Datentyp\* | Größe | Kurzbeschreibung |
| id | INT | 3 | die einzigartige ID des durchgeführten Tests |
| day | INT | 2 | Tag des Datums der Durchführung des Tests |
| month | INT | 2 | Monat des Datums der Durchführung des Tests |
| year | INT | 4 | Jahr des Datums der Durchführung des Tests |
| hour | INT | 2 | Stunde des Datums der Durchführung des Tests |
| minute | INT | 2 | Minute des Datums der Durchführung des Tests |
| second | INT | 2 | Sekunde des Datums der Durchführung des Tests |
| testtype | VARCHAR | 15 | der durchgeführte Testtyp (CBR o. RMI Variante) |
| noParam | INT | 2 | Anzahl der Parameter |
| noRules | INT | 2 | Anzahl der Regeln |
| noFacts | INT | 2 | Anzahl der Fakten |
| exTime | INT | 4 | Dauer der Ausführung |
| noErr | INT | 2 | Anzahl der Fehlermeldungen |
| noErrSec | INT | 2 | Anzahl der Fehlermeldungen je Sekunde |
| cpuUsage | INT | 2 | durchschn. Prozessorauslastung in Prozent |

\*Datentyp und Schreibweise gemäß mySQL

# Schnittstellen

Um es dem Programm zu ermöglichen, mit den einzelnen Komponenten zu kommunizieren, sind folgende Schnittstellen notwendig und zu implementieren. Die benötigten Schnittstellen sind in der folgenden Grafik, in Abbildung 2, orange-rot umrahmt.



Abbildung 2: Schnittstellen

Um eine funktionierende Kommunikation zwischen den Systemkomponenten zu garantieren, werden die in Abbildung 2 aufgezeigten Systemschnittstellen implementiert. Genauer beschrieben werden die notwendigen Schnittstellen im nachfolgenden Abschnitt.

## User Schnittstelle

Die User Schnittstelle ist die Verbindung zwischen dem Benutzer des Programmes und des Evaluierungsprogramm, dem Mittelpunkt des Systems. Die Eingabe vom Benutzer erfolgt über die Konsole, wobei die grafische Darstellung vom User Interface nicht zum Ziel gehört.

Der User wählt aus, welche Art von Testfälle er durchführen möchte.

Die Kommunikation zwischen User und Programm erfolgt über die Konsole. Die Konsole gibt dem User die Auswahl, welche Testfälle dieser durchführen möchte:

1 CBR

2 Rule Model Inheritance

Der User gibt die Zahl, als Integer Wert, ein, welche Art von Testfälle dieser durchführen möchte.

Basierend auf der Auswahl des Nutzers, verläuft das Programm wie in den folgenden zwei Abschnitten beschrieben.

Die vom User getroffene Auswahl über CBR oder Rule Model wird an das Evaluierungsprogramm übergeben.

### CBR

Hat der User als gewünschte Art der Testfälle 1 CBR gewählt, wird dieser erneut nach einer Auswahl gefragt, um die Anzahl der Parameter, Parameter Values, Contexte und Business Cases festzulegen.

Parameter: (Integer Eingabe User)

Parameter Values: (Integer Eingabe User)

Contexte: (Integer Eingabe User)

Business Cases: (Integer Eingabe User)

Die Eingabe wird von der User Schnittstelle an das Evaluierungsprogramm übergeben.

### Rule Model Inheritance

Hat der User als gewünschte Art der Testfälle 2 Rule Model Inheritance gewählt, wird dieser erneut nach einer Auswahl gefragt, welche der Rule Model Inheritance Testfälle durchgeführt werden soll.

1 abstractionOnly

2 conformanceOnly

3 dynamicBehavioralDetectionOnly

4 inheritanceOnly

5 staticBehaviouralDetectionOnly

6 structuralDetectionOnly

Der User gibt die Zahl, als Integer Wert, ein, welche Art von Testfälle dieser durchführen möchte.

Basierend auf der Auswahl des Users, verläuft das Programm wie in den folgenden Abschnitten beschrieben. Die vom User getätigten Eingaben in den folgenden Abschnitten werden anschließend an das Evaluierungsprogramm übergeben.

#### abstractionOnly

Um diese Art der Testfälle durchzuführen wird als Input ein Module gefordert, für das alle Vererbungsbeziehungen aufgelöst wurden – Resolved Module.

Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, welche der User als Zahl, Integer, eingeben soll.

Rules: (Integer Eingabe User)

Facts: (Integer Eingabe User)

#### conformanceOnly

Um diese Art der Testfälle durchzuführen werden als Input erkannte/gemeldete Änderungsvorgänge und Modifikationsbeschränkungen gefordert – detected/reported modification and operations modification restrictions.

Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, welche der User als Zahl, Integer, eingeben soll.

Rules: (Integer Eingabe User)

Facts: (Integer Eingabe User)

#### dynamicBehavioralDetectionOnly

Um diese Art der Testfälle durchzuführen werden als Input Meta-Repräsentationen von Resultset-Paaren (Eltern- und Child-Resultsets) und Vererbungsbeziehungen von Modulen, für die Ergebnismengen angegeben sind gefordert – Meta-representations of resultset pairs (parent and child resultsets) and inheritance relations of modules for which resultsets are given.

Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, welche der User als Zahl, Integer, eingeben soll.

Rules: (Integer Eingabe User)

Facts: (Integer Eingabe User)

#### inheritanceOnly

Um diese Art der Testfälle durchzuführen werden als Input ein Modul, das abgeleitet werden soll (toDerive) eine Modul-Metadarstellungen des relevanten Vererbungsbaums gefordert – module to be derived (toDerive) and module meta-representations of relevant inheritance tree.

Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, welche der User als Zahl, Integer, eingeben soll.

Rules: (Integer Eingabe User)

Facts: (Integer Eingabe User)

#### staticBehaviouralDetectionOnly

Um diese Art von Testfälle durchzuführen wird als Input eine Meta-Repräsentationen von Modulen im Vererbungs-Teilbaum gefordert – Meta-representations of modules in inheritance subtree.

Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, welche der User als Zahl, Integer, eingeben soll.

Rules: (Integer Eingabe User)

Facts: (Integer Eingabe User)

#### structuralDetectionOnly

Um diese Art von Testfälle durchzuführen wird als Input eine Metadarstellungen des relevanten Modulvererbungsbaums gefordert – Meta-representations of relevant module inheritance tree.

Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, welche der User als Zahl, Integer, eingeben soll.

Rules: (Integer Eingabe User)

Facts: (Integer Eingabe User)

## Rule Model Inheritance Datengenerierung Schnittstelle

Diese Schnittstelle dient als Verbindung zwischen dem Rule Model Inheritance Datengenerator und des Evaluierungsprogramms.

Das Evaluierungsprogramm ruft über diese Schnittstelle den Rule Model Inheritance Datengenerator auf. Dabei werden die vom Benutzer eingegeben Daten, die Anzahl der Rules und die Anzahl der Facts, welche im Evaluierungsprogramms gespeichert wurden, über diese Schnittstelle an den Datengenerator übergeben.

Basierend auf diesen Eingaben generiert der Datengenerator die gewünschten Testdaten, dies wird im Abschnitt 1.2 genauer erläutert.

Die vom Generator generierten Daten werden anschließend über diese Schnittstelle an das Evaluierungsprogramm übergeben.

## CBR Datengenerierung Schnittstelle

Die Schnittstelle für die CBR Datengenerierung ist die Verbindung zwischen dem CBR Datengenerator und dem Evaluierungsprogramm.

Das Evaluierungsprogramm ruft über diese Schnittstelle den CBR Datengenerator auf. Dabei werden die vom Benutzer eingegeben Daten, die Anzahl der Rules und die Anzahl der Facts, welche im Evaluierungsprogramms gespeichert wurden, über diese Schnittstelle an den Datengenerator übergeben.

Basierend auf diesen Eingaben generiert der Datengenerator die gewünschten Testdaten, dies wird im Abschnitt 1.3 genauer erläutert.

Die vom Generator generierten Daten werden anschließend über diese Schnittstelle an das Evaluierungsprogramm übergeben.

## Vadalog Schnittstelle

Die Vadalog Schnittstelle ist die Verbindung zur Ausführung des generierten Codes. Diese Schnittstelle, wird von der Universität zur Verfügung gestellt.

Der generierte Code wird zur Ausführung übergeben, nach Beendigung der Durchführung des Tests erhält das Evaluierungsprogramm über diese Schnittstelle eine Information darüber und kann somit den Endzeitpunkt des Testfalles feststellen und damit die Durchführungsdauer des Testfalles ermitteln.

Bei der Implementierung unseres Programms wird ein Dummy anstatt der Vadalog Schnittstelle implementiert.

Im Programm wird eine Dummy Methode callVadalog() implementiert. Diese gibt die Performancewerte der Code Ausführung zurück.

Im speziellen werden folgende Werte zurückgegeben:

* Die Ausführungsdauer in Sekunden
  + Wird durch einen Random Zahlengenerator erzeugt
  + Datentyp: Double
* Die Anzahl der Fehler
  + Werden durch einen Random Zahlengenerator erzeugt
  + Datentyp: Integer
* Die durchschnittliche Anzahl der Fehler pro Sekunde
  + Die Gesamtanzahl der Fehler wird durch die Ausführungszeit dividiert
  + Datentyp: Double
* Die durchschnittliche Auslastung des Prozessors in Prozent
  + Wird durch einen Random Zahlengenerator erzeugt
  + Datentyp: Double

## Speicherung Schnittstelle

Mithilfe der Speicherung Schnittstelle werden die Performance Ergebnisse vom Evaluierungsprogramm an den externen Speicher übergeben.

Diese Verbindung wird durch eine JDBC Verbindung (Java Database Connectivity) ermöglicht. Die zu speichernden Daten werden mithilfe von JDBC in der SQL Datenbank des externen Speichers persistent erhalten.

Die Schnittstelle zur Speicherung wird vom Evaluierungsprogramm aufgerufen um Daten zur Speicherung an die Datenbank zu übergeben.

Konkret werden folgende Daten an die Datenbank zur Speicherung übergeben:

* Eine einzigartige ID des Tests
  + ID
    - Datentyp: Integer
* Das Datum des Durchführungszeitpunkt
  + Tag des Datums
    - Datentyp: Integer
  + Monat des Datum
    - Datentyp: Integer
  + Jahr des Datums
    - Datentyp: Integer
* Die Uhrzeit zum Durchführungszeitpunkt
  + Stunde der Ausführung
    - Datentyp: Integer
  + Minute der Ausführung
    - Datentyp: Integer
  + Sekunde der Ausführung
    - Datentyp: Integer
* Der Testtyp, welchen der Benutzer auswählt
  + Testtyp
    - Datentyp: String
* Die Anzahl der zu testenden Elemente
  + Anzahl der Parameter
    - Datentyp: Integer
  + Anzahl der Regeln
    - Datentyp: Integer
  + Anzahl der Fakten
    - Datentyp: Integer
* Diverse Testergebnisse
  + Dauer der Ausführung
    - Datentyp: Integer
  + Anzahl der Fehlermeldungen
    - Datentyp: Integer
  + Anzahl der Fehlermeldungen je Sekunde
    - Datentyp: Integer
  + Durchschnittliche Prozessorauslastung
    - Datentyp: Integer

# Umsetzung

Die Umsetzung der Anforderungen erfolgt mit der Programmiersprache Java JDK 8 (oder durch eine neuere Version). Als Entwicklungsumgebung wird Eclipse herangezogen. Zur Unterstützung der Umsetzung der Datengeneratoren und des Evaluierungsframeworks werden keine Frameworks benutzt, da keine für die Aufgabenstellung passende Frameworks bei der Recherche gefunden wurden.

Um die Umsetzung zu unterstützen werden passende Java Bibliotheken importiert und verwendet.

Die Datenspeicherung wird mithilfe einer mySQL Datenbank ermöglicht und die Verbindung zwischen dem Evaluierungsprogramm wird mithilfe von JDBC umgesetzt, um die zu speichernden Daten an die im Abschnitt 1.4 Externer Speicher, beschriebene Datenbank übermittelt.

Die geplante konkrete Umsetzung er Implementierung und der geplanten Klassen wird im nachfolgenden Abschnitt erläutert.

## Klassenstruktur

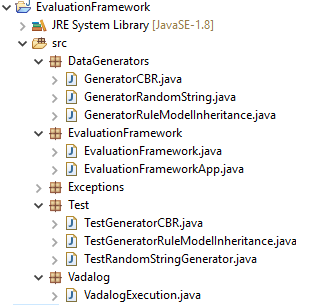
Das Evaluierungsframework wird als Java Projekt „EvaluationFramework“ implementiert. Die geplante Klassenstruktur ist in der nachfolgenden Grafik, Abbildung 3, dargestellt.

Abbildung 3: geplante Klassenstruktur

Die Implementierung wird geplant aus 5 Packages bestehen:

* DataGenerators
* EvaluationFramework
* Exceptions
* Tests
* Vadalog

Im Folgenden werden die Packages und die dazugehörigen geplanten Klassen beschrieben.

### DataGenerators

In diesem Package befinden sich die beiden Datengeneratoren, den Datengenerator für CBR und den Datengenerator für Rule Model Inheritance. Sie bilden die Funktionalität der oben beschriebenen Systemkomponenten der Datengeneratoren ab. Jeder der beiden Datengeneratoren erhält eine Klasse, um die Funktionalität der Codegeneration durch strukturierte Methoden abbilden zu können.

Ebenso befindet sich in diesem Package der selbst implementierte RandomStringGenerator, welcher einen beliebig langen String aus Groß- und Kleinbuchstaben, bestimmt durch den mitgegebenen Parameter, erstellt. Dieser wird für die Codegeneration benötigt.

### EvaluationFramework

In dem Package „EvaluationFramework“ befindet sich das Herzstück des Programmes, die Umsetzung der Systemkomponente des Evaluierungsprogrammes.

In der Klasse „EvaluationFramework“ werden die Funktionalitäten des Evaluierungsprogrammes realisiert. Die Klasse „EvaluationFrameworkApp“ beinhaltet die ausführbare Main Methode, welche die Funktionen aufruft und auch als Schnittstelle zum User agiert und die Eingaben des Users entgegennimmt.

### Exceptions

In dem Package „Exceptions“ werden alle Excpetions implementiert und verwaltet, welche für das Fehler Handling notwendig sind. Es sollen so z.B. ungültige Eingaben vom User verhindert werden können.

### Tests

In diesem Package werden die Tests für die Generatoren durchgeführt und verwaltet. Um die richtige Funktionalität zu gewährleisten, müssen die Datengeneratoren strukturiert getestet werden.

### Vadalog

In dem „Vadalog“ Package werden die Dummy Methoden implementiert, welche die Performanceergebnisse simulieren sollen. Diese Ergebnisse werden durch mehrere Methoden simuliert und in der Klasse „VadalogExecution“ implementiert.

# Abbildungsverzeichnis

[Abbildung 1: Systemkomponenten 2](file:///C:\Users\Anwender\Documents\GitHub\258321_DKE_PR\Dokumente\Konzeptueller%20Entwurf\Konzeptueller_Entwurf.docx#_Toc531003258)

[Abbildung 2: Schnittstellen 5](file:///C:\Users\Anwender\Documents\GitHub\258321_DKE_PR\Dokumente\Konzeptueller%20Entwurf\Konzeptueller_Entwurf.docx#_Toc531003259)

[Abbildung 3: geplante Klassenstruktur 10](file:///C:\Users\Anwender\Documents\GitHub\258321_DKE_PR\Dokumente\Konzeptueller%20Entwurf\Konzeptueller_Entwurf.docx#_Toc531003260)