HAW HAMBURG

BACHELORTHESIS
Ömer Kirdas

Analyse und Vergleich zweier Online-Plattformen für die Bearbeitung von Programmieraufgaben

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Ömer Kirdas

Analyse und Vergleich zweier Online-Plattformen für die Bearbeitung von Programmieraufgaben

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Axel Schmolitzky
Zweitgutachter: Prof. Dr. Jens von Pilgrim

Eingereicht am: 31.05.2021

Ömer Kirdas

Thema der Arbeit

Analyse und Vergleich zweier Online-Plattformen für die Bearbeitung von Programmieraufgaben

Stichworte

ArTEMiS, JACK, OPPSEE, Online Programmieren lernen, Automatische Feedbackgenerierung, Programmieraufgaben

Kurzzusammenfassung

Im Rahmen dieser Arbeit werden die Plattformen JACK und ArTEMiS analysiert und miteinander verglichen. Diese Plattformen wurden entwickelt, damit Lehrende Aufgaben und Kurse darin verwalten können. Der Schwerpunkt der Plattformen liegt bei der Feedbackgenerierung, die automatisch erzeugt wird, sobald der Student eine Lösung für die Aufgabe einreicht. Zu Beginn der Arbeit werden Anforderungen untersucht, die von einer Projektgruppe aus der HAW-Hamburg definiert wurden. Dieses Team hat das Ziel eine eigene Plattform zu implementieren. Mit der Erfahrung bereits vorhandener Plattformen, soll der Entwurf erleichtert werden. Bei der Analyse werden beide Plattformen unabhängig voneinander betrachtet. Dabei wird erläutert, welche Funktionen bereitgestellt werden und wie die Architektur sowie der Quellcode der Systeme aufgebaut sind. Am Ende der Arbeit werden die Anforderungen, die zu Beginn definiert wurden, als Vergleichskriterien genutzt und die Plattformen werden gegenübergestellt. Dabei werden die Unterschiede einiger Funktionen der Plattformen deutlich.

Ömer Kirdas

Title of Thesis

Analysis and comparison of two online platforms for the work on programming exercises-

Keywords

ArTEMiS, JACK, OPPSEE, learn programming online, automatic feedback generation, Programming assignments

Abstract

This thesis analyses and compares the JACK and ArTEMiS platforms. These platforms were developed so that teachers can manage assignments and courses. The focus of the platforms is on feedback generation, which is automatically generated as soon as the student submits a solution for the assignment. At the beginning of the work, requirements defined by a project group from HAW-Hamburg are researched. This team has the goal of implementing its own platform. With the experience of already existing platforms, the design should be facilitated. In the analysis, both platforms are considered independently of each other. It will be explained which functions are provided and how the architecture and the source code of the systems are structured. At the end of the thesis, the requirements defined at the beginning are used as comparison criteria and the platforms are compared. In this way, the differences between some of the functions of the platforms become clear.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

OPP	**O**nline **P**rogramming **P**latform
OPPSEE	**O**nline **P**rogramming **P**latform for **S**oftware **E**ngineering **E**ducation
JACK	**Ja**va Che**ck**er
ArTEMiS	**A**u**T**omated ass**E**ssment **M**anagement**S**ystem for interactive learning
LIA	Language-Independent Assignments
LSA	Language-Specific Assignments

1 Einleitung

1.1 Betreuung der Studierenden in der Hochschule

Damit ein Informatikstudium an der HAW Hamburg erfolgreich absolviert werden kann müssen Studierende Praktika durchführen. In den Praktika bekommen die Studierenden eine Aufgabe vom Lehrenden, die sie lösen und nach einer bestimmten Frist abgeben müssen. Bei der Abnahme achtet der Lehrende auf die Richtigkeit der Lösungen zu einer Aufgabe und entscheidet anhand von bestimmten Kriterien, ob die Aufgabe bestanden ist oder nicht. Fast jedes Modul im Studiengang der Informatik besteht aus einem Praktikums- und Vorlesungsteil, wobei ersteres eine Pflichtveranstaltung ist. Die Studierenden müssen ihre Fähigkeiten im Praktikum unter Beweis stellen und zeigen, dass sie die Konzepte aus der Vorlesung verstanden haben und praktisch anwenden können. Das erfolgreiche Bestehen des Praktikums qualifiziert die Studierenden schließlich für die Teilnahme an der Modulprüfung. Der Studierende muss somit eine Prüfungsvorleistung erbringen.

Viele Studierende haben oft Schwierigkeiten, ihre Programmierkenntnisse fortzubilden und bekommen meistens Probleme, die gestellten Aufgaben im Praktikum zu lösen. Nicht nur in der Syntax tun sich die Studierenden schwer, sondern auch darin, wie eine Programmiersprache funktioniert und wie mit ihr komplexe Aufgaben gelöst werden können. Besonders um einen wartbaren und lesbaren Code zu schreiben und eine solide Architektur zu bauen, braucht es eine Menge Übung. Auch Studierende in höheren Semestern zeigen Schwächen, in den Praktika komplexe Aufgaben zu lösen und Algorithmen in Code umzusetzen. Der Grund hierfür liegt am mangelnden Training, denn meistens wissen die Studierenden nicht, wie sie effizient lernen können und ihre Fähigkeiten weiterbilden können. Außerdem müssen vorhandene Fähigkeiten oftmals erneut auf die Probe gestellt werden, damit diese nicht verlernt werden. Die Studierenden müssten sich also mithilfe von Übungsaufgaben stets fit halten können. Doch sie

haben oftmals nicht die Möglichkeit, sofortiges Feedback für ihre Lösung zu erhalten. Den Lehrenden fehlt die Zeit, die Studierenden stets zu betreuen und ihnen Lösungshinweise zu geben.

Auch in den Praktika fehlt oft die Zeit, um zusätzliche Übungseinheiten einzubinden. Hierfür ist auch die ständig wachsende Studierendenanzahl der Grund. Die Praktika sind mit viel Arbeitsaufwand verbunden, denn der Lehrende muss die Aufgaben von jedem Studierenden effizient, fair und gerecht abnehmen.

1.2 Eine Plattform zum Bearbeiten von Programmieraufgaben

Im Folgendem geht es um **O**nline **P**rogramming **P**latform for **S**oftware **E**ngineering **E**ducation, was mit OPPSEE abgekürzt wird. Der Begriff ist während eines Projektes an der HAW entstanden. Das Ziel der Projektgruppe besteht darin, eine Plattform zu modellieren und zu implementieren, welche die Lehrenden sowie die Studierenden unterstützen soll. Anfangs legte die Projektgruppe den Namen auf OPP, also Online Programming Platform fest, der jedoch mit der Zeit als unpassend galt und deshalb zu OPPSEE geändert wurde. Denn der Name OPP könnte den Eindruck erwecken, dass es sich hierbei nur um eine Plattform handelt, in dem ein einfacher Editor für das Programmieren bereitgestellt wird, ähnlich wie Codingground[1].

Doch das Team stellt weitere Anforderungen, die in dieser Arbeit berücksichtigt werden. Beispielsweise müssen Lehrende Aufgaben bereitstellen können und die Studierenden müssen automatisches Feedback erhalten, sobald sie eine Lösung eingereicht haben.

Bei der Plattform muss der Kontext genau eingegrenzt werden. Denn es könnte dafür entwickelt werden, um die Lehre an einer Universität bzw. Hochschule zu betreiben oder nur zu unterstützen. Eine Plattform wäre dann unterstützend, wenn der Lehrende beispielsweise zusätzliches Lehrmaterial über das System dem Studierenden zur Verfügung stellt. Wenn eine Plattform jedoch dafür ausgelegt werden soll, den Unterricht zu betreiben, müssen spezifischere Anforderungen an das System gestellt werden, wie zum Beispiel die Vergabe von Noten oder das Durchführen von Prüfungen etc.

[1] https://www.tutorialspoint.com/execute_groovy_online.php

Die OPPSEE-Plattform soll unterstützend sein [1].

1.3 Erfahrung zweier Plattformen

Im Rahmen dieser Abschlussarbeit werden JACK und ArTEMiS analysiert und verglichen, um den Entwurf einer eigenen Plattform an der HAW zu unterstützen. Diese Plattformen wurden dafür entwickelt, um den Unterricht zu betreiben. Demnach unterscheiden sich die Ziele zwischen ihnen und der HAW-Projektgruppe, jedoch ähneln sich die Anforderungen. Es wird daher untersucht und verglichen, wie diese umgesetzt wurden, um aus den Erfahrungen zu lernen. Zusätzlich können bestehende Plattformen zu neuen Ideen inspirieren, deshalb werden alle möglichen Funktionen von beiden Systemen analysiert.

Die Plattformen werden regelmäßig in verschiedenen Universitäten bzw. Hochschulen für die Lehre eingesetzt. Das Hauptziel der Plattformen besteht darin dem Studierenden automatisches Feedback für ihre Aufgabenlösungen zu liefern. Außerdem erhalten Lehrende einen umfangreichen Service, um ihre Praktika bzw. Kurse zu organisieren. In beiden Plattformen können nicht nur Programmieraufgaben erstellt werden, sondern auch andere Aufgabentypen, wie beispielsweise Modellieraufgaben, die in Kurse eingebunden werden und dem Studierenden zur Verfügung gestellt werden können. So können Studierende einen Themenbereich auswählen und ihr Training durchführen, um ihre Fähigkeiten weiterzuentwickeln.

Darüber hinaus bietet jede Plattform ihre eigenen besonderen Funktionen. Beispielsweise bietet ArTEMiS die Verwaltung von Teams für Gruppenarbeiten [2] oder das Prüfen von Plagiaten. JACK hingegen ermöglicht das Visualisieren [3] einer Lösung von Studierenden oder das Prüfen von Metriken im Code.

Ganz unabhängig voneinander gehen die Entwickler der Plattformen anders vor, um ihr Ziel durchzusetzen. Die Plattformen unterscheiden sich nicht nur, wie erwähnt, in ihrem Funktionsumfang, sondern auch in ihrer Architektur, die ebenfalls in dieser Arbeit untersucht wird.

1.4 Vorgehensweise

Wie der Titel der Arbeit bereits andeutet, enthält sie einen Analyse- und einen Vergleichsteil. Bevor die Plattformen genauer betrachtet werden, müssen einige Definitionen festgelegt

werden. In Kapitel 2 werden zum einen die Anforderungen des OPPSEE-Teams thematisiert, und zum anderen wird erläutert, welche Aktoren mit der Plattform interagieren.

Danach werden die Plattformen JACK und ArTEMiS unabhängig voneinander betrachtet. Für jede Plattform gibt es ein eigenes Kapitel, in dem jeweils die Funktionen und das technische Design analysiert werden. Die Funktionsanalyse kann als praktischer Teil der Arbeit betrachtet werden, denn die Erkenntnisse werden sich größtenteils aus eigener Erfahrung mit den Plattformen ergeben. Beim technischen Design wird erklärt, wie der Quelltext aufgebaut ist und aus welchen Komponenten das System besteht bzw. welche architektonischen Konzepte verwendet werden. Im Kapitel JACK gibt es einen zusätzlichen Abschnitt für die Feedbackgenerierung, weil diese besonders umfangreich ist und es verschiedene Wege gibt, ein Feedback zu erzeugen.

Im anschließenden Kapitel 5 werden die Plattformen verglichen. Dabei wird untersucht, welche Anforderung aus Kapitel 2, von welcher Plattform erfüllt wurden. Außerdem wird diskutiert, wie diese Anforderungen jeweils umgesetzt wurden, um die Unterschiede deutlich zu machen.

2 Anforderungsanalyse

Zunächst werden Stakeholder definiert, die mit einer Lernplattform interagieren. Im Anschluss werden Anforderungen an das System festgelegt. Diese werden als Vergleichskriterien in Kapitel 5 benutzt.

2.1 Stakeholder

Bei Stakeholdern handelt es sich um Aktoren bzw. externe Entitäten, die Informationen mit dem System austauschen. Dabei können sie entweder eine Benutzerrolle oder ein anderes System darstellen [4]. Im Folgenden werden verschiedene Stakeholder definiert, die sich aus dem

OPPSEE Projekt[2] kristallisiert haben. Im Verlauf der Arbeit werden die Systeme nur noch aus den Blickwinkeln der Stakeholder 1-3 betrachtet. Es wird also analysiert, welche Funktionen jeweils für diese Aktoren bereitgestellt werden.

Die restlichen Stakeholder werden definiert, um sich die verschiedenen Zuständigkeitsbereiche vor Augen zu führen.

1. **Student:** Die Rolle des Studenten bezeichnet die Person, die Aufgaben bearbeitet bzw. Lösungen für Aufgaben entwickelt. Bei der Bezeichnung „Student“ muss es sich nicht explizit um einen Studierenden handeln, welcher an einer Universität bzw. Hochschule immatrikuliert ist. Vielmehr ist ein Student eine Person, die danach strebt, neue Fähigkeiten zu erlangen, indem sie Übungsaufgaben bearbeitet. Beispielsweise kann auch ein Lehrender die Rolle des Studenten einnehmen, wenn er die beschriebenen Spezifikationen erfüllt. Die Bezeichnung „Student“ wurde gewählt, weil Studierende die Hauptzielpersonengruppe der Plattformen sind [1].

2. **Aufgabenentwickler:** Diese Rolle erstellt und entwickelt neue Aufgaben für die Plattform oder überarbeitet die Aufgabenstellung [1].

3. **Lehrender:** Ein Lehrender ist eine Person, welche die Plattform nutzt und in ihre Veranstaltung einbindet, um den Studenten die Möglichkeit zu bieten erlerntes Wissen zu praktizieren und sich darin zu üben. Der Lehrende ist für die Betreuung der Studenten verantwortlich und hat das Ziel, ihre Veranstaltung mithilfe der Plattform zu verbessern. Sie ist nicht für die Erstellung von Aufgaben verantwortlich, kann aber diese Rolle einnehmen, ebenso wie sie die Rolle des Studenten übernehmen kann, wenn sie Aufgaben bearbeitet [1].

4. **Plattformentwickler:** Der Plattformentwickler ist für die Weiterentwicklung der Software verantwortlich. Er implementiert neue Funktionen und behebt Fehler, die während des Betriebs auftreten [1].

5. **Plugin-Entwickler:** Der Plugin-Entwickler ist für die Entwicklung neuer Komponenten für verschiedene Aufgabentypen zuständig. Er ist an der Weiterentwicklung des Frontends und Backends des Systems beteiligt [1].

6. **Betreiber:** Der Betreiber ist dafür verantwortlich, alle Ressourcen bereitzustellen, die für den Betrieb des Systems notwendig sind (Server, Hardware etc.). Er stellt sicher, dass das System stets nutzbar bzw. erreichbar ist und überwacht dessen Wartung [1].

7. **Administrator:** Der Administrator ist für das Benutzermanagement zuständig. Er fügt neue Benutzer zur Plattform hinzu und entscheidet, welche Rollen diese bekommen

[2] https://git.haw-hamburg.de/oppsee-devls/oppsee-mvp/-/tree/srs_start/srs

sollen und somit auch welche Rechte. Möglicherweise haben nicht alle Benutzer das Recht, eine Aufgabe zu erstellen oder an einer Aufgabe zu arbeiten [1].

8. **Editor**: Der Editor ist für das Qualitätsmanagement verantwortlich und überwacht, ob die erstellten Aufgaben die Standards erfüllen und bewertet diese [1].

2.2 Funktionale Anforderungen

Funktionale Anforderungen beschreiben die Interaktionen zwischen dem System und dessen Umgebung, d.h. seinen Benutzern (Stakeholdern) oder anderen Systemen, mit denen es interagiert [4]. Im Folgenden werden die Anforderungen in die Kategorien „Student“, „Lehrender“ und „Aufgabenentwickler“ eingeteilt. Dabei ergeben sich die Requirements aus dem OPSSEE Projekt [1], die dafür ausgelegt sind, **den Unterricht** in der Uni bzw. Hochschule zu **unterstützen**.

1. <u>Student</u>

FR1.1 Code-Editor: Der Code-Editor muss Funktionen, wie z.B. Syntaxhervorhebung, Autovervollständigung und „Suchen und Ersetzen“, wie sie von den vorhandenen Entwicklungsumgebungen (Eclipse, IntelliJ etc.) bekannt sind, unterstützen [1].

FR1.2 Bearbeitung & Einreichung: Die Aufgabe muss in mehreren Dateien bearbeitet und eingereicht werden können. Dies erhöht die Übersicht der Lösung, weil beispielsweise mehrere Klassen eines Sourcecodes nicht in einem einzigen Dokument geschrieben werden müssen.

FR1.3 Teamarbeit: Studenten müssen im Team arbeiten können. Die Plattform muss somit fähig sein, Teams zu verwalten [1].

FR1.4 Versionskontrolle: Die Plattform muss die Möglichkeit bieten, den Fortschritt in einem Versionskontrollsystem zu speichern, damit Änderungen nachvollzogen werden können [1].

FR1.5 Feedback: Das Feedback muss automatisch und schnell geliefert werden, sobald eine Lösung vom Studenten eingereicht wird. Dabei muss das Feedback verständlich sein und der Student muss daraus Lehren ziehen können. Außerdem muss es jederzeit abrufbar sein, damit der Student stets seinen Fortschritt überprüfen bzw. nachvollziehen kann. Das Feedback kann wie folgt in verschiedene Typen eingeteilt werden.

   a. **Visuelles Feedback:** Das Feedback visualisiert den Lösungsfortschritt des Studenten.

   b. **Textuelles Feedback:** Das Feedback beschreibt, warum die jeweilige Lösung falsch ist, damit die Studierenden aus ihren Fehlern lernen können.

[1]

FR1.6 Hinweise: Der Student soll während der Bearbeitung einer Aufgabe Hinweise bzw. Hilfestellungen bekommen, wenn dieser ein Problem nicht lösen kann [1].

FR1.7 Code Reviews: Den Studenten oder Teams muss es möglich sein, ihre Lösungen mit anderen zu teilen und zu vergleichen oder andere Lösungen zu bewerten [1].

FR1.8 Sprache: Der Student muss eine beliebige Programmiersprache zum Bearbeiten der Aufgabe wählen können (sofern definiert vom Aufgabenentwickler (s. FR3.3) und sofern nicht limitiert vom Lehrenden (s. FR2.1 c.)).

FR1.9 Lösungsfortschritt: Der Fortschritt der Aufgabe muss während der Bearbeitung sichtbar sein. Beispielsweise durch das Anzeigen, wie viele Tests der Aufgabe bereits bestanden sind.

FR1.10 Anmeldung: Der Student muss sich mit seinen Benutzerdaten aus der Hochschule bzw. Universität anmelden können.

2. <u>Aufgabenentwickler</u>

FR2.1 Aufgabentypen: Der Entwickler muss verschiedene Arten von Programmieraufgaben erstellen können. Diese können in folgende Kategorien aufgeteilt werden.

   a. **Einzelaufgaben:** Der Student bearbeitet die Aufgabe allein.

   b. **Teamaufgaben:** Mehrere Studenten arbeiten gemeinsam an einer Aufgabe.

   c. **Wettbewerbsaufgaben:** Die Studenten treten im Wettbewerb gegeneinander an. Die Lösungen der Studenten werden verglichen und anhand verschiedener Kriterien bewertet.

[1]

FR2.2 Kategorie: Es muss möglich sein, Kategorien zu definieren, um Aufgaben mit bestimmten Inhalten oder Themen leichter auffindbar zu machen.

FR2.3 Teilen: Die entwickelten Aufgaben müssen für alle Lehrenden bereitgestellt werden können.

FR2.4 Material: Beim Erstellen einer Aufgabe muss der Entwickler aufgabenspezifische Informationen bzw. Materialien zur Verfügung stellen können (Aufgaben-beschreibung, Hinweise, Links, Literatur etc.) [1].

FR2.5 Sprache: Es muss festgelegt werden können in welchen Programmiersprachen die Aufgabe gelöst werden kann. Dabei kann die Aufgabe sprachunabhängig werden (beschrieben als LIA's in [1]).

FR2.6 Tests: Der Entwickler muss Tests implementieren und bereitstellen können, welche die Lösungen der Studenten überprüfen. Dabei können auch verschiedene Einschränkungen für den Quellcode einer Lösung definiert werden, wie zum Beispiel den Ausschluss von Arrays. (beschrieben als LSA's in [1]). Die Kriterien zum Bestehen der Aufgabe soll für alle öffentlich sichtbar sein.

3. Lehrender

FR3.1 Kursmanagement: Die Plattform muss die Möglichkeit bieten, Kurse zu erstellen, in denen der Lehrende Aufgaben einbinden und verwalten kann. Auch das Verwalten von Teammitgliedern und einzelnen Kursmitgliedern muss möglich sein.

FR3.2 Limitierungen: Der Lehrende muss Limitierungen bzw. Einschränkungen für die Studenten bei der Aufgabenbearbeitung festlegen können. Die Limitationen lassen sich wie folgt in verschiedene Kategorien einordnen.

- **a. Zeit:** Es muss möglich sein, eine zeitliche Begrenzung für die Aufgabenbearbeitung festzulegen [1] [5].
- **b. Bearbeitungsreihenfolge:** Mehrere Aufgaben in einem Kurs sollen nur in einer bestimmten Reihenfolge bearbeitet werden können [1].
- **c. Sprache:** Für jede Aufgabe kann eine beliebige Programmiersprache festgelegt werden, sofern diese in der Aufgabenerstellung definiert wurde (s. FR2.5) [1] [6].

FR3.3 Belohnung: Der Lehrende muss die Möglichkeit haben, Studenten zu belohnen (z.B. für sauberen Code oder eine besonders intelligente Lösung) [1].

FR3.4 Lösungsfortschritt beobachten: Der Lösungsfortschritt der Studierenden muss einsehbar sein.

FR3.5 Material: Es muss eine Funktion geben, die es ermöglicht nach der Erstellung der Aufgabe Material für den Studenten zur Verfügung zu stellen (Verlinken von Literatur etc.). So soll jeder Lehrende individuell für sich entscheiden können, welche Materialien für den Studenten geeignet sind.

2.3 Nicht-Funktionale Anforderungen

Des Weiteren wurden für den Studenten und Lehrenden nicht-funktionale Anforderungen vom OPSSEE Team gestellt. Diese beschreiben Aspekte des Systems, die nicht direkt mit dem funktionalen Verhalten des Systems zusammenhängen [4].

1. Student

NFR1.1 **Benutzerfreundlichkeit:** Die Aufgaben müssen leicht zugänglich sein, weshalb keine komplizierte Einrichtung oder Konfiguration erforderlich sein darf [1].

NFR1.2 **Aufgabenstellungen:** Die Aufgabenstellungen müssen den Studenten anregen, Aufgaben zu lösen. Daher sollten diese interessant, unterhaltsam, herausfordernd, verständlich und realistisch sein [1].

2. Lehrender

NFR2.1 **Benutzerfreundlichkeit:** Die Lehrenden müssen die Plattform in ihren Veranstaltungen ohne großen Arbeitsaufwand integrieren können [1].

3 ArTEMiS

Nach einer kurzen Einleitung wird der Funktionsumfang von ArTEMiS analysiert. Dabei soll die Sicht der Aufgabenentwickler, der Lehrenden und der Studenten aufgedeckt werden, d.h. welche Funktionen für die jeweiligen Aktoren zur Verfügung gestellt werden. Danach wird der technische Hintergrund der Plattform behandelt.

3.1 Einleitung

ArTEMiS wurde von Stephan Krusche an der Technischen Universität München entwickelt und beinhaltet die Bezeichnung **A**u**T**omated ass**E**ssment **M**anagement **S**ystem for interactive learning [6]. Die Plattform ermöglicht es den Studenten Quiz-, Modellierungs-, Text- oder Programmieraufgaben zu bearbeiten. ArTEMiS ist dafür ausgelegt, in der Menge der Teilnehmer

skalierbar zu sein. Dies bedeutet, dass das System eine große Anzahl an Studenten verwalten kann und stets zuverlässiges und schnelles Feedback für Lösungen von Aufgaben liefert. Im Gegensatz zur OPPSEE-Plattform, welches an der HAW entwickelt wird, ist ArTEMiS dafür gedacht, den Unterricht **zu betreiben**. Dafür bietet die Plattform auch beispielsweise Funktionen, um Prüfungen durchzuführen.

Bis jetzt wurde ArTEMiS in mehreren großen Veranstaltungen mit Hunderten von Teilnehmern erfolgreich benutzt, wobei eine Veranstaltung 1400 Studenten umfasste [6].

3.2 Analyse der Funktionalitäten

Für die Übersicht der Funktionalitäten wurden für jeden Stakeholder Use-Cases entworfen. Sie beziehen sich nicht auf die Anforderungen aus Kapitel 2, sondern wurden erstellt, während die Plattform untersucht bzw. analysiert wurde. Die gesammelten Erkenntnisse können im ArTEMiS-Guide[3] abgeglichen werden. Wichtig zu wissen ist, dass es in ArTEMiS keine Trennung zwischen Aufgabenentwickler und Lehrenden gibt. Das bedeutet, dass beide Aktoren im Grunde dieselbe Person sind und sie beide Zugriff auf alle Funktionalitäten des Systems haben. Es werden aber dennoch beide getrennt voneinander behandelt. Zusätzlich gibt es in ArTEMiS noch die Tutoren als Aktoren, welche eingeschränkte Lehrendenrechte haben. Auf den Tutor wird im Folgenden kein Bezug genommen, weil mit den Lehrendenrechten bereits alle verfügbaren Funktionen erläutert werden können. Im Verlauf der Analyse wird mitunter auf den Anhang A verwiesen, in dem sich Aufnahmen von der Benutzeroberfläche von ArTEMiS befinden.

Aufgabenentwickler

Abbildung 1 zeigt die Erstellung einer Aufgabe in ArTEMiS und Abbildung 2 das Überarbeiten dieser Aufgabenstellung (s. A.1 u. A.2). In ArTEMiS muss eine Aufgabe zwingend nach der Erstellung überarbeitet werden, weil einige Konfigurationen erst nach dem Abspeichern der Aufgabe bereitgestellt werden können.

[3] https://artemis-platform.readthedocs.io

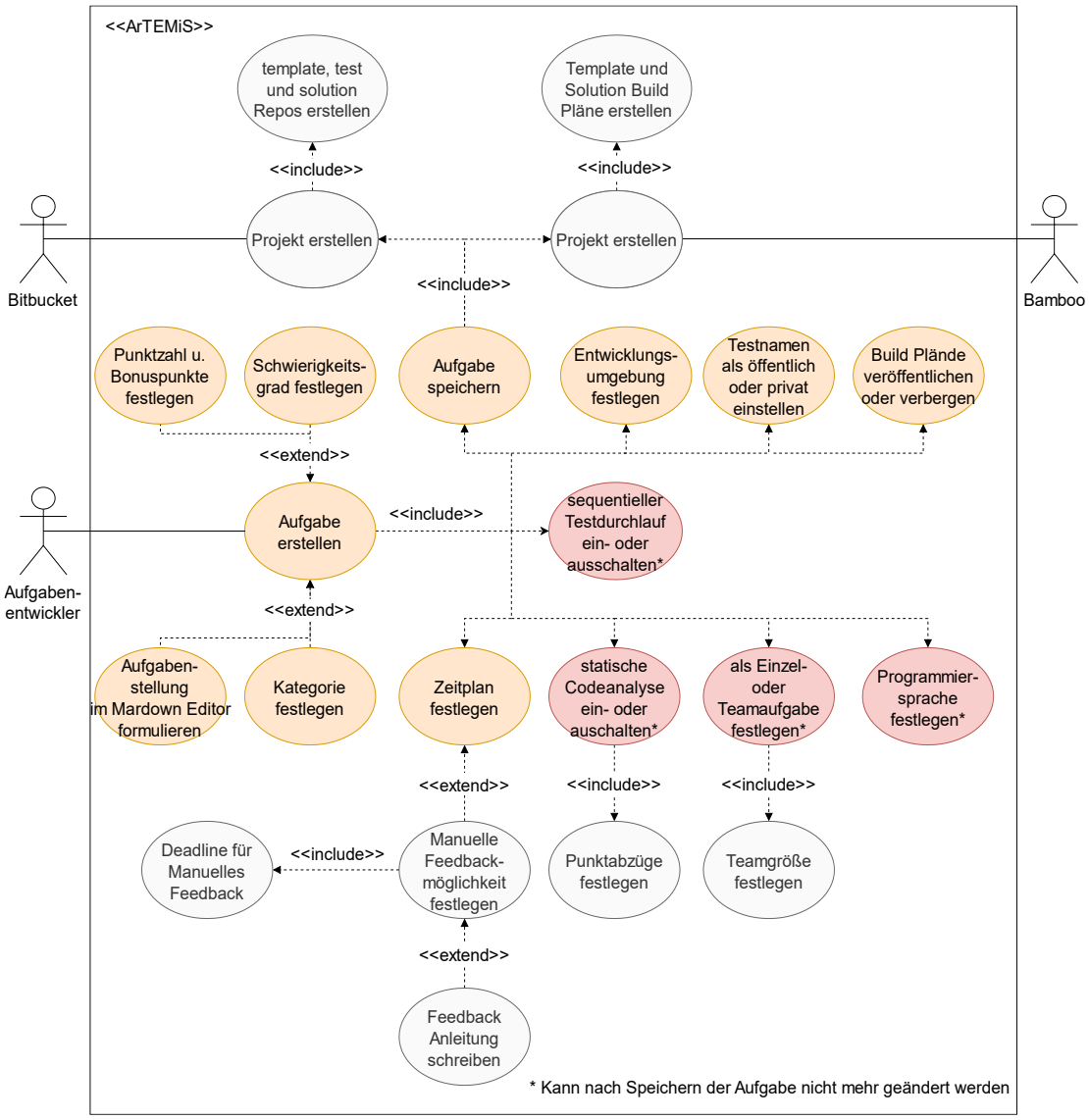


Abbildung 1: Use-Cases-Diagramm: Aufgabenentwickler in ArTEMiS. Alle roten und orangen Use-Cases haben eine Verbindung zum Use-Case „Aufgabe erstellen“. Der Rest der Use Cases sind grau, damit die Trennung übersichtlicher ist. Rot hingegen hat zusätzlich die besondere Bedeutung, dass diese Use Cases nach der Erstellung der Aufgabe nicht mehr rückgängig gemacht werden können. Die Idee mit den Aktoren Bitbucket und Bamboo stammt aus [7] [5].

In Abbildung 1 ist zu sehen, dass der Aufgabenentwickler die Wahl hat, Punkte und Bonuspunkte für die Aufgabe festzulegen. Es stellt sich die Frage, warum es explizit zwei

Punkteeinstellungen gibt. Schließlich könnten die Bonuspunkte auch einfach in die Punkte mit hinein gerechnet werden. Der Sinn dahinter ist möglicherweise, dass später der Lehrende eventuell nichts an der Punktzahl verändern möchte, wenn er mal Bonuspunkte für diese Aufgabe verteilen möchte.

Weiterhin kann der Aktor einen Schwierigkeitsgrad (Easy, Medium, Hard) festlegen. Außerdem besteht die Wahl, bis zu zwei beliebige Kategorien zu bestimmen, welche dem Studenten kurze Infos darüber geben sollen, worum es sich bei der Aufgabe handelt (s. A.16). ArTEMiS unterstützt die Sprachen Java, Python, C, Haskell, Kotlin, VHDL, Assembler, Swift und O-Caml, die beim Erstellen der Aufgabe festgelegt werden muss. Pro Aufgabe kann nur eine Sprache bestimmt werden. Die Aufgabenstellung kann in einem MarkDown-Editor formuliert werden.

Der Entwickler muss festlegen, ob die Möglichkeit bestehen soll, die Aufgabe online in einem Editor oder auf der ArTEMiS Benutzeroberfläche zu bearbeiten. Ansonsten kann die Aufgabe nur lokal bearbeitet werden, indem der Student alle nötigen Ressourcen herunterlädt und diese in eine gewohnte Entwicklungsumgebung, wie beispielsweise Eclipse, einbindet.

Der Entwickler muss in einem Zeitplan bestimmen, wann die Aufgabe veröffentlicht werden soll. Außerdem kann er eine Bearbeitungsfrist festlegen. Zusätzlich kann er entscheiden, ob nach der Bearbeitungsfrist alle Einreichungen nochmals geprüft bzw. getestet werden sollen. In dem Zeitplan besteht außerdem die Möglichkeit, manuelles Feedback einzuschalten und auch dafür eine Frist festzulegen, wann das letzte manuelle Feedback herausgegeben werden darf. Sobald das manuelle Feedback für eine Aufgabe freigeschaltet wird, besteht noch die Möglichkeit, eine Feedbackanleitung für Lehrende und Tutoren in einem zusätzlichen Mark-Down-Editor zu formulieren.

ArTEMiS unterstützt die Verwaltung von Teams. Damit ein Team von dem Lehrenden für eine Aufgabe erstellt bzw. verwaltet werden kann, muss die Funktion bei der Aufgabenerstellung freigeschaltet werden. Zusätzlich muss die maximale und minimale Teamgröße angegeben werden, die für die Aufgabe zulässig sein soll.

Sobald eine Aufgabe gespeichert wird, reagieren die Aktoren Bitbucket und Bamboo. Bitbucket erstellt ein Projekt, in dem drei verschiedene Repositories angelegt werden. In dem

Template-Repository speichert der Entwickler alle nötigen Ressourcen für den Studenten, wie beispielsweise Klassen und Methoden mit leeren Rümpfen, die dann weiter implementiert werden müssen. In dem Test Repository wird der Code (Junit Tests etc.) abgelegt, die für das Testen der Lösungen zuständig ist. In den Tests werden die Feedbacks formuliert, die auf der ArTEMiS Benutzeroberfläche dem Studenten angezeigt werden. Bei der Aufgabenerstellung kann zusätzlich bestimmt werden, ob die Namen der Testfälle dem Studenten eingeblendet werden sollen (s. A.8). Dem Solution-Repository können Musterlösungen hinzugefügt werden. Die Test- und Solution-Repositories sind für den Studenten nicht zugänglich.

Bamboo erstellt ebenfalls ein Projekt und legt einen Template-Build-Plan und einen Solution-Build-Plan an, die per Polling prüfen, ob es Änderungen in den Repositories gegeben hat. Dabei verknüpft der Template-Build-Plan das Test- und das Template-Repository und der Solution-Build-Plan das Test- und das Solution-Repository. Sobald eine Änderung (commit) im Template- oder Solution-Repository durchgeführt wird, werden die Tests ausgeführt. Sobald die Tests terminieren, werden die Feedbacks aus den Testfällen auf der Benutzeroberfläche von ArTEMiS angezeigt. Wenn sich ein Student für eine Aufgabe anmeldet, wird das Template-Repository sowie der Template-Build-Plan für den Studenten kopiert (fork). Der Solution-Build-Plan ist speziell für den Entwickler, damit dieser seine Musterlösung auf Korrektheit prüfen kann.

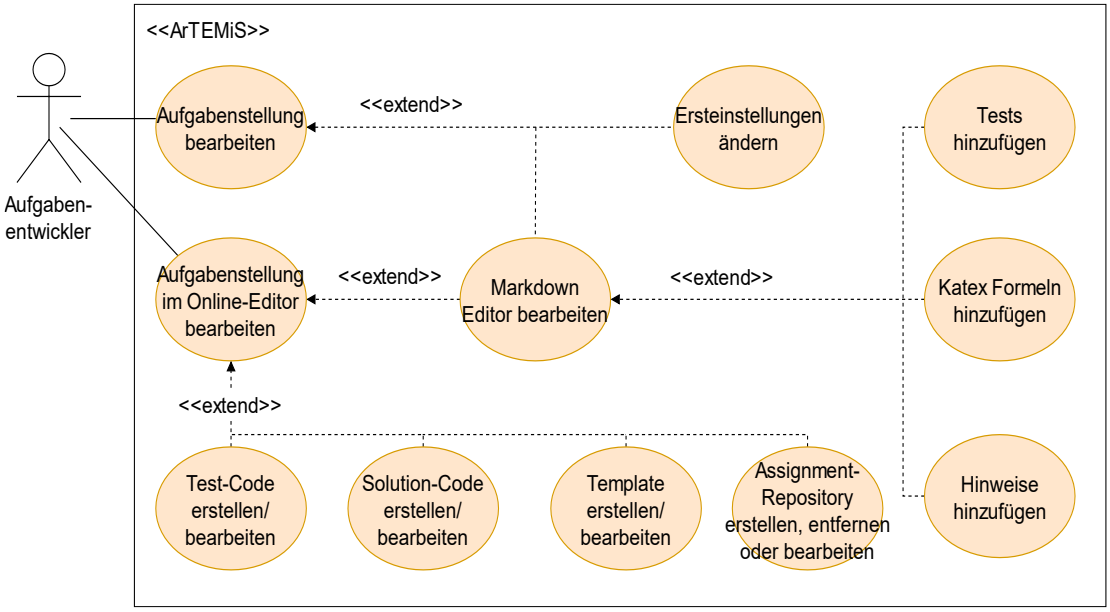


Abbildung 2: Use-Cases-Diagramm: Aufgabenentwickler in ArTEMiS.
Aufgabenstellung überarbeiten.

Das Template, sowie der Solution- und Testcode kann erst nach Speichern der Aufgabe den Repositories hinzugefügt werden, weil diese erst nach dem Speichern der Aufgabe erstellt werden. Deshalb ist in Abbildung 2 das Bearbeiten der Aufgabenstellung dargestellt. Der Entwickler würde für gewöhnlich nach dem Speichern in die Bearbeitung gehen und seine Aufgabenstellung ergänzen. Er hat zum einen die Möglichkeit, in die gewohnte Bearbeitungsumgebung zu gehen, in dem er die gewöhnlichen Konfigurationen verändern kann (s. A.1, A.2 u. Abbildung 1). Oder er bearbeitet die Aufgabe in dem Online-Editor von ArTEMiS, welche die studentische Benutzersicht darstellt (s. A.5). In diesem können jedoch nur der MarkDown-Editor oder die Repositories überarbeitet werden. Die Repositories können nur in dem Online-Editor oder lokal auf dem Rechner bearbeitet werden.

Sobald die Tests dem Test-Repository hinzugefügt worden sind, können diese ebenfalls dem MarkDown-Editor zur Aufgabenbeschreibung hinzugefügt werden. Die Testfälle können auch mit einem UML-Diagramm in Verbindung gebracht werden (s. A.2), um den Zusammenhang des Codes besser zu verdeutlichen. Wenn die Studenten ihre Lösung erfolgreich einreichen, werden die Tests in der Aufgabenbeschreibung grün angezeigt und anderenfalls rot (s. A.8) [6].

Der Entwickler kann außerdem entscheiden, wie die Lösungen des Studenten getestet werden sollen. Hierfür lässt sich bei der Aufgabenerstellung entweder die statische Codeanalyse einschalten oder der sequenzielle Testdurchlauf, jedoch nicht beide zugleich (s. Abbildung 1). Die statische Codeanalyse wird nur für die Programmiersprache Java unterstützt. Bamboo generiert die Buildpläne so, dass diese zusätzlich statische Code Analyse Tools ausführen. Die gefundenen Probleme werden kategorisch dem Studenten als Feedback zur Verfügung gestellt. Diese Funktion soll den Studenten auf Codequalitätsprobleme aufmerksam machen [8]. Der sequenzielle Testdurchlauf wird nur für die Sprachen Java, Python, Haskell oder Kotlin unterstützt. Wenn diese Funktion eingeschaltet wird, wird das Test-Repository mit einem Behavior- und Structural-Ordner erstellt. Wie der Name bereits vorhersagt werden Strukturtests in dem Ordner Structural implementiert, die immer als Erstes ausgeführt werden. Im Ordner Behavior werden Verhaltenstests implementiert, die dann als Zweites ausgeführt werden. Die Ausführungsreihenfolge der Testordner wird in den Build Plänen festgelegt. Ein Verhaltenstest würde beispielsweise testen, ob die implementierte Methode den erwarteten Rückgabewert liefert. Ein Strukturtest würde beispielsweise prüfen, ob die Methode einen korrekten Namen hat.

Neben dem Feedback, die durch die Tests erzeugt werden, können der Aufgabenbeschreibung, also dem MarkDown-Editor, noch Hinweise hinzugefügt werden [7] (s. A.13.1), die auf der Benutzeroberfläche für den Studenten zunächst verborgen sind und mit einem Klick auf ein Fragezeichensymbol abgerufen werden können (s. A.13.2). Der Hinweis wird in ArTEMiS nicht in der Bearbeitungsübersicht erstellt (s. A.1, A.2), sondern in der Aufgabenübersicht (s. A.4). Ähnlich wie bei den Tests können Hinweise erst erstellt werden, wenn bereits die Aufgabe existiert bzw. erstellt und gespeichert wurde. Die Reihenfolge wäre also *„Aufgabe erstellen -> Hinweis erstellen -> Aufgabe bearbeiten und im MarkDown-Editor den Hinweis zur Aufgabenbeschreibung hinzufügen“*.

Dem MarkDown-Editor können außerdem mithilfe von Katex[4] mathematische Formeln hinzugefügt werden.

[4] https://katex.org/

Der Entwickler hat zudem die Möglichkeit, ein neues Repository zu erstellen (Assignment-Repository). Dieses Repository ist ebenfalls nicht für den Studenten verfügbar.

Student

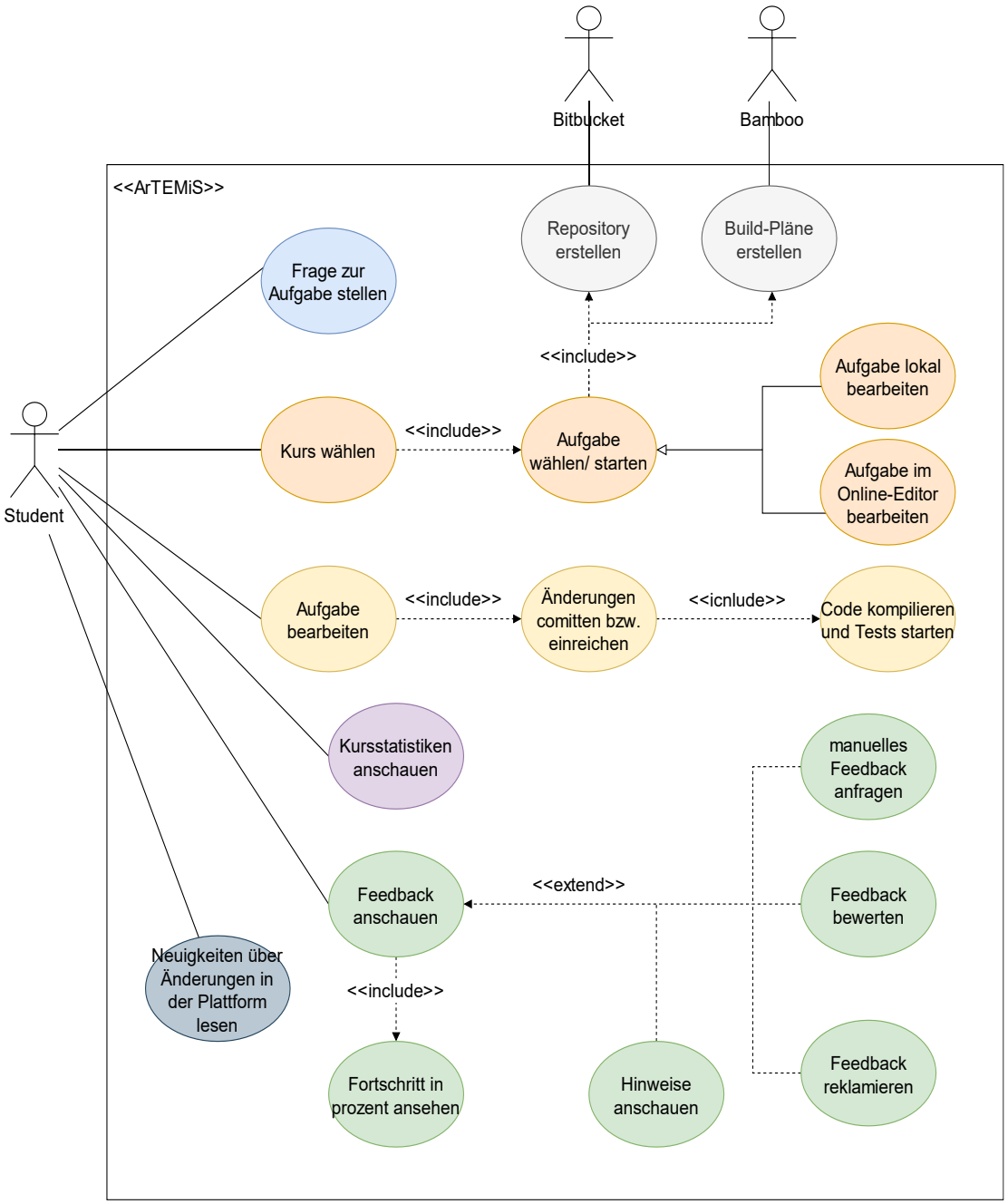


Abbildung 3 Use-Cases-Diagramm: Student in ArTEMiS

In Abbildung 3: ist zunächst zu sehen, dass der Student einen Kurs auswählen und sich dafür anmelden kann. In dem Kurs können nun unterschiedliche Aufgaben zur Auswahl stehen, wofür er sich ebenfalls anmelden und die Aufgabe starten kann. Sobald sich ein Student für eine Aufgabe angemeldet hat, erstellt Bitbucket automatisch ein Repository für den Studenten. Wie bereits erwähnt, ist das Repository eine Kopie bzw. ein Fork des Template Repositories. Außerdem erstellt Bamboo einen Build-Plan, in dem das Repository des Studenten und das Test Repository des Aufgabenentwicklers als Ziele angegeben werden. Der Build-Plan ist ebenfalls eine Kopie von dem bereits erwähnten Template Build-Plan (Base-Build-Plan in [6]).

Der Student hat nun die Möglichkeit die Aufgabe online in einem Editor auf dem ArTEMiS User-Interface zu bearbeiten, falls diese Funktion beim Erstellen der Aufgabe freigeschaltet wurde. Oder er kann lokal auf seinem Rechner arbeiten. Wenn er sich dafür entscheidet lokal zu arbeiten, würde er einfach das Repository auf seinen Rechner clonen und das Template in eine gewohnte Entwicklungsumgebung, wie zum Beispiel Eclipse, einbinden. Die Build-Pläne prüfen zyklisch nach, ob es neue Änderungen in den Repositories gibt. Sobald der Student etwas Neues pusht, werden die Tests automatisch ausgeführt. Danach kann er sich das Feedback auf dem User-Interface von ArTEMiS anschauen. Ein Commit bzw. ein Push entspricht also einer Einreichung, bei dem die Tests des Entwicklers den Code des Studenten referenzieren.

Neben dem Feedback existieren außerdem die bereits erwähnten Hinweise, die dem Studenten helfen sollen, wenn dieser bei einer Aufgabe nicht weiterkommt. Sollte dies auch nicht helfen, kann der Student jederzeit eine Frage zur Aufgabe stellen oder manuelles Feedback vom Lehrenden anfordern. Darüber hinaus können die Feedbacks vom Studenten bewertet werden. Somit bekommt der Lehrende eine Rückmeldung über seine Feedbackqualität und kann diese weiterentwickeln und verbessern. Der Student kann das Feedback auch reklamieren, falls er sich ungerecht behandelt fühlt.

Damit der Student eine Übersicht zu seinen Aufgaben hat, wird ihm eine Kursstatistik angezeigt, in dem alle Aufgaben und deren Fortschritt aufgeführt sind (s. A.14).

Wenn es zu Änderungen innerhalb der Plattform kommt (beispielsweise, wenn eine Aufgabenstellung überarbeitet wird), wird der Student mit einer Push-Nachricht auf der Benutzeroberfläche darauf aufmerksam gemacht.

Lehrender

Im Folgenden werden die Abbildungen 2-4 beschrieben, welche die Möglichkeiten des Lehrenden aufzeigen.

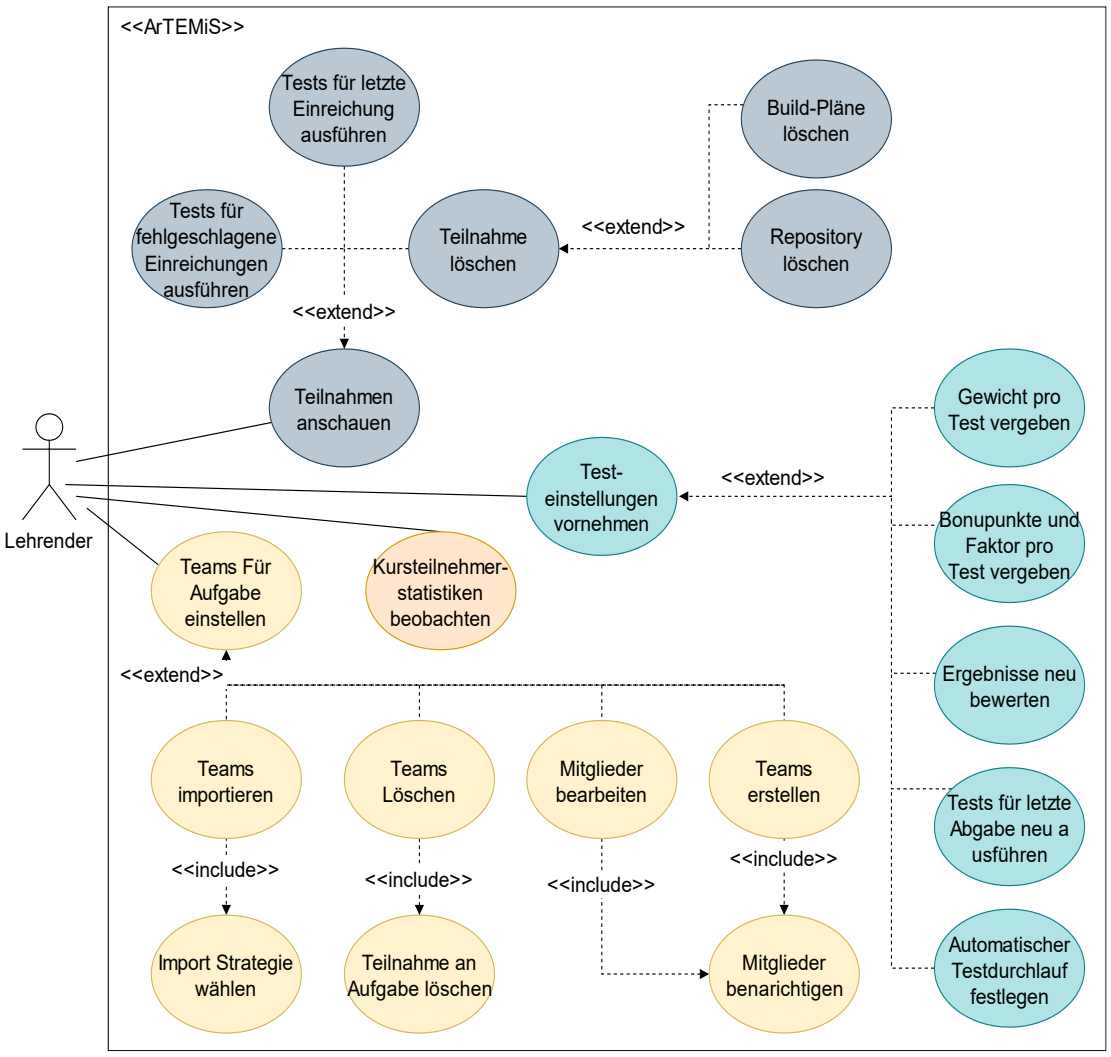


Abbildung 4: Use-Cases-Diagramm: Lehrender in ArTEMiS. Team Use-Cases aus [2].

In Abbildung 4 beschreiben die grauen Use-Cases das Beobachten der Teilnahmen von Studenten (s. A.5). Geht ein Lehrender in die Teilnahmeübersicht von ArTEMiS besteht für ihn die Möglichkeit die Teilnahme zu löschen, wobei er aussuchen kann, ob auch die Repositories sowie der Build-Plan des Studenten gelöscht werden sollen. Das bedeutet, dass der Lehrende

die Kontrolle darüber hat, die Lösungen und Teilnahmen jedes Studenten unwiderruflich zu löschen. Wie bereits bekannt ist, kann ein Student in ArTEMiS seine Ergebnisse einreichen, indem er seine Änderungen committed bzw. in sein Repository pusht. Nach dem Commit werden die Tests durch den Build-Plan automatisch ausgeführt. Es kann vorkommen, dass die Testfälle in den Test-Repositories verändert bzw. modifiziert werden, nachdem der Student seine Lösung eingereicht hat. Aus diesem Grund kann der Lehrende die Tests jederzeit neu starten, damit die neuen Tests die letzte Einreichung bzw. den letzten Commit des Studenten prüfen. Außerdem besteht noch die Möglichkeit die Tests für alle fehlgeschlagenen Einreichungen neu zu starten.

Die gelben Use Cases in Abbildung 4 betreffen das Teammanagement in ArTEMiS. Wie bereits erwähnt kann der Entwickler die Teamfunktion in ArTEMiS für eine Aufgabe einschalten. Die Aufgabe wird somit zu einer Teamaufgabe und kann ausschließlich nur von Teams bearbeitet werden. Der Lehrende hat die Möglichkeit Teams für die Aufgabe zu erstellen (s. A.8), wobei die Studenten per E-Mail benachrichtigt werden, sobald sie einem Team hinzugefügt worden sind [2]. Das gilt auch, wenn die Mitglieder aus dem Team entfernt werden. Die Teams sind immer Aufgaben zugeordnet, d.h. sie können nur Aufgaben bearbeiten, für die sie auch vom Lehrenden hinzugefügt worden sind. Beim Erstellen eines Teams kann ein Tutor zugewiesen werden, der als Betreuer agiert. Dieser kann dem Team außerdem schriftliche Feedbacks hinterlassen [2]. Der Lehrende hat nicht die Pflicht, die Anzahl der Teammitglieder, die beim Erstellen der Aufgabe festgelegt werden, einzuhalten. Nach meiner Meinung wird diese Funktion beim Erstellen der Teams dadurch völlig überflüssig. Die Teams können auch wieder gelöscht werden, wobei dann auch gleichzeitig die Teilnahme an der Aufgabe gelöscht wird. Hier hat der Lehrende nicht die Auswahl, das Repository oder den Build-Plan des Teams zu löschen, diese bleiben erhalten. Der Lehrende kann das erstellte Team in eine andere Aufgabe importieren. Somit kann der Lehrende ein Team schnell zu mehreren Aufgaben hinzufügen, anstatt für jede Aufgabe einzeln das Team neu zu erstellen.

Die türkisfarbigen Use Cases in Abbildung 2 geben eine Übersicht darüber, welche Funktionen für die Verwaltung von Testfällen zur Verfügung stehen. Der Lehrende kann sich alle Testcases auf dem User-Interface von ArTEMiS anzeigen lassen und sieht, welche aktiv bzw. inaktiv sind (s. A.10). Inaktiv sind die Tests nur dann, wenn diese aus dem Repository gelöscht werden.

In ArTEMiS gibt es nicht die Möglichkeit per Knopfdruck einen Test auszuschalten. Jedem Test kann ein Gewicht zugewiesen werden. Je höher das Gewicht eines Tests ist, desto mehr hat dieser eine Auswirkung auf das Gesamtergebnis. Würden beispielsweise zwei Tests existieren und es hätten beide dasselbe Gewicht, dann wäre die Aufgabe bei einem erfolgreichen Test zur Hälfte bestanden. Zu dem Test können noch Faktoren (Bonusmultiplizierer) sowie Bonuspunkte vergeben werden. Der Faktor wird mit der Punktzahl des Testfalls multipliziert. Dabei wird der Einfluss von anderen Testfällen auf das Gesamtergebnis nicht berücksichtigt. Faktoren, die einen Wert größer als Eins haben, können somit ein Gesamtergebnis von über 100% erzeugen (In ArTEMiS wird das Ergebnis immer in Prozent angezeigt und nicht als Punktzahl). Wenn es beispielsweise zwei Testfälle gibt und beide bestanden werden, wäre das Ergebnis bei 100%. Wenn die beiden Testfälle die gleiche Punktzahl liefern, wobei der eine aber einen Faktor von zwei hat, dann wäre das Ergebnis bei 150%. Die Bonuspunkte, die vergeben werden, werden auf die Punktzahl des Testfalls addiert. Dadurch ist auch hier, wie beim Faktor, ein Gesamtergebnis von über 100% möglich. Darüber hinaus kann für jeden Testfall bestimmt werden, ob diese nach Ablauf einer Aufgabe automatisch ausgeführt werden sollen. Zudem besteht die Möglichkeit, die Testfälle für die letzte Einreichung bzw. für das letzte Ergebnis neu zu starten (Ein Ergebnis kann vorliegen, ohne dass dieses eingereicht wurde).

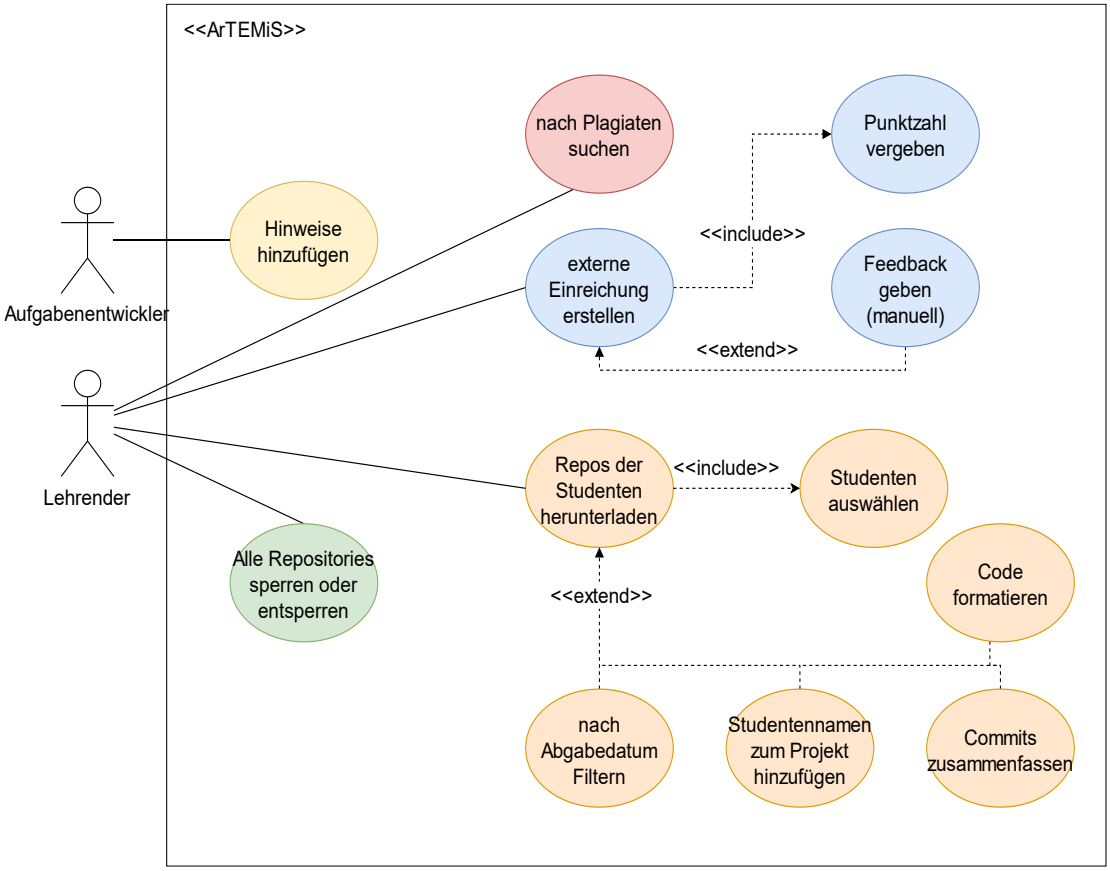


Abbildung 5: Use-Cases-Diagramm: Lehrender in ArTEMiS.

ArTEMiS bietet eine Aufgabenübersicht (s. A.4) an, in der verschiedene Funktionen für den Lehrenden bereitgestellt werden (s. Abbildung 5). Hier befindet sich zusätzlich die Funktion für den Aufgabenentwickler, Hinweise zu erstellen, die dann im MarkDown-Editor in der Aufgabenbearbeitung hinzugefügt werden können (beschrieben unter Abbildung 2).

Wie bereits bekannt ist, hat der Lehrende viele Rechte in Bezug auf die Repositories der Studenten, welche in Abbildung 5 ebenfalls deutlich werden. Repositories können gesperrt werden, d.h. es können keine Änderungen mehr eingereicht werden. Außerdem werden diese automatisch gesperrt, wenn die Aufgabe abgelaufen ist. Auch nach Ablauf der Aufgabe können die Repositories jedoch wieder entsperrt werden, sodass weitere Lösungen eingereicht werden können. Der Lehrende hat dabei aber nicht die Wahl einzelne Repositories zu sperren bzw. zu

entsperren, sondern nur alle auf einmal. Die Repositories der Studenten können heruntergeladen werden, indem die Namen der Studenten angegeben werden. Als Auswahl stehen zusätzlich verschiedene Filterfunktionen zur Verfügung. Dabei kann bestimmt werden, welches Abgabedatum (welcher Commit) beim Download berücksichtigt werden soll. Die Repositories werden als Projekt heruntergeladen, wobei die Möglichkeit besteht, die Namen der Studenten dem Projektnamen hinzuzufügen, damit mehrere Projekte in Eclipse importiert werden können (mehrere Projekt in Eclipse mit demselben Namen sind nicht zulässig). Zusätzlich kann der Code beim Download formatiert werden, d.h. auf Konventionen angepasst werden. Die Commits der verschiedenen Repositories können wahlweise zu einem Commit zusammengeführt werden.

Abgesehen von Einreichungen über das Repository per Commit kann der Student auch über andere Wege (E-Mail etc.) seine Lösung beim Lehrenden einreichen. Dadurch würden die Tests aber nicht ausgeführt werden und der Student würde kein Feedback zu seiner Lösung bekommen. Aus diesem Grund gibt es die Funktion externe Einreichungen zu erstellen (s. A.6). Dabei werden die erreichten Punkte sowie das Feedback manuell vergeben.

In ArTEMiS können Plagiate entdeckt werden, indem die Plattform die Lösungen der Studenten miteinander vergleicht. So werden Ähnlichkeiten entdeckt und der Lehrende kann sich diesen Report herunterladen.

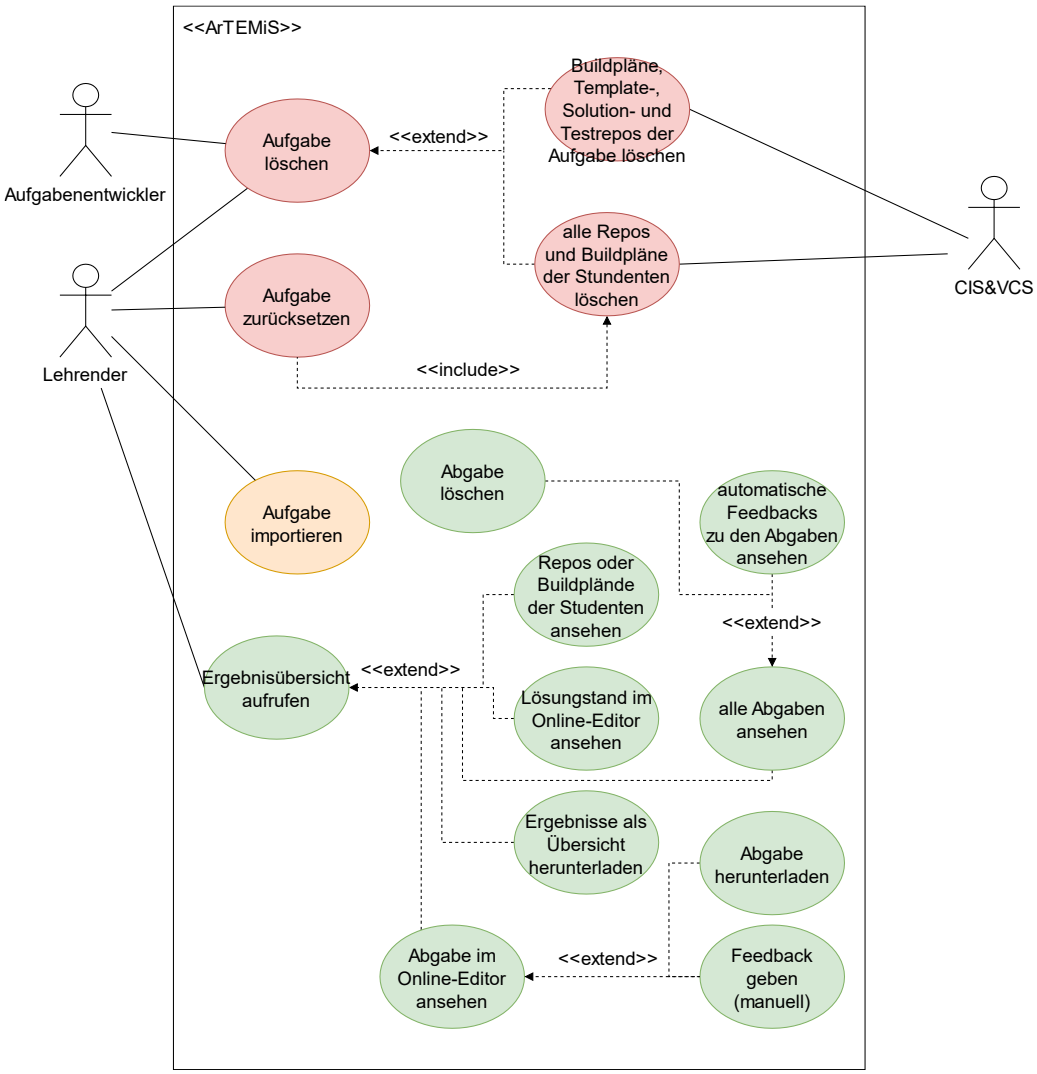


Abbildung 6: Use-Cases-Diagramm: Lehrender in ArTEMiS.

Abbildung 6 beschreibt die Use Cases in der Kursübersicht von ArTEMiS. Die roten Use Cases beschreiben das Löschen bzw. Zurücksetzen einer Aufgabe. Wenn eine Aufgabe zurückgesetzt wird, werden alle Teilnahmen an dieser Aufgabe gelöscht und somit auch alle Repositories und Build-Pläne aller Studenten. Wenn die Aufgabe jedoch gelöscht wird, hat der Lehrende die Wahl, die Repositories sowie die Build-Pläne der Stunden bestehen zu lassen. Build-Pläne und Repositories der Aufgabe können ebenfalls gelöscht werden, was in den Zuständigkeitsbereich des Aufgabenentwicklers greift. Denn die Repositories werden beim Erstellen der Aufgabe

generiert und der Aufgabenentwickler verwaltet diese und speichert darin seine Ressourcen. Da ArTEMiS Aufgabenentwickler und Lehrende nicht voneinander trennt, ist das Löschen der Aufgabe schwer in einen Zuständigkeitsbereich zu sortieren. Denn der Lehrende müsste die Chance haben, seine Kurse so zu verwalten wie er möchte und dazu gehört auch das Entfernen von Aufgaben aus dem Kurs. Jedoch wird die Aufgabe unwiderruflich gelöscht, wenn diese aus dem Kurs entfernt wird, was wiederum in den Zuständigkeitsbereich des Aufgabenentwicklers fällt. Aus diesem Grund haben beide auf der Abbildung den Zugriff auf das Use-Case „Aufgabe löschen“.

In ArTEMiS gibt es keinen Aufgabenpool, stattdessen werden Aufgaben direkt in einem Kurs erstellt und können in andere Kurse transferiert bzw. importiert werden. Sollte also eine Aufgabe in nur einem Kurs existieren und würde diese gelöscht werden, dann wäre die Aufgabe gänzlich verloren. Das Löschen von Aufgaben betrifft dennoch den Lehrenden, weil dieser die Entscheidungsmacht haben muss, welche Aufgaben in seinem Kurs verfügbar sein sollen.

ArTEMiS bietet dem Lehrenden eine Übersicht an (grüne Use-Cases in Abbildung 6), in der die Ergebnisse der Studenten aufgelistet werden (s. A.8). In dieser Übersicht kann der Lehrende zu den Repositories oder zum Online-Editor navigieren, um sich die aktuelle Lösung des Studenten anzuschauen. Er kann sich aber auch einen bestimmten Commit bzw. eine bestimmte Abgabe ansehen und diese herunterladen oder ein Feedback hinterlassen. Er kann auch alle Abgaben aller Studenten ansehen und sich die automatischen Feedbacks zu den Lösungen anschauen. Außerdem hat er die Wahl einzelne Abgaben zu löschen.

3.3 Analyse des technischen Designs

3.3.1 Datenmodellierung

Im Folgenden wird die Datenmodellierung mithilfe eines Klassendiagrammes erklärt, das in Abbildung 7 dargestellt ist.

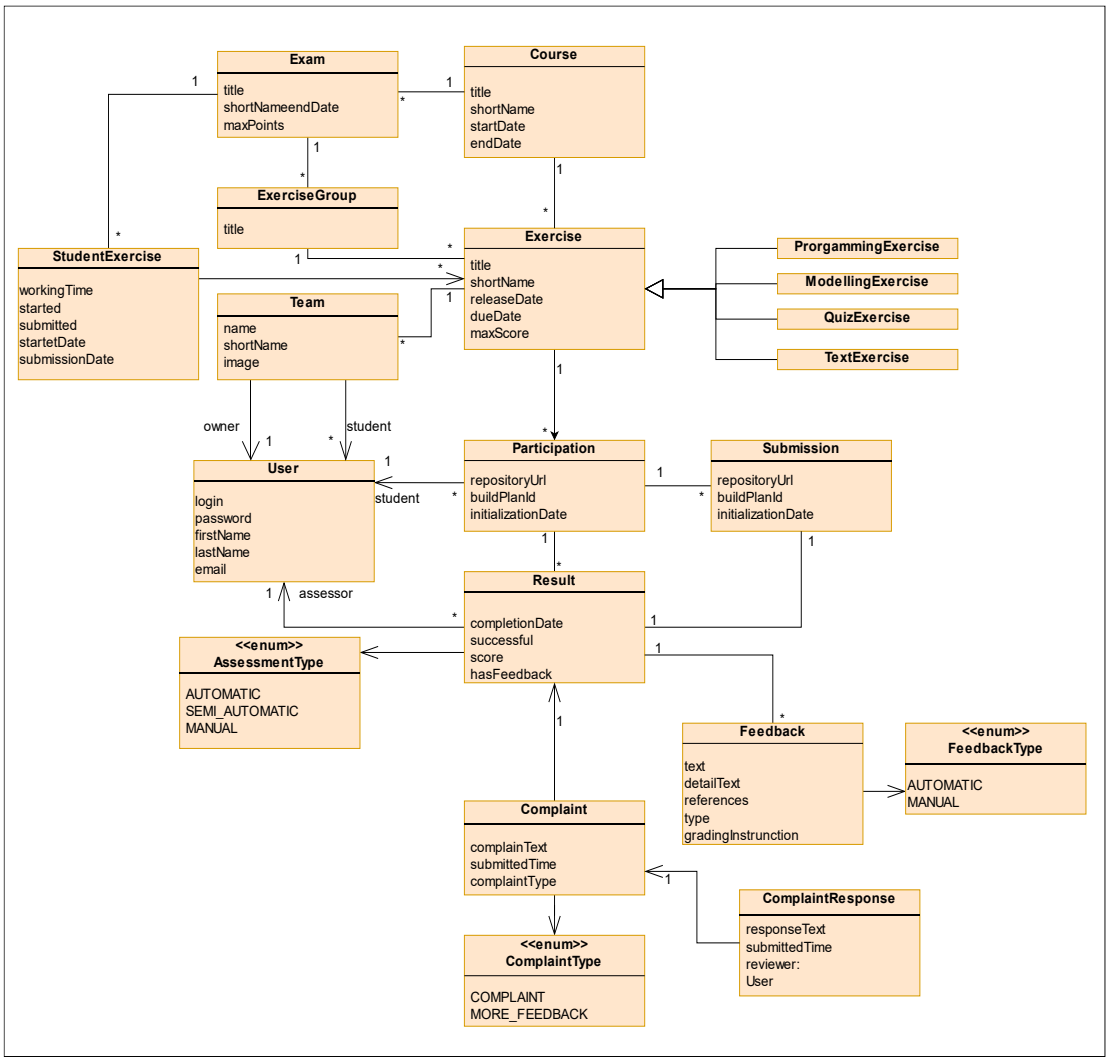


Abbildung 7: ArTEMiS Klassendiagramm: Für die Übersichtlichkeit wurden nicht alle Exemplarvariablen bzw. Instanzvariablen der Klassen dargestellt. Durchgezogene Striche ohne Pfeil sind eine Bidirektionale „enthält“ Beziehung. Die Pfeilbeschriftungen beschreiben die Kardinalitäten. [9]

Ganz oben angefangen kann eine *Course* Klasse mehrere *Exercise* und *Exam* Objekte enthalten, wobei verschiedene Klassen von *Exercise* erben. Das bedeutet, dass die unterschiedlichen Aufgabentypen, wie zum Beispiel *ProgrammingExercise*, *ModellingExercise* etc., Subklassen von *Exercise* sind und diese somit repräsentieren. Wie bereits in dem vorherigen Kapitel erwähnt, kann der Lehrende einer Aufgabe bzw. *Exercise* mehrere Teams zuweisen, wobei ein

Team mehrere *User* (Studenten, Tutoren) enthalten kann. Die *Participation* (Teilnahme) wird erstellt, sobald ein Student eine Aufgabe startet. Sollte der Student einem Team zugeordnet sein wird eine *Participation* für das Team erstellt, d.h. jedes Mitglied kann für das Team eine *Participation* erzeugen. Einer *Exercise* sind mehrere *Participation* Objekte zugeordnet, wobei diese *Submission* (Einreichung), *User* und *Result* (Ergebnis) enthalten. *Result* enthält ebenfalls die *Submission* und den *User*, wobei es sich bei dem *User* um den Lehrenden bzw. den Prüfer handelt. Die Kardinalitäten zwischen *User* und *Result* bzw. *User* und *Participation* beschreiben, dass mehrere *Result* bzw. *Participation* Klassen denselben User enthalten. Zusätzlich enthält *Result* das Enum *AssessmentType* und N Objekte von der Klasse Feedback. Feedback enthält den FeedbackType, welcher ähnlich wie *AssessmentType* beschreibt, ob die Bewertung bzw. das Feedback automatisch oder manuell erfolgt ist. Ganz unten ist noch die Klasse *Complaint*, welcher *Result* enthält. *Complaint* kann entweder eine Beschwerde zu einem Ergebnis sein oder ein Antrag auf mehr Feedback zu dem Ergebnis.

3.3.2 System Architektur

Im Folgenden wird die Architektur von ArTEMiS analysiert und das Zusammenspiel der wichtigsten Komponenten des Systems beschrieben. ArTEMiS verwendet das Konzept der Client Server Architektur. Die oberste Hierarchie ist in Abbildung 8 dargestellt.

Unten links ist das User (Student, Lehrender etc.) Device dargestellt, welches die Komponenten Version-Control-Client und ArTEMiS-Application-Client enthält. Das lokale Repository des Studenten ist mit dem Versions Control Server verbunden, der auf der Infrastruktur der Universität gehostet wird. Der ArTEMiS-Application-Client ist direkt mit dem ArTEMiS-Application-Server verbunden, der ebenfalls auf der Infrastruktur der Universität verwaltet wird. Die Application-Client- und Server-Komponenten werden im Folgenden genauer betrachtet, diese sind blau markiert. Auf der Infrastruktur befinden sich zusätzlich noch der Continuous-Integration-Server, der seine Aufträge an den lokalen Build-Agent weiterleitet. Sollte die Last im System zu hoch sein, können die Aufträge zusätzlich zu einem Externen Build-Agent weitergeleitet werden, der auf einem Amazon web Service basiert. ArTEMiS verwendet außerdem das User Management System der Universität, d.h. jeder kann sich bei

ArTEMiS mit seinem Passwort und Usernamen anmelden, die von der Universität bei der Immatrikulation erstellt wurden [6].

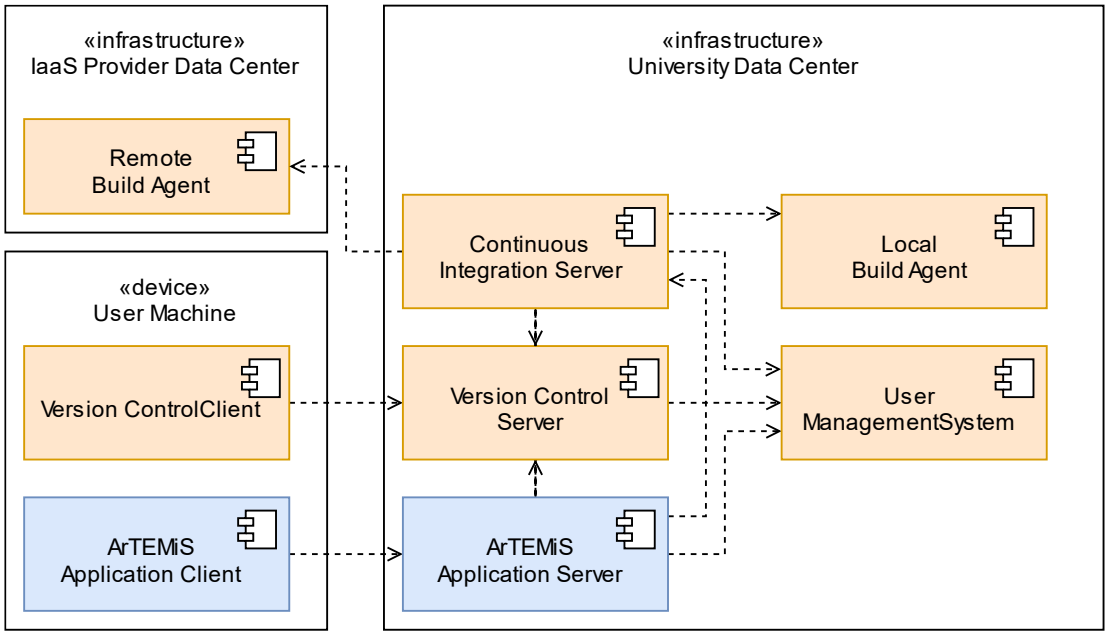


Abbildung 8: ArTEMiS Top Level Architektur [6]

Abbildung 9 stellt den Application Client dar. Bei dem Client handelt es sich um einen sogenannten Thin-Client. Das bedeutet, dass ein Ganzes oder ein Teil der Benutzerschnittstelle auf dem Client ausgeführt wird, wobei die Datenhaltung und die Anwendung auf dem Server platziert sind [10].

Die Hauptlogik befindet sich dementsprechend auf dem Server und der Client dient lediglich als Benutzeroberfläche zur Interaktion mit dem System. Die Abbildung zeigt außerdem, dass individuelle Sichten der Benutzeroberfläche existieren. Dem Lehrenden werden beispielsweise eine andere Sicht bzw. andere Funktionen als dem Studenten geboten. In Abhängigkeit dazu muss eine Authentifizierung und Autorisierung geregelt werden, die mit dem Connector verbunden sind. Dieser nutzt die API und verschiedene Dienste von ArTEMiS und kommuniziert mit dem Server [5].

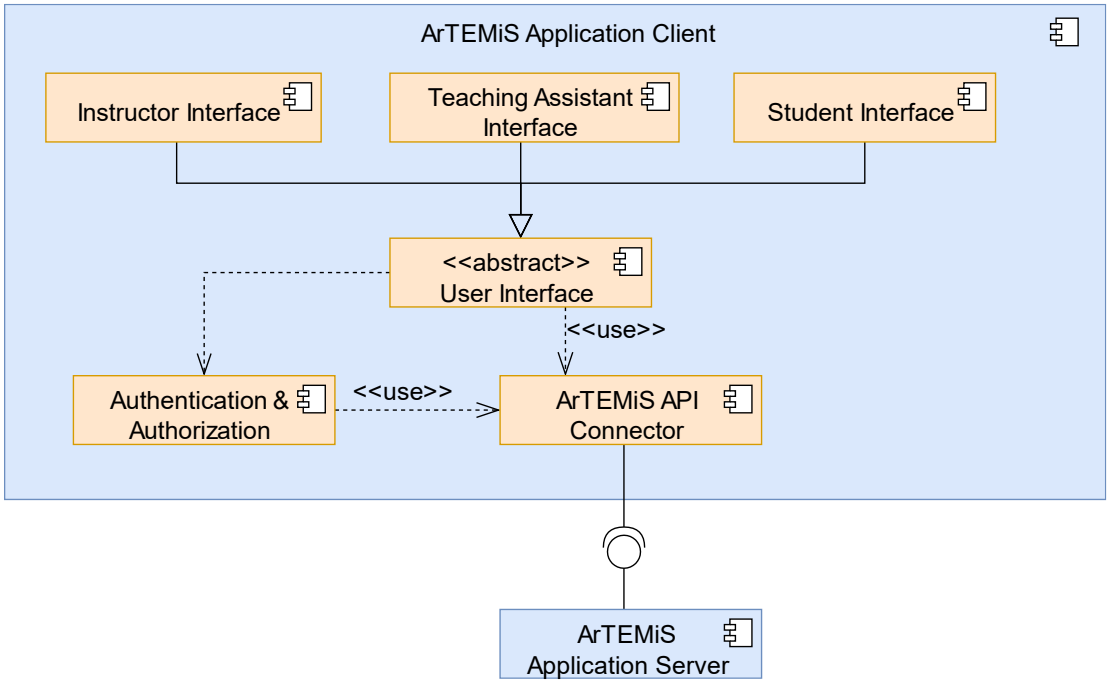


Abbildung 9: ArTEMiS Apllication Client [5]. Application Client und Server sind blau hervorgehoben.

Die verschiedenen Dienste, die vom Server bereitgestellt werden, sind in Abbildung 10 wiederzufinden. Der Server verwendet das Konzept der Drei-Schichten-Architektur (engl. three tier architecture). Dieser Entwurf ist sehr flexibel, weil einzelne Schichten leicht umgebaut bzw. weiter ausgebaut werden können. Ganz oben in dem Komponentendiagramm kann der Application Client wiedergefunden werden und ist mit den Komponenten der Präsentationsschicht bzw. Web Layers verbunden. Diese Schicht ist für die Darstellung der Benutzeroberfläche und die Eingabe und Ausgabe von Daten verantwortlich. Wenn der Student beispielsweise eine Aufgabe starten möchte, wird dies über die Exercise Participation Resource entgegengenommen, die mit einer Komponente aus dem Application Layer bzw. der Logikschicht verbunden ist. Hier befinden sich verschiedene Services. Zusätzlich befinden sich dort Adaptoren, um externe Services in Anspruch zu nehmen. Der Exercise Participation Service beinhaltet die Funktion, eine Aufgabe für einen Studenten zu starten und kommuniziert daraufhin mit dem Adapter, welches die Verwaltung der Repositories regelt, um für den Studenten die bereits besprochenen Repositories zu erstellen. Das Speichern und Laden der Daten erfolgt aus

der untersten Schicht, der Datenhaltungsschicht (engl. Data Layer). Jede Komponente aus der Logikschicht kommuniziert mit einer Komponente aus der Datenhaltungsschicht, weshalb diese beiden Schichten in der Abbildung mit einem Pfeil verbunden sind.

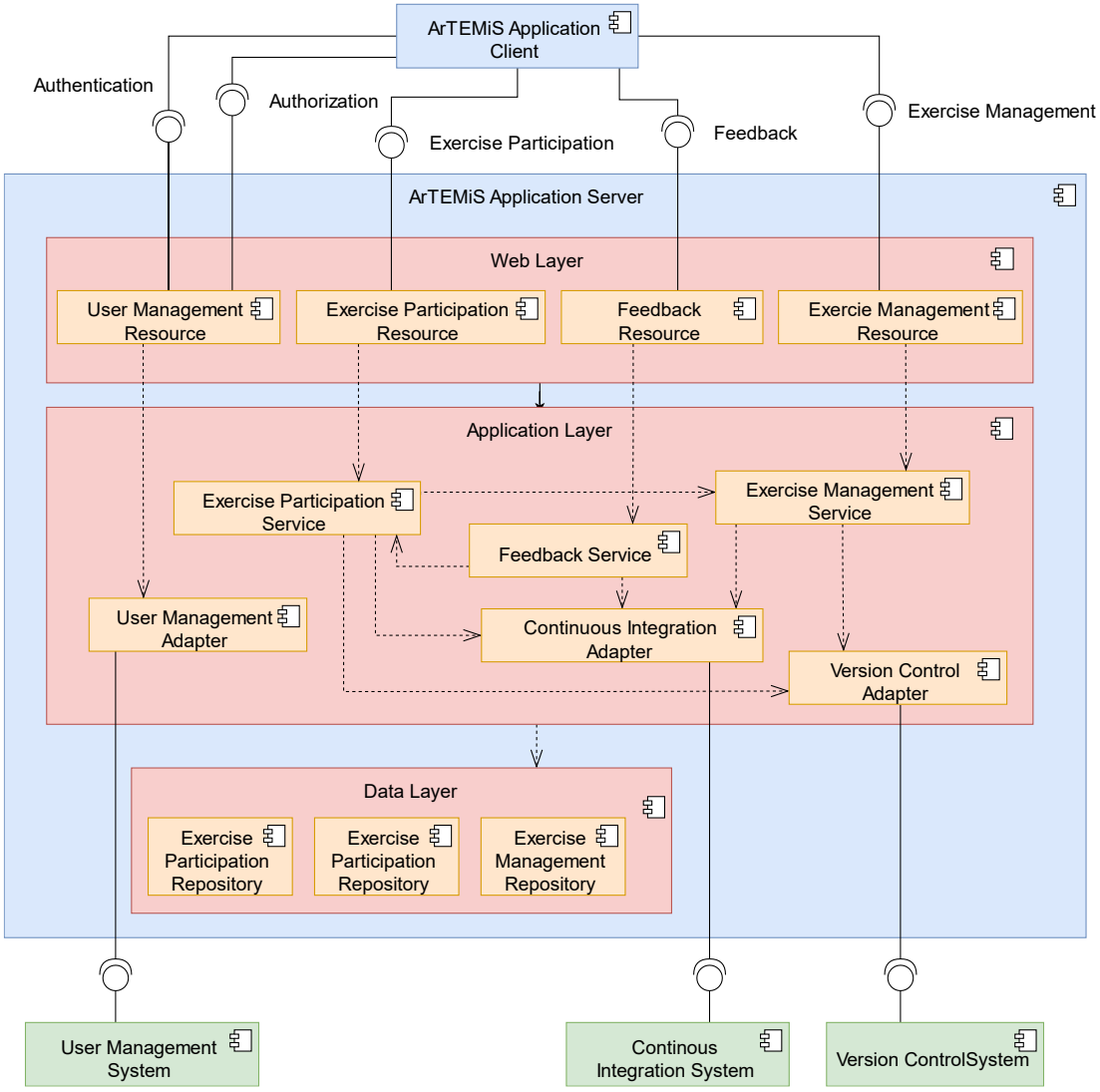


Abbildung 10: ArTEMiS Application Server [5]. Application Client und Server sind blau hervorgehoben. Die drei Schichten Architektur ist in rot dargestellt und externe Komponenten in grün.

4 JACK

Nach einer kurzen Einleitung wird der Funktionsumfang von JACK beschrieben. Da es hier eine Reihe von Feedbackmöglichkeiten gibt, wird es dafür ein zusätzliches Unterkapitel geben. Anschließend wird auch hier der technische Hintergrund erläutert.

4.1 Einleitung

JACK wurde 2006 an der Universität Duisburg am Paluno Institut[5] entwickelt. Es existieren mehrere Versionen. Zurzeit ist JACK3[6] in Entwicklung und soll den Vorgänger JACK2[7] ersetzen. Letzteres wird nicht weiterentwickelt und ist bis zur Fertigstellung von JACK3 im Einsatz. Wenn in der Arbeit von JACK gesprochen wird, ist immer JACK2 gemeint und auch nur diese Version wird betrachtet.

Bei dem Namen JACK handelt es sich um ein Akronym, welches aus „**Ja**va Che**ck**er“ zusammengesetzt ist [11]. In der Plattform können Programmier-, Modellierungs-, Multiple-Choice-, Lückentext- und Code-Reading-Aufgaben bearbeitet werden. Die verschiedenen Checker (der Java Checker ist einer davon) werden dazu genutzt, um die Lösung des Studenten zu testen und Feedback zu liefern. Bei den Code-Reading Aufgaben muss der Student sein Verständnis des Quelltextes unter Beweis stellen. Dabei füllt er eine Trace-Tabelle aus. Diese beschreibt in welchen Zeilen die Variablen welche Belegung haben. Um die Lösung dieser Tabelle zu testen, wurde ein spezieller Code-Reading Checker entwickelt. Aus diesem Grund unterscheidet sich der Aufgabentyp von herkömmlichen Multiple-Choice Aufgaben, in denen das Verständnis des Codes abgefragt werden kann. Diese Aufgabentypen werden jedoch nicht weiter in dieser Arbeit behandelt, denn der Fokus liegt weiterhin in den Programmieraufgaben.

[5] https://paluno.uni-due.de/

[6] https://jack-community.org/wiki/index.php/JACK3

[7] https://jack-community.org/wiki/index.php/Hauptseite

4.2 Analyse der Funktionalitäten

Ähnlich wie im letzten Kapitel werden für die Analyse Use Cases genutzt, um die Funktionalitäten übersichtlich darzustellen, die für die einzelnen Stakeholder bzw. Aktoren zur Verfügung stehen. Die Erkenntnisse sind entstanden, indem in der Plattform Aufgaben erstellt und bearbeitet wurden. Ähnliche Beschreibungen der Funktionalitäten sind außerdem im JACK-Wiki[8] zu finden. Auch JACK unterscheidet nicht zwischen Lehrenden und Aufgabenentwicklern. Dennoch werden auch hier die Stakeholder getrennt voneinander betrachtet. Die Vorgehensweise ist dieselbe wie im letzten Kapitel. Da JACK nicht Open-Source ist bzw. nicht für jeden frei zugänglich ist, befinden sich im Anhang Aufnahmen von der Benutzeroberfläche, worauf im Fließtext verwiesen wird.

Aufgabenentwickler

Abbildung 11 zeigt die Erstellung einer Aufgabe. Zunächst muss ein Aufgabentyp gewählt werden, die Java, UML, Formbased, AES, Code_Reading oder R sein kann. In dieser Arbeit wird nur noch auf die programmierspezifischen Aufgaben eingegangen, also Java und R. In JACK sagt der Aufgabentyp nicht unbedingt immer etwas darüber aus, um welche Programmiersprache es sich bei der Aufgabe handelt. Mit dem Aufgabentyp Java können ebenso C++-Aufgaben entwickelt werden. Damit Klarheit besteht, werden im weiteren Verlauf der Arbeit keine C++-Aufgaben gemeint sein, wenn die Rede vom Aufgabentyp Java ist.

Für die Aufgabe kann außerdem noch eine Kategorie festgelegt werden, damit diese später leichter zu finden ist. Die Aufgaben können für Moodle freigeschaltet werden, damit der Student diese dort bearbeiten kann. Diese externe Plattform wird in dieser Arbeit nicht weiter thematisiert. Außerdem kann die Aufgabe zusätzlich für Kurse freigeschaltet werden, damit der Lehrende diese in seine Kurse integrieren kann. Für die Aufgabe kann eine interne Formulierung beschrieben werden, die für Studenten nicht öffentlich ist. Die externe Beschreibung ist für Studenten sichtbar und gibt bei Java- und C++-Aufgaben kurze Auskunft über die Aufgabenstellung. Bei der Erstellung von R-Aufgaben kann auch eine externe Beschreibung formuliert werden, die dem Studenten jedoch nicht angezeigt wird, weil die

[8] https://jack-community.org/wiki/index.php/Hauptseite

Aufgabenbeschreibung in einer XML-Datei formuliert werden muss. Hierauf wird im späteren Verlauf dieses Kapitels eingegangen.

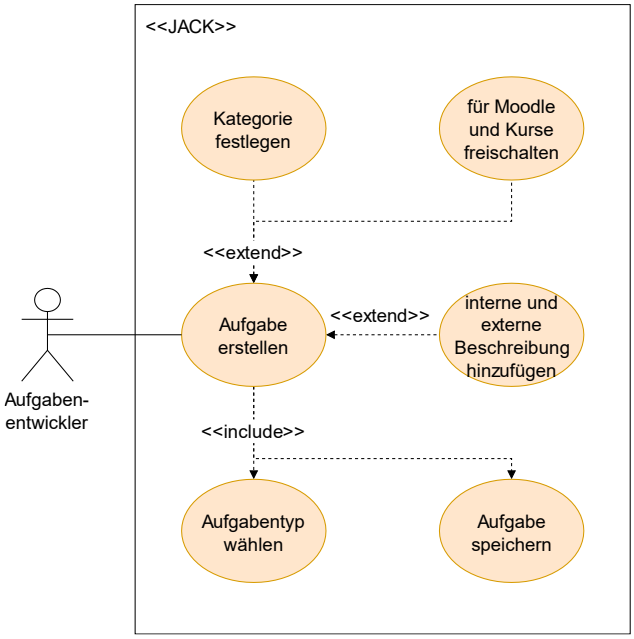


Abbildung 11: Use-Cases-Diagramm: Aufgabenentwickler in JACK

Sobald die Aufgabe gespeichert wird, muss die Aufgabenstellung überarbeitet werden (s. B.1, B.3 u. Abbildung 12). Dies kann von mehreren Entwicklern durchgeführt werden, denn es besteht die Möglichkeit, eine Aufgabe für mehrere Entwickler zu autorisieren bzw. freizuschalten. In der Abbildung ist das Use-Case als „Lehrenden und Aufgabenentwickler autorisieren“ beschrieben, weil die Lehrenden auch Berechtigungen für die Aufgabe haben müssen, um beispielsweise die Ergebnisse und Lösungen der Studenten anzusehen. Dies wird im Abschnitt „Lehrender“ genauer erläutert.

Die Konfigurationen (s. Abbildung 11), die im Vorfeld getroffen werden, können jederzeit wieder geändert werden. Zusätzlich zur Kategorie, die bei der Voreinstellung definiert wird, können Tags hinzugefügt werden. Diese werden nur bei der Erstellung von Kursen verwendet, um die Aufgabe über diese Tags dem Kurs hinzuzufügen (s. Abbildung 16).

Aufgaben können eine Schwierigkeitsstufe bekommen, die laut JACK-Wiki[9] zwischen eins und fünf liegen muss. Jedoch ist mir aufgefallen, dass auch höhere Werte vom System akzeptiert werden (s. B.4). Zusätzlich kann auch eine Schwierigkeitsstufe für einen adaptiven Kurs festgelegt werden. Diese Kurse sollen Aufgaben vorschlagen, die für den Studenten vom Schwierigkeitsgrad her als geeignet gesehen werden. Dies soll verhindern, dass der Student zu leichte oder zu schwierige Aufgaben bekommt. Wie genau dies funktioniert ist im Wiki nicht beschrieben, weil die Entwicklung noch nicht abgeschlossen ist. Daher werden diese Kurse in dieser Arbeit nicht weiter betrachtet.

Die Aufgaben sind bestanden, sobald die minimale Punktzahl erreicht wird, die ebenfalls definiert werden muss. Es ist zusätzlich möglich, Punktabzüge festzulegen. Die Punktabzüge werden in Prozent definiert und kommen erst zum Einsatz, wenn der Student drei Mal eine falsche Lösung eingereicht hat. Für jede weitere Einreichung wird der angegebene Prozentsatz von dem Gesamtergebnis abgezogen. Wenn der Punktabzug also 5% beträgt, wird bei der vierten Einreichung 5%, bei der Fünften 10% und bei der Sechsten 15% usw. abgezogen.

Das Herz der Aufgabe sind die Ressourcen, die in JACK hochgeladen werden müssen. Diese werden in verschiedene Kategorien eingeteilt, in sogenannten „Sheets“. Als Instruction Sheet werden Dateien hochgeladen, in der die Aufgabenstellung beschrieben ist. Als Working Sheet wird die zu bearbeitende Datei hochgeladen, wie zum Beispiel ein Template, in dem Klassen mit leeren Methodenrümpfen erweitert werden müssen. Als Reference Sheet werden die Dateien hochgeladen, die vom Working Sheet referenziert werden, also zum Beispiel fertig implementierte Klassen. Ferner gibt es noch die Hidden Sheet Dateien, die dem Studenten verborgen bleiben und beispielsweise in die „Checker“ als Testklassen eingebunden werden.

Die automatischen Tests in JACK sind über verschiedene „Checker“ realisiert, die unterschiedliche Feedbacks liefern [11]. Alle Checker können unterschiedlich gewichtet werden und tragen somit unterschiedlich zur erreichten Gesamtpunktzahl der Aufgabe bei. Für jeden Checker gibt es die Standardfunktion, ob die Punktzahl, die von diesem ausgeht, dem Studenten angezeigt werden soll oder nicht. Außerdem können die Feedbacks, die aus dem Checker bzw. aus

[9] https://jack-community.org/wiki/index.php/Aufgaben

den Testklassen kommen, abgeschaltet werden. Mehr zu den einzelnen Checkern gibt es im nächsten Kapitel 4.3.

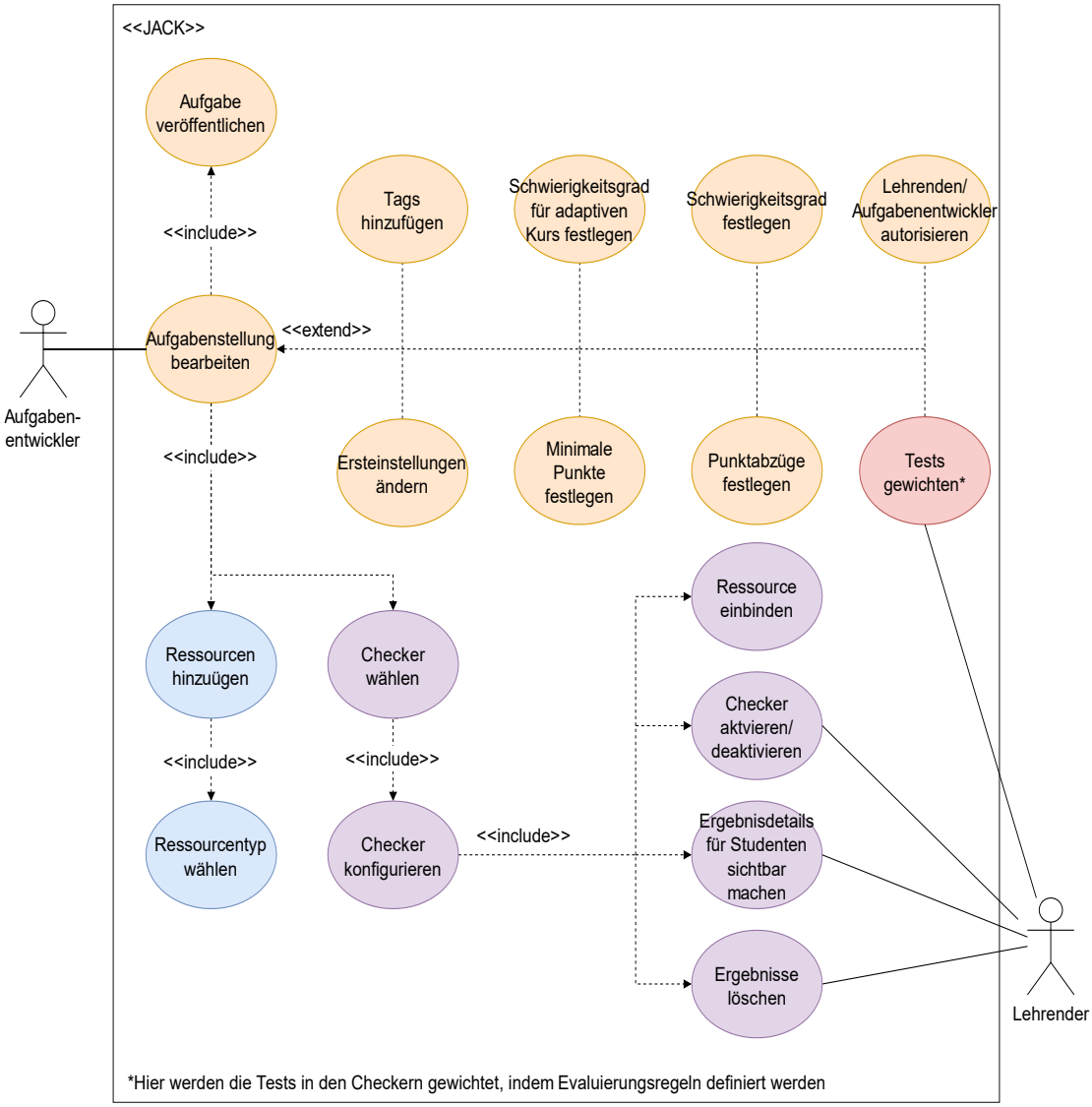


Abbildung 12: Use-Cases-Diagramm: Aufgabenentwicklung in JACK.

Die gezeigten Einstellungsmöglichkeiten in B.2 - B.3 sind bei allen Aufgabentypen dieselben. Bei jedem Aufgabentypen werden Konfigurationsmöglichkeiten vorgeschlagen, die nicht unbedingt notwendig sind. Beispielsweise die externe Beschreibung bei R-Aufgaben oder die

Auswahl von Checkern, die einen anderen Aufgabentyp betreffen (s. B.6). Es lässt sich außerdem vor der Veröffentlichung nicht testen, ob die Erstellung einer Aufgabe erfolgreich war. Wie bisher beschrieben wird in JACK eine Aufgabe erstellt und danach überarbeitet. Ohne die Überarbeitung würde die Aufgabenstellung keinen Sinn ergeben, weil beispielsweise die Ressourcen fehlen, die Checker nicht konfiguriert sind usw. Dennoch lässt sich eine Aufgabe nach der Erstellung direkt veröffentlichen.

Die verschiedenen Aufgabentypen müssen bestimmte Vorrausetzungen erfüllen, damit sie als Aufgabenstellung Sinn ergeben und vom Studenten bearbeitet werden können. Für Java und C++-Aufgaben ist als Mindestvoraussetzung ein Instruction Sheet für die Aufgabenbeschreibung, sowie ein Hidden Sheet, womit die Checker konfiguriert werden können, hochzuladen. Diese Aufgaben werden lokal bearbeitet und die Lösung wird vom Studenten in die Plattform hochgeladen, wobei die Abgabedatei genauso heißen muss, wie die Working Sheet Datei.

Im Gegensatz dazu können R-Aufgaben direkt auf der Plattform in einem Editor bearbeitet werden. Das Grundgerüst dieses Aufgabentyps wird gebündelt in einer XML-Datei[10] hochgeladen. Sowohl Aufgabenbeschreibung als auch Hinweise sowie Musterlösungen zur Aufgabe können in dieser Datei formuliert werden. Auch globale Variablen können definiert werden, die vom Studenten bei der Lösungsentwicklung genutzt werden können. Zudem kann in der XML-Datei auf eine andere Datei referenziert werden, in dem ein initialer R-Code beschrieben ist. Diese Datei wird als Working Sheet hochgeladen und der darin enthaltene Code wird dem Studenten zur Verfügung gestellt [12]. In der XML-Datei kann außerdem bestimmt werden, ob während der Aufgabenbearbeitung dem Studenten ein Menü angezeigt werden soll, in dem R-Befehle zusammengefasst sind (s. B.7.3). Dieses ist besonders für Programmieranfänger hilfreich.

Die Aufgaben können nur von einem Studenten gestartet und bearbeitet werden. Der Ersteller muss somit ein Studentenaccount besitzen, um die Aufgabe starten zu können. Er muss sich also aus seinem Entwickleraccount ausloggen und sich mit seinem Studentenaccount

[10] https://jack-community.org/wiki/index.php/Exercise(Datei_f%C3%BCr_R-Aufgaben)

einloggen, um zu prüfen, ob die Aufgabe bearbeitet werden kann bzw. ob die Erstellung erfolgreich war.

Student

Sobald eine Aufgabe bzw. ein Kurs erfolgreich erstellt wurde, kann der Student diese unter der definierten Kategorie finden und auswählen. Mit Ausnahme von R-Aufgaben wird eine Übersicht (s. B.7.2) angezeigt, sobald eine Aufgabe gestartet wird, auf der Ressourcen heruntergeladen werden und Lösungen hochgeladen bzw. eingereicht werden können. Außerdem wird auf dieser Übersicht die externe Aufgabenbeschreibung angezeigt. R-Aufgaben können nur direkt auf der Plattform in einem Online-Editor (s. B.7.3) bearbeitet werden, der erscheint, sobald die Aufgabe gestartet wird. Der Editor ist ein einfaches Textfeld und hat keinerlei Funktionen, wie beispielsweise Autovervollständigung. Der geschriebene Code kann direkt auf der Plattform ausgeführt werden. Und im Terminal, direkt unter dem Editor, wird das Ergebnis angezeigt (inkl. Errors, Warnings, Syntaxfehler etc.) [12]. Zusätzlich können in dem Editor Hinweise eingeblendet werden, wenn der Student bei einer Aufgabe nicht vorankommt. Sollten Fehler in der Aufgabenstellung enthalten sein, so kann der Student dies melden. Für R-Aufgaben können keine Ressourcen zum Download bereitgestellt werden. Für die anderen Programmieraufgaben existiert kein Online-Editor, sondern es kann nur lokal gearbeitet werden und die Lösung wird hochgeladen.

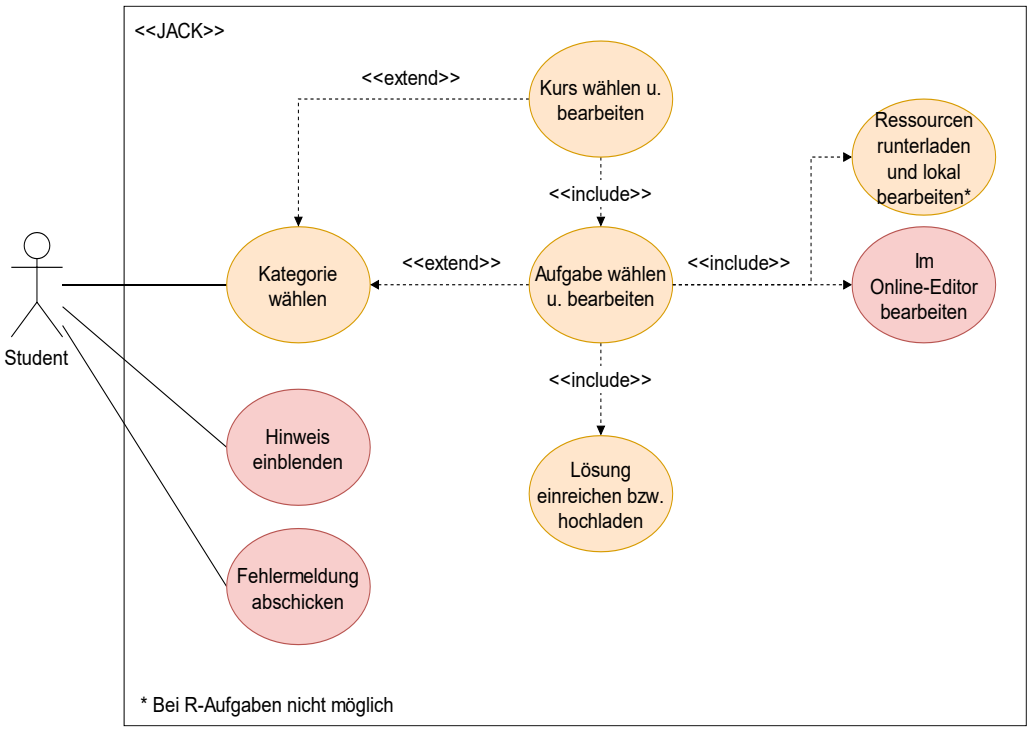


Abbildung 13: Use-Cases-Diagramm: Student in JACK. Rote Use-Cases gelten nur für den Aufgabentyp R.

Im Folgenden ist es wichtig, eine Unterscheidung zwischen den Wörtern Einreichung, Ergebnis und Lösung einer Aufgabe zu treffen. Eine Lösung ist beispielsweise ein Quellcode, um die Anforderungen einer Aufgabe zu erfüllen und um ein korrektes Ergebnis zu bekommen. Ein Ergebnis ist demnach das Resultat einer Lösung. Eine Einreichung beinhaltet diese Lösung. Zusätzlich gibt eine Einreichung weitere Informationen an, wie zum Beispiel das Einreichungsdatum oder den Absender der Einreichung. Auf der Benutzeroberfläche von JACK sind diese Begriffe nicht konsequent voneinander getrennt worden. Beispielsweise wird bei dem Lösungsüberblick eine Einreichungsübersicht angezeigt und nicht die eigentliche Lösung (s. B.8). Da in den Use-Cases die Begrifflichkeiten voneinander getrennt wurden, aber in JACK nicht, könnte es beim Vergleich der Screenshots im Anhang und den Use-Cases zur Verwirrung führen.

Sobald eine Lösung hochgeladen bzw. eingereicht wurde, werden die Checker automatisch gestartet. Die Einreichungen können über eine Einreichungsübersicht (s. B.8) in JACK

abgerufen werden. Dort werden Aufgaben- und Kurseinreichungen aufgelistet. Diese Liste kann außerdem als Excel-Datei heruntergeladen werden. In dieser Datei stehen zusätzliche statistische Informationen, wie z.B. die größte erreichte Punktzahl oder die Anzahl der Versuche bzw. Lösungseinreichungen einer Aufgabe. Sobald eine Kurseinreichung ausgewählt wird, werden die Aufgaben, die in diesem Kurs existieren, dargestellt, und es wird eine Ergebnisstatistik über den Kurs angezeigt (s. B.9.1). Wie auch in der Einreichungsübersicht, können nun die Ergebnisse der Lösung der Aufgaben abgerufen werden. Bei dem Ergebnisüberblick (s. B.9.2) werden Checker Ergebnisse angezeigt und die daraus resultierenden Feedbacks sowie Punkte (sofern vom Entwickler freigeschaltet). Außerdem besteht die Möglichkeit, für den Lehrenden ein manuelles Feedback für Aufgabenlösungen zu hinterlassen, die ebenfalls in der Ergebnisübersicht der Lösung der Aufgabe auftauchen. Die Lösung bzw. der Quellcode kann in der Übersicht ebenfalls evaluiert werden, wobei Zeilen gelb hervorgehoben werden, wenn diese durch die Checker bzw. Testfälle nicht erreicht werden. Auch der Lehrende kann sich diese Übersicht anschauen, wie im Folgenden noch beschrieben wird.

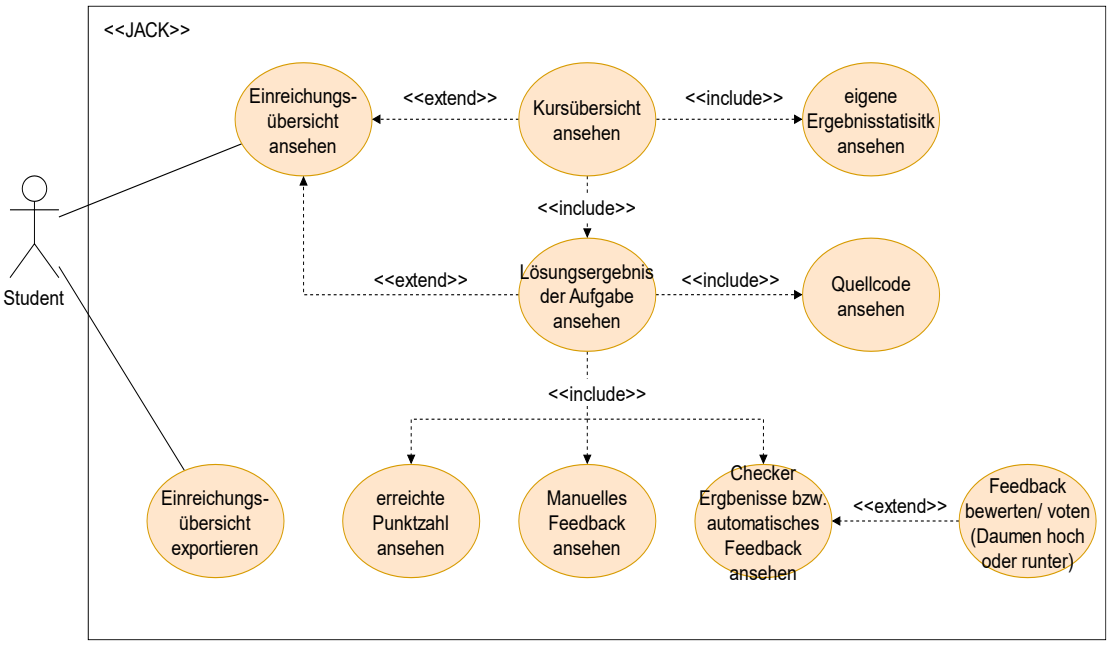


Abbildung 14: Use-Cases-Diagramm: Student in JACK.

Lehrender

Im Folgenden werden die Funktionen analysiert, die für den Lehrenden bereitgestellt werden. Abbildung 15 zeigt die Möglichkeiten im Hauptmenü (s. B.1).

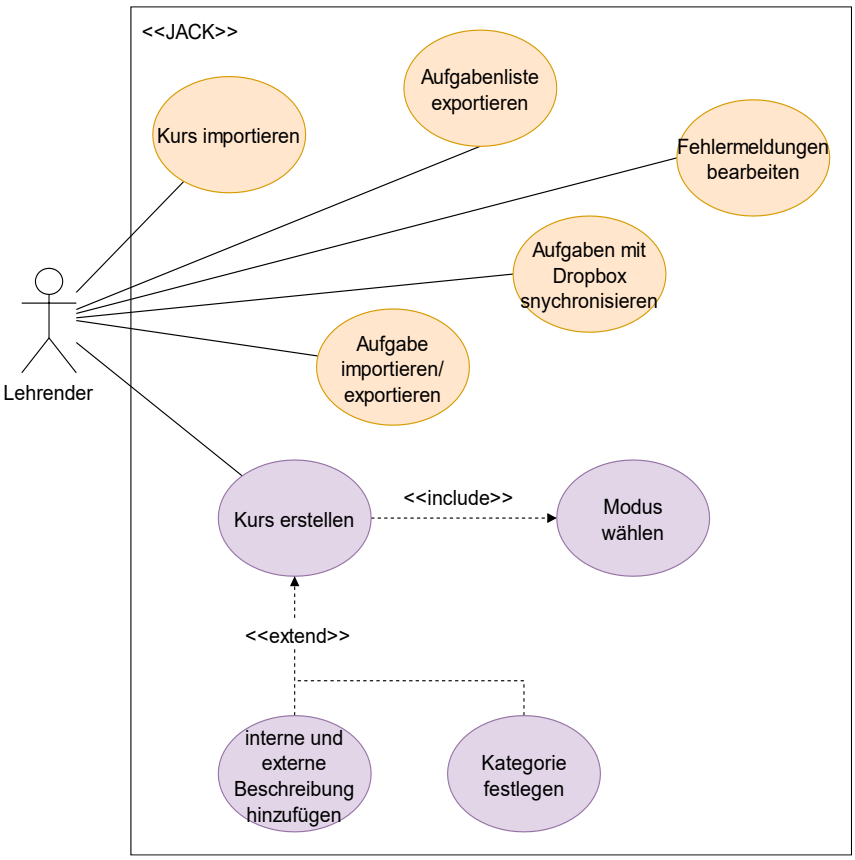


Abbildung 15: Use-Cases-Diagramm: Lehrender in JACK.

Kurse und Aufgaben können in der Plattform exportiert und importiert werden. Es ist außerdem möglich, eine Dropbox anzubinden („Meine Aufgaben“ im Hauptmenü) und die Aufgaben dort zu speichern oder von dort aus Aufgaben in JACK zu importieren. Darüber hinaus können die Aufgaben als Übersichtstabelle heruntergeladen werden.

Der Lehrende kann sich außerdem die Fehlermeldungen der Studenten ansehen und diese bearbeiten.

Bei der Erstellung eines Kurses muss, so wie bei der Aufgabenerstellung, eine Vorkonfiguration durchgeführt werden (s. B.4). Zunächst kann auch hier eine interne und externe Beschreibung sowie eine Kategorie definiert werden, die beliebig sein können. Außerdem muss zwischen sechs Modis gewählt werden, die besondere Regeln für den Kurs festlegen. Es gibt zwei klausurspezifische Regeln, auf die nicht weiter eingegangen wird. Der Rest der Regeln besagt, ob Feedbacks und Hinweise bei der Einreichung herausgegeben werden sollen und wie oft eine Aufgabe eingereicht werden darf. Die Tabelle 1 zeigt eine Übersicht der Regeln.

Tabelle 1 JACK: Kursregeln

Regel bzw. Modus	***Feedback einblenden***	***Hinweise einblenden***	***Einreichung pro Aufgabe***
1	Ja	Ja	Beliebig oft
2	Ja	Nein	Beliebig oft
3	Nein	Nein	Beliebig oft
4	Nein	Nein	Nur einmal

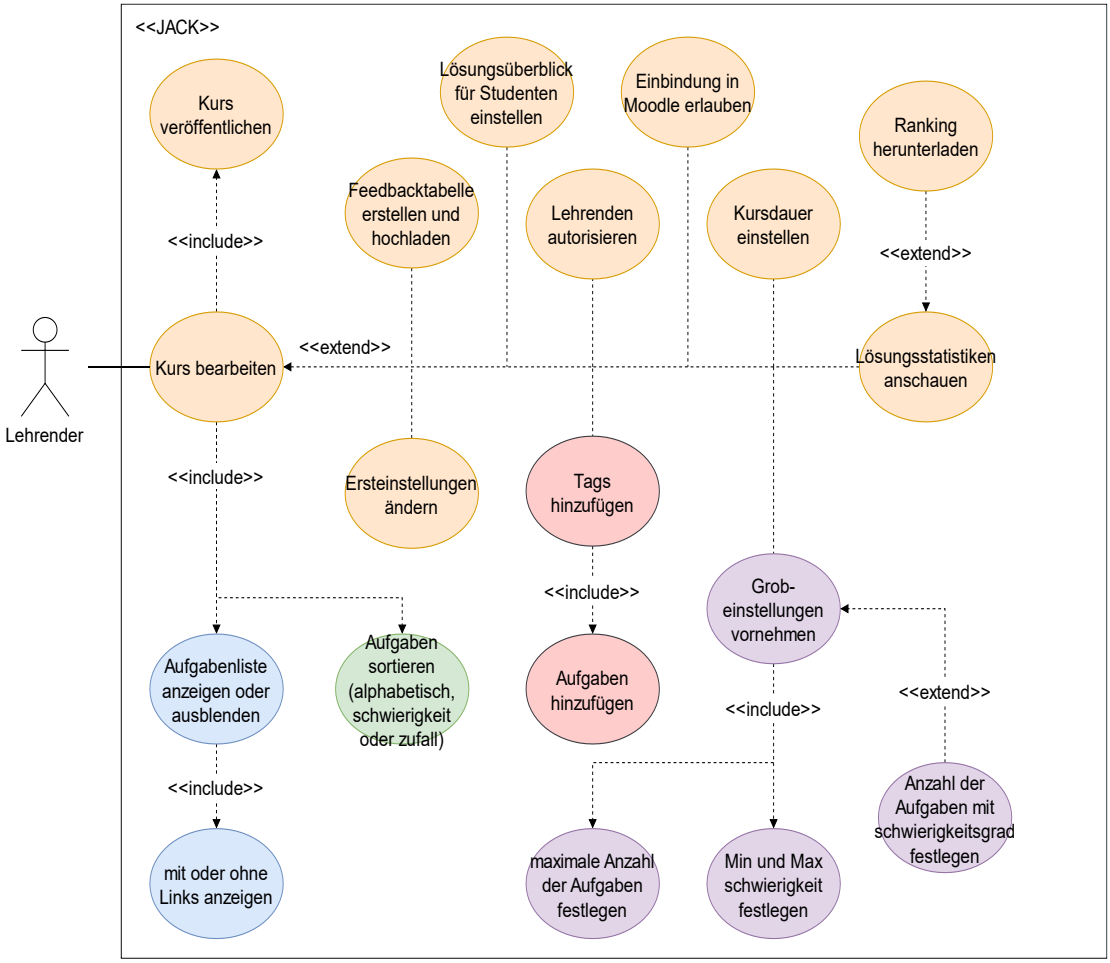


Abbildung 16: Use-Cases-Diagramm: Lehrender in JACK

Nach der Erstellung eines Kurses muss dieser überarbeitet werden (s. Abbildung 16, B.5). Auch hier können die Einstellungen bzw. Vorkonfigurationen aus Abbildung 15 wieder verändert werden, wodurch diese Ersteinstellungen gewissermaßen überflüssig werden. Jedem Kurs können Aufgaben mithilfe der Tags hinzugefügt werden. Sobald ein Tag ausgewählt wird, werden alle Aufgaben, die unter diesem Tag erstellt wurden, automatisch zu dem Kurs hinzugefügt. Das gleiche gilt für die Aufgaben, für die der Lehrende nicht autorisiert worden ist. Dabei hat der Lehrende nicht die Möglichkeit wahlweise einzelne Aufgaben auszuschließen. Nur mit der Grobeinstellung können Aufgaben aus der „Tag Gruppierung“ bzw. dem Aufgabenpool entfernt werden. In der einfachen Grobeinstellung kann vorgegeben werden, wie viele Aufgaben

dem Kurs hinzugefügt werden dürfen. Dabei kann festgelegt werden, welche minimale und maximale Schwierigkeitsstufe die Aufgaben haben dürfen. Wenn beispielsweise unter einem Tag 10 Aufgaben existieren und die maximale Anzahl der Aufgaben in einem Kurs auf zwei festgelegt wird, wobei der minimale und maximale Schwierigkeitsgrad zwischen 1-3 festgelegt wird, werden zwei zufällige Aufgaben mit diesem gültigen Schwierigkeitsgrad aus den 10 Aufgaben ausgewählt werden. In der erweiterten Grobeinstellung (s. B.5) kann außerdem noch explizit angegeben werden, wie viele Aufgaben mit welchem Schwierigkeitsgrad zufällig hinzugefügt werden dürfen.

In JACK ist ein Kurs somit eine zufällige Ansammlung von Aufgaben, wobei nur Programmiersprachen vom Typ R hinzugefügt werden können. Aufgaben vom Typ C++ oder Java können keinem Kurs zugewiesen werden. Ansonsten können noch andere Aufgabentypen (Multiple Choice etc.) einem Kurs hinzugefügt werden, auf die in dieser Arbeit kein Bezug genommen wird.

Dem Studenten können die Aufgaben aus dem Kurs als Liste angezeigt werden, während dieser eine Aufgabe aus dem Kurs bearbeitet. Zusätzlich kann entschieden werden, ob die Aufgaben aus der Liste linkbehaftet bzw. aufrufbar sind. Somit kann der Student leichter zwischen Aufgaben wechseln. Dem Studenten wird außerdem eine Liste der Aufgaben angezeigt, sobald dieser einen Kurs öffnet bzw. startet. Die Liste kann alphabetisch, nach Schwierigkeitsgrad oder nach Zufall sortiert werden. Eine Bearbeitungsreihenfolge für die Aufgaben lässt sich jedoch nicht festlegen, dafür aber eine Bearbeitungszeit für den gesamten Kurs, welche alle Aufgaben beinhaltet. Die Zeit beginnt, sobald der Kurs gestartet wird. Wenn die Zeit vorüber ist, werden alle Aufgaben gesperrt und der Student kann nicht weiter an seiner Lösung arbeiten.

Wie bereits beschrieben kann der Student eine Kursüberblicksseite aufrufen, in der die Statistik des Kurses angezeigt wird (s. B.9.1). Diesen Kursüberblick kann der Lehrende wahlweise abschalten. Auch der Aufruf des Ergebnisüberblickes einzelner Aufgaben im Kursüberblick kann abgeschaltet werden. Somit würde der Student kein Feedback für die Aufgaben in seinem Kurs erhalten.

Wie bei der Aufgabenerstellung kann auch hier der Kurs für die Bearbeitung auf der externen Plattform Moodle freigeschaltet werden.

Nachdem alle Konfigurationen für die Aufgabe abgeschlossen sind, kann die Aufgabe veröffentlicht werden. In der Kursbearbeitung kann sich der Lehrende außerdem eine Lösungsstatistik anschauen (s. B.8.1) oder sich eine Rankingtabelle herunterladen. Er kann sich jedoch nicht die einzelnen Einreichungen von Aufgabenlösungen aus dem Kurs anschauen. Dies können nur Lehrende, die für die Aufgabe autorisiert wurden (beschrieben unter Abbildung 12). Dem Kurs werden jedoch, wie bereits beschrieben, zufällige Aufgaben hinzugefügt, wofür der Lehrende nicht unbedingt autorisiert ist. Auch Kurse können für mehrere Lehrende autorisiert werden. Mit der Autorisierung bekommt ein Lehrender alle Rechte für diesen Kurs, die bis hierhin beschrieben wurden.

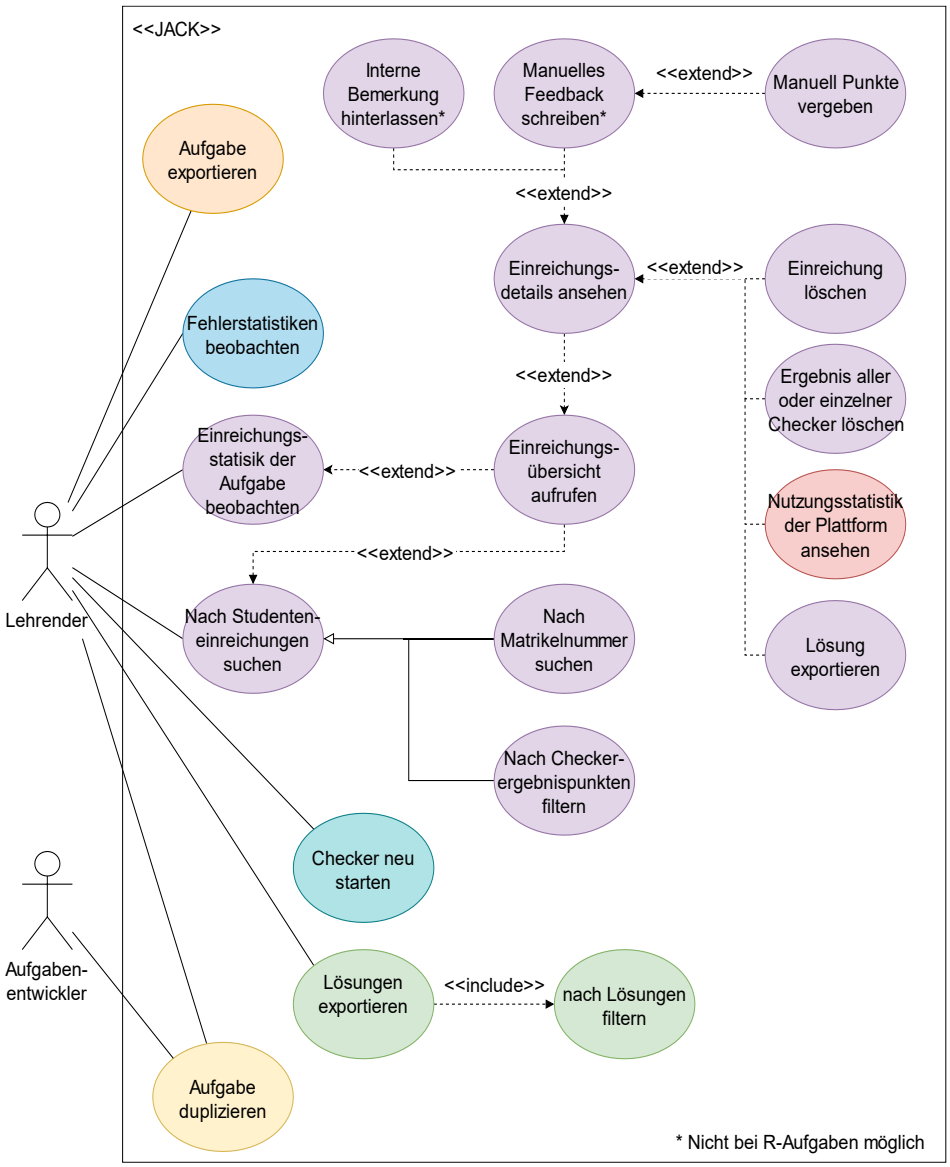


Abbildung 17: Use-Cases-Diagramm: Lehrender in JACK.

Einige Funktionen, die den Lehrenden betreffen, befinden sich in dem Überarbeitungsmenü der Aufgabe (s. B.3), d.h. er braucht eine Berechtigung für die Aufgabe (erwähnt im Abschnitt des Aufgabenentwicklers in Abbildung 12). Diese Funktionen sind in Abbildung 17 dargestellt.

Die Aufgabe kann exportiert bzw. heruntergeladen werden oder dupliziert werden. Die Duplikation betrifft sowohl den Lehrenden als auch den Aufgabenentwickler. Möglicherweise will

der Lehrende die gleiche Aufgabe in eine andere Kategorie einordnen oder der Aufgabe andere Tags zuweisen, um diese in andere Kurse integrieren zu können. Für den Aufgabenentwickler ist diese Option auch hilfreich, weil beispielsweise die Checker-Einstellungen sehr aufwendig zu konfigurieren sind und somit weniger Arbeitszeit notwendig wäre, um eine ähnliche Aufgabe zu entwickeln.

Eine Unterscheidung zwischen den Wörtern Einreichung, Ergebnis und Lösung einer Aufgabe wurde bereits getroffen. Da auch in den lilafarbigen Use-Cases die Begrifflichkeiten voneinander getrennt wurden, aber in JACK nicht, könnte es beim Vergleich der Screenshots im Anhang und den Use-Cases zur Verwirrung führen.

Die lila Use-Cases beschreiben das Beobachten und Verwalten einer Aufgabe. In JACK gibt es mehrere Wege, um verschiedene Menüs oder Funktionen aufzurufen, weshalb diese Use-Cases eng miteinander gekoppelt sind. Der Lehrende kann sich eine Einreichungsstatistik ansehen (s. B.8.2), in der verschiedene Einreichungsinformationen aufgelistet sind, wie zum Beispiel die Anzahl erfolgreicher oder fehlgeschlagener Einreichungen. Die Zahlen in der Statistik sind linkbehaftet und führen zu einer Einreichungsübersicht. Dort werden die Einreichungen in einer Liste angezeigt, die nach verschiedenen Kriterien sortiert werden kann. Diese Übersicht kann außerdem über eine Suchfunktion aufgerufen werden, in der die Matrikelnummer des Studenten gesucht wird und schließlich alle Einreichungen von dem Studenten angezeigt werden. Eine zweite Suchfunktion bietet die Möglichkeit, nach bestimmten Checker-Ergebnissen zu suchen (s. B.8.3). Dadurch können beispielsweise alle Einreichungen aller Studenten angezeigt werden, in der ein bestimmter Checker eine bestimmte Punktzahl geliefert hat. Jede Einreichung kann außerdem im Detail angeschaut werden (s. B.8.4 u. B.8.5). In der Detailübersicht aller Aufgabentypen (Java, C++, R) kann der Lehrende die Einreichung oder einzelne Checker-Ergebnisse löschen. Letzteres kann auch, anstatt es zu löschen, vor dem Studenten verborgen werden.

In der Detailübersicht gibt es noch die Besonderheit, eine andere Übersicht (s. B.8.7) aufzurufen. Das Use-Case ist in der Abbildung rot markiert, weil es keinen Zusammenhang zu der aufgerufenen Ergebnisübersicht darstellt.

Die Statistiken in der Übersicht beziehen sich auf alle existierenden Aufgaben in der Plattform. Dort wird zum einen tabellarisch die Nutzungshäufigkeit der Checker gezeigt und zum anderen

die Einreichungshäufigkeit aller Studenten in einem Balkendiagramm. Der Lehrende kann die Zeitachse des Diagrammes anpassen und die Balken anklicken. Der Link führt zu einer Einreichungsübersicht, ähnlich wie in B.8.3. Hier werden jedoch Einreichungen aller Aufgaben aufgelistet. Auch die, auf die der Lehrende keinen Zugriff hat. Damit der Lehrende sich eine Einreichung (also B.8.4 u. B.8.5) ansehen kann, muss er hierfür berechtigt werden. Dies ist dieselbe Autorisierung, die beim Aufgabenentwickler erwähnt wurde (s. Abbildung 12).

In der Detailübersicht von Java- oder C++-Aufgaben werden noch zusätzliche Funktionen angeboten, die bei R-Aufgaben nicht erhältlich sind. Der Lehrende kann manuell Punkte vergeben und ein Kommentar bzw. Feedback für den Studenten hinterlassen. Außerdem kann er sich die Lösung bzw. den Quelltext ansehen oder herunterladen.

Auf der Bearbeitungsseite ganz unten (s. B.3) gibt es eine weitere kleinere Einreichungsstatistik, in der die Einreichungen aller Studenten zusammengefasst sind. Zudem werden hier die Buttons „Alle Ergebnisse löschen“ und „Alle inkorrekten Ergebnisse löschen“ angeboten. Diese Buttons löschen zwar die Ergebnisse, jedoch starten die Checker die Tests automatisch neu, weil die Einreichungen von den Studenten im System weiterhin vorhanden sind. Aus diesem Grund findet der Use Case „Checker neu starten“ in Abbildung 17 seinen Platz. Die Buttons können also auch für einen Testneustart genutzt werden. Da sich die Checker aber deaktivieren lassen, können die Buttons ebenfalls ihre tatsächliche Funktion erfüllen.

Damit der Lehrende einen Überblick darüber bekommt, an welchen Stellen der Aufgabe es häufig zu Fehlern kommt, kann er sich eine Fehlerstatistik ansehen (s. B.8.6). Dort wird ihm angezeigt, welche Testfälle wie oft fehlgeschlagen sind. Dies kann dem Lehrenden helfen, seine Aufgabenstellung zu optimieren und eventuell besonders schwierige Stellen in der Aufgabenstellung zu überarbeiten.

4.3 Feedbackgenerierung

Für das Liefern von Feedback und Testen der Lösungen sind die Checker in JACK zuständig. Dabei gibt es für jeden Aufgabentyp verschiedene Checker (s. B.6). Damit diese funktionieren, müssen verschiedene Ressourcen eingebunden werden.

Im Folgenden werden die Checker für die Java Programmieraufgaben erläutert. Dabei wird aufgezeigt, wie diese zu konfigurieren sind und wie das Feedback erzeugt wird (s. B.9.2). Für C++ und R-Aufgaben gibt es jeweils einen dynamischen und einen statischen Checker, die ähnlich wie die von Java funktionieren.

Zunächst einmal besitzt jeder Checker Standardkonfigurationen (s. B.6). Dabei werden drei verschiedene Namen für einen Checker vergeben. Der Variablenname wird automatisch beim Erstellen des Checkers vergeben und wird zum Gewichten genutzt. Dieser wird mit einem Faktor multipliziert und schließlich mit den anderen Produkten addiert (Bsp. Checker1*0,5+Checker2*0,5). Doch nicht jeder Checker muss in die Gewichtung aufgenommen werden. Einige können nur dafür benutzt werden, um Feedback zu liefern, ohne dass diese zur erreichten Punktzahl beitragen [13]. Der Checker-Name ist nur für den Aufgabenentwickler sichtbar und dient als Unterscheidung gegenüber den anderen Checkern. Das Ergebnis-Label ist der Name, der dem Studenten in der Ergebnisübersicht angezeigt wird (s. B.9.2). Als weitere Standardkonfiguration lassen sich außerdem Feedback und Punkte, die von dem Checker ausgehen, vor dem Studenten verbergen, sodass diese in der Ergebnisübersicht nicht angezeigt werden.

In jedem Checker muss eine Source Datei eingebunden werden, die getestet werden soll. Hierzu wird für gewöhnlich die Working Sheet Datei (beschrieben in Kapitel 4.2) eingebunden. Es ist etwas verwunderlich, wieso eine Datei eingebunden wird, die lediglich ein Template ist. Hier wird jedoch nicht das Template getestet, sondern die Lösung des Studenten. Der Student muss bei einer Einreichung seiner Lösung immer eine Datei hochladen, die genauso heißt wie die Working Sheet Datei.

Für die Sprache Java stehen derzeit vier Checker zur Verfügung. Für genauere Details, wie diese Checker und deren Ressourcen und Ergebnislieferung aussehen, wird auf den Anhang verwiesen. Da einige Checker, wie beispielsweise der Java-Visualizer recht komplex sind, empfiehlt es sich beim Lesen den Anhang nicht ungeachtet zu lassen.

1. **Static Java Checker (s. B.6.1):** Dieser Checker benötigt drei verschiedene Ressourcen, wobei die Library File obligatorisch zum Einbinden von externen Klassen ist. Die Rule- und Source-File sind Pflicht, damit der Checker funktionieren kann. Ersteres ist eine XML-Datei, in der Testfälle und Feedback festgelegt werden können. Die Struktur dieser Datei hat zwei entscheidende Tags. In dem Query Tag wird der Testfall per GReQL [14] formuliert, der eine statische Überprüfung durchführt. Beispielsweise kann hier ein Testfall erzeugt werden, der den Code auf Namenskonventionen prüft

(Groß- und Kleinschreibung von Methode-, Klassen- oder Variablennamen etc.). Es ist auch möglich zu überprüfen, ob im Code leere If-Rümpfe existieren oder ob bestimmte Variablenzuweisungen durchgeführt wurden. Es gibt viele verschiedene statische Testmöglichkeiten, welche im JACK-Wiki[11] übersichtlich zusammengefasst wurden. In dem zweiten Tag wird dann das Feedback formuliert, das von dem Testfall ausgehen soll. Die Query- und Feedback-Tags befinden sich als Kindknoten in dem Rule-Tag. In diesem kann wiederum als Attribut die Punktzahl definiert werden, die von diesem Testfall ausgehen soll.

2. **Tracing Java Checker (s. B.6.2):** Dieser Checker produziert eine Tracing-Tabelle, in der zu sehen ist, welche Zuweisung die Variablen in den jeweiligen Zeilen bekommen haben. Zusätzlich können Testklassen eingebunden werden, die **dynamische Tests** durchführen und Feedback liefern. Diese müssen bestimmte Konventionen erfüllen, damit diese vom Checker akzeptiert werden. Dabei muss jede Klasse eine Instanz- bzw. Exemplarvariable besitzen, welche die Anzahl der Punkte hält. Jede erfolgreiche Testmethode addiert dann einen beliebigen Wert in diese Exemplarvariable. Die Testklasse muss außerdem eine getResult Methode besitzen, welche diese Exmplarvariable zurück liefert. Somit kann JACK die Punktzahl anzeigen, sobald alle Testfälle terminiert sind.

   Die Feedbacks werden über die Standardfunktion System.out.println beschrieben. Damit Errors und Warnings auf der Benutzeroberfläche von JACK angezeigt werden können, gibt es hierfür interne Klassen, mit denen ebenfalls ein Feedback formuliert werden kann (Bsp. TracingFramework.printError(„Feedback“)). Die Tracing Tabelle erzeugt nur Informationen zu Zeilen, die durch die Testfälle erreicht wurden. Innerhalb der Testfälle lässt sich das Tracing außerdem mit der internen Funktion switchTracingOff abschalten. Standardmäßig ist das Tracing jedoch immer aktiviert.

3. **Java Metric Checker (s. B.6.4):** Hier gibt es keine Konfigurationsmöglichkeiten. Es muss lediglich die Source Datei eingebunden werden. Hier werden die Anzahl der Methoden, Schleifen, Felder oder Anweisungen im Code ermittelt. Auch die Komplexität und Verschachtelungstiefe des Quelltextes wird berechnet. Alles zusammen wird dann als Feedback in der Ergebnisübersicht der Aufgabe angezeigt.

4. **Java Visualizer (s. B.6.3):** Als Visualisierung werden hier Objektdiagramme angezeigt. Damit das grafische Feedback erzeugt werden kann, müssen auch hier verschiedene Ressourcen eingebunden werden. Die Visualisierungskonfiguration ist eine XML, in der die Klassennamen unter dem Tag *classes* eingetragen werden. Diese Klassen werden für die grafische Darstellung benutzt. Dies sind auch die Klassen, die laut Aufgabenstellung vom Studenten implementiert werden müssen. Der Breakpoint-Tag enthält einen Klassennamen (den Testtreiber) als Attribut und *breakpoints* als Kindknoten. Die Breakpoints sind die Zeilennummer des Quelltextes der Testklasse und

[11] https://jack-community.org/wiki/index.php/GReQL_Regeln

beschreiben, zu welchem Zeitpunkt der Programmausführung die Visualisierung erzeugt werden soll. Der Testtreiber muss eine Main-Methode besitzen, in dem die nötigen Objekte für das Diagramm erzeugt werden.

Mit der Visualisierung kann der Student überprüfen, ob er seine Lösung richtig umgesetzt hat, denn hier werden Beziehungen und Referenzen der Klassen verdeutlicht. im Testtreiber (s. Anhang) ist zu sehen, dass verschiedene Telefonbücher mit verschiedenen Einträgen erzeugt werden. In der Visualisierung ist beispielsweise deutlich, dass das Telefonbuch „Syke“ die Einträge alphabetisch korrekt sortiert hat. Wenn diese Sortierung eine Anforderung der Aufgabe war, kann der Student das nun mit dieser Visualisierung überprüfen. Der Kopfeintrag bzw. der erste Eintrag ist somit „Burger", der ein Anwalt ist. Auch die Berufe sind alphabetisch geordnet, weshalb dieser Brancheneintrag auch als Kopfbranche dargestellt wurde. „Objekte, die zum Zeitpunkt der Erzeugung des Snapshots bereits vom Garbage Collector entfernt worden sind, werden mit einem roten Rand dargestellt“ [11].

4.4 Analyse des technischen Designs

In diesem Kapitel wird der technische Hintergrund von JACK betrachtet. Bei der Datenmodellierung werden mithilfe eines Klassendiagrammes wichtige Elemente des Quelltextes gezeigt und erklärt, wie diese in Verbindung stehen und welchen Nutzen sie haben. Im darauffolgenden Kapitel werden die wichtigsten Komponenten des System erklärt und dargestellt. Zudem wird erläutert, welche architektonischen Konzepte aus der Softwareentwicklung verwendet wurden.

4.4.1 Datenmodellierung

Die Datenmodellierung von JACK ist in Abbildung 18 als Klassendiagramm dargestellt.

Die Klassen werden von Michael Striewe in drei verschiedene Gruppen eingeteilt [15]. Die erste Gruppe befasst sich mit Aufgaben und deren Verwaltung in Kursen und Prüfungen. Bei der Zweiten geht es um die Einreichungen von Lösungen zu Aufgaben und deren Feedbacks (also Checker Konfigurationen etc.). Und bei der Dritten geht es um das Verwalten von Benutzern und deren Rechten. [15]

Die Hauptkomponente der ersten Gruppe ist das *AbstractExercise*, welches die Verwaltung von Aufgaben und Prüfungen übernimmt. Außerdem werden hier die Daten gesichert, die beim Erstellen der Aufgabe festgelegt werden, wie zum Beispiel Titel oder Evaluationsregeln bzw.

das Gewichten der Tests. Die Klasse *Exercise* erbt von *AbstractExercise*. Die *Exercise* Klasse enthält *ExerciseResource* und *CheckerConfiguration*. [15]

Die Beziehung der Klassen ist hier verwunderlich dargestellt, weil die Pfeile verkehrt herum gezeichnet sind. Eigentlich müsste der Pfeil von Exercise zu den beiden Klassen zeigen und nicht anders herum, damit die „enthält" Beziehung korrekt dargestellt werden kann. Nach Rücksprache mit Michael Striewe erkannte ich, dass die Pfeile eine „**gehört zu**" Beziehung bilden. Denn die Software ist in einer älteren EJB Version implementiert worden, wobei es sich bei den Klassen um Container handelt. Ressourcen **gehören** immer zu genau einer Aufgabe, d.h. dieselben Ressourcen werden nicht unter den Aufgaben geteilt.

Weiterhin legt die *CheckerConfiguration* fest, welche Ressourcen an welchem Checker und zu welchem Zweck übergeben werden, wobei nicht alle Ressourcen in den Checkern verfügbar sind. Beispielsweise kann der Java Metric Checker keine Ressourcen halten. Außerdem können gleiche Checker in die Aufgabe integriert werden. Beispielsweise können in einer Aufgabe mehrere Dynamic Java Checker existieren, welche verschiedene Ressourcen enthalten und somit verschiedene Tests ausführen und verschiedene Feedbacks liefern können [15]. Durch die Tag Klasse können Aufgaben Prüfungen und Kursen zugewiesen werden.

Bei der zweiten Gruppe kann zunächst das *AbstractSolution* betrachtet werden. Diese Klasse verwaltet die Lösungen zu einer Aufgabe, also die Ressourcen, welche vom Studenten eingereicht wurden. Zur Einreichung gehören ebenfalls Daten, wie z.B. Einreichungszeitpunkt. Außerdem werden hier die Feedbacks, welche vom Checker geliefert wurden oder manuell durch den Lehrenden hinzugefügt wurden, verwaltet. [15]

Solution erbt von *AbstractSolution* und ein *Result* (Ergebnis) **gehört** zu einer *Solution*. Im *ErrorRecord* werden Feedbacks und deren Bewertungen (Daumen hoch oder runter durch den Studenten) gehalten. Ein *Job* gehört ebenfalls zu einer *Solution* und zusätzlich zu einer *CheckerConfiguration*. Dabei handelt es sich um die Lösung des Studenten und die asynchron konfigurierten Checker. Für synchrone Checker werden keine Jobs erstellt. Hier wird die Lösung direkt übergeben und es wird sofort ein Ergebnis geliefert. Ein *Job* wird bei einer Ergebniserstellung genutzt und wieder gelöscht, sobald alle Ergebnisse vom Checker vorliegen. Checker können außerdem ein *SolutuionAttribute* liefern, damit sie Informationen untereinander austauschen können. [15]

Laut Rücksprache mit Michael Striewe, wird das *SolutuionAttribute* derzeit nicht verwendet und wurde deshalb im Kapitel 4.3 nicht behandelt. Aus der Kommunikation mit Striewe wurde deutlich, dass die Terminierungsreihenfolge der Checker nicht von vornherein festgelegt ist und somit kein Informationsaustausch möglich ist. Die Idee ist jedoch grundsätzlich gut, denn es wäre sinnvoll, wenn sich die Checker untereinander „helfen“ könnten, indem sie beispielsweise mitteilen, wie lange sie für einen Test benötigt haben oder welche Zeilen des Codes (engl. Code-Coverage) sie erreicht haben. Somit könnten Lasten verringert werden, weil ein Checker dann wissen würde, welche Testfälle nicht ausgeführt werden müssen, wenn diese bereits von einem anderen Checker abgedeckt sind.

Die letzte Gruppe besteht lediglich aus der *User* Klasse, wobei dieser ein Student, Lehrer oder Admin sein kann. Jeder dieser *User* besitzt einen Namen, ein Passwort und eine Rolle. Zusätzlich werden hier noch die Zugriffsrechte auf Kurse und Prüfungen verwaltet. *ParticipationRecord* regelt die Teilnahme an Prüfungen, wobei hier ein individuelles Passwort benötigt werden kann [15].

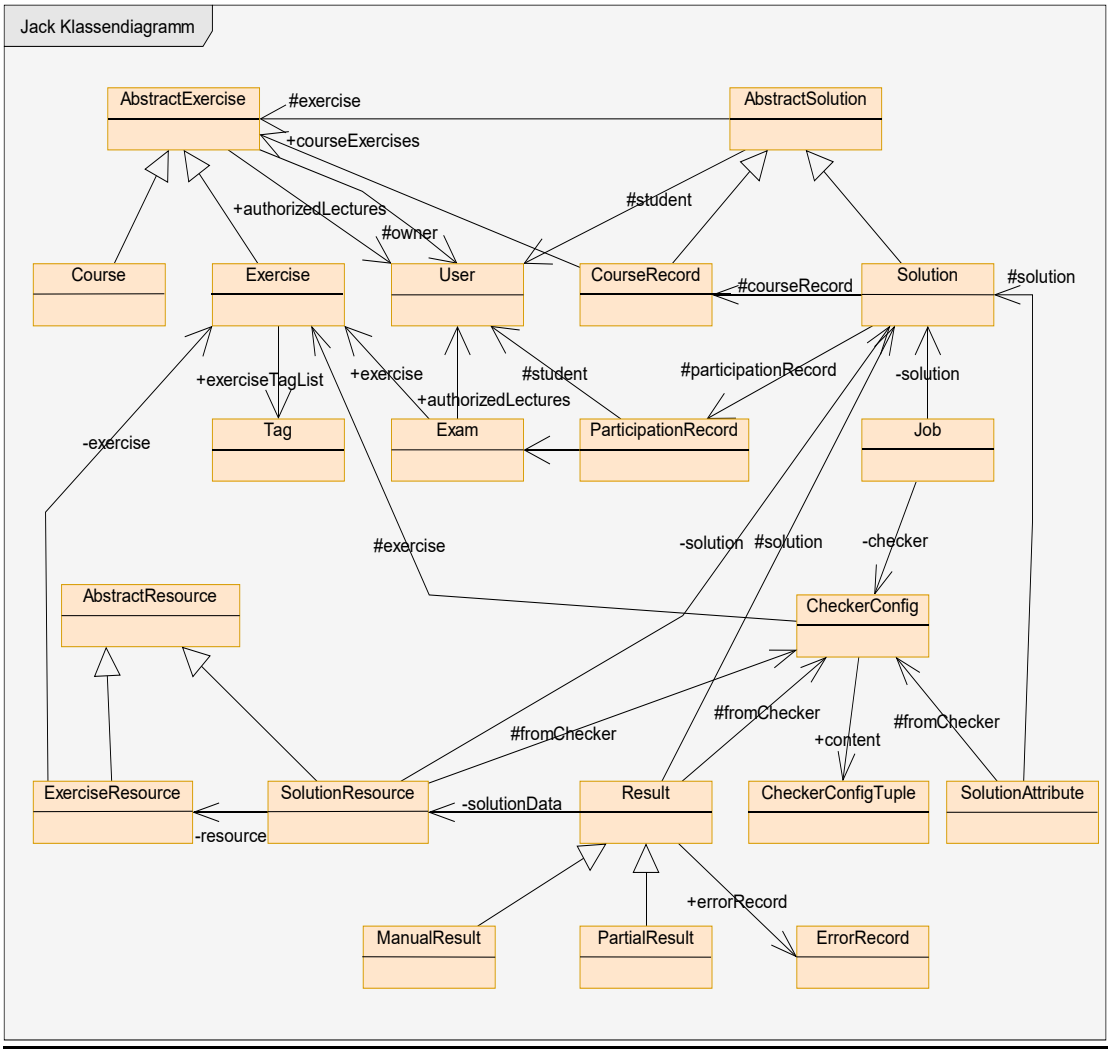


Abbildung 18: JACK Klassendiagramm – „Methoden und Variablen, sowie vielfache Assoziationen sind für die Lesbarkeit nicht mit enthalten“ [15, p. 29]. Die Pfeilbeschriftungen sind Exemplarvariablen und das Zeichen davor beschreibt die Sichtbarkeit, also private(-), public(+) oder protected(#).

4.4.2 System Architektur

In JACK wird das Konzept der Master Worker Architektur verwendet. In Abbildung 19 befindet sich der Master auf der linken Seite und kümmert sich um die Datenhaltung und Bereitstellung von Ressourcen für den User. Abhängig vom Aufgabentyp werden die Checker als asynchrone Jobs nacheinander auf den Worker Servern abgearbeitet oder direkt auf dem Master als

synchrone Checker. Synchrone Checker sind beispielsweise in Multiple Choice oder Fill-In Aufgaben zu finden, bei denen es nur jeweils einen Checker zur Auswahl gibt. Asynchrone Checker werden in Programmier- und Modellieraufgaben verwendet, in denen entschieden werden kann, welche Checker zum Einsatz kommen sollen [13].

Bei dem Master Server handelt es sich um eine Drei-Schichten-Architektur, wobei die Präsentationsschicht aus den Komponenten *Web-Frontend* und *Web-Service für Eclipse* gebildet wird. Die Vorteile einer Drei-Schichten-Architektur wurden bereits in Kapitel 3.3.2 beschrieben. Beispielsweise könnte eine weitere Komponente für das Bearbeiten der Aufgaben vom Smartphone heraus einfach in die Präsentationsschicht aufgenommen werden [13]. Die Logikschicht direkt darunter kommuniziert mit der Worker Seite über einen Web Service und speichert lokale Daten in der Datenhaltungsschicht. Zusätzlich befindet sich in der Ebene noch ein externer Authentifizierungsservice [13]. Weiterhin heißt es in [11]: „Der Master ist als EJB-Anwendung auf einem JBOSS Application Server realisiert und verwendet eine PostGreSQL-Datenbank zur Datenhaltung“.

Der Worker Server ist ein aktives System und holt sich die Jobs aus der Queue ab, die auf dem Master Server verwaltet wird. Außerdem besitzt ein Worker mehrere Checker und leitet die Aufgaben an diese weiter und liefert dann das Ergebnis zurück an den Server. Es können außerdem mehrere Worker Server existieren, wobei nicht jeder die gleichen Checker besitzen muss [11] [13].

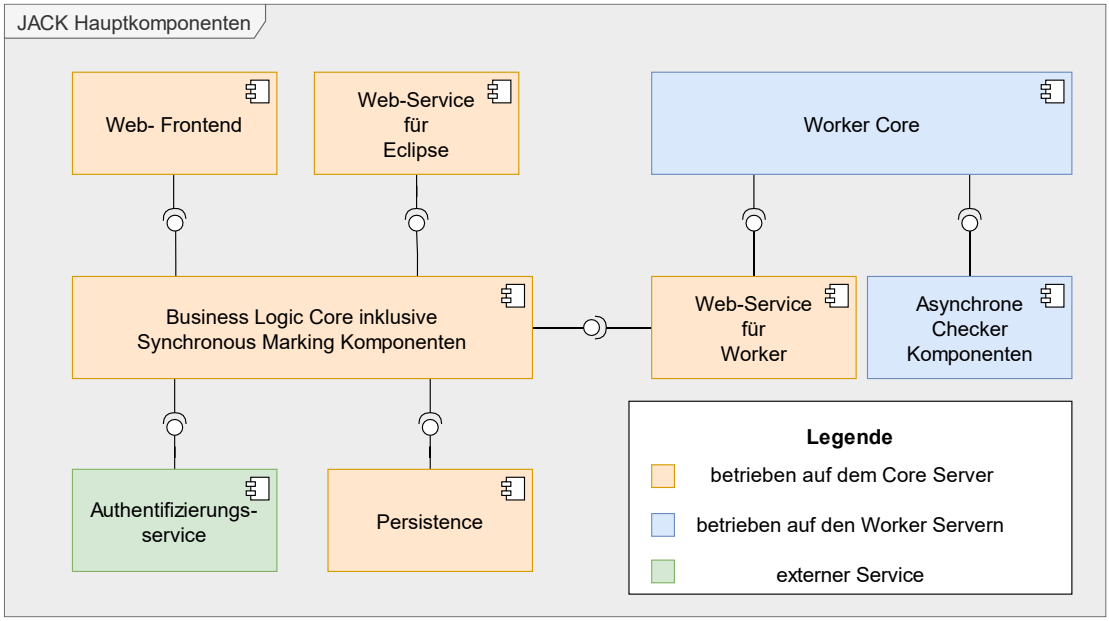


Abbildung 19 JACK: Komponentendiagramm [13]

5 Vergleich von JACK und ArTEMiS

In diesem Vergleichskapitel werden die beiden Plattformen JACK und ArTEMiS gegenübergestellt. Zunächst wird ausgewertet, welche Plattform welche Anforderungen aus Kapitel 2.2 erfüllt. Diese werden schließlich im darauffolgenden Kapitel als Vergleichskriterien genutzt. Dort wird erläutert, warum die Anforderungen erfüllt bzw. nicht erfüllt wurden. Dabei wird sich auch herausstellen, welche Plattform einige Funktionen besser oder schlechter umgesetzt hat.

5.1 Vergleich der Anforderungsumsetzungen

In den folgenden Tabellen werden gezeigt, welche Anforderungen aus Kapitel 2.2 von welcher Plattform umgesetzt wurden. Ein „++“ bedeutet, dass die Anforderung erfüllt wurde und ein „-

“ bedeutet das Gegenteil. Zusätzlich gibt es noch ein einzelnes „+“, welches darauf hindeutet, dass die Anforderung nur zum Teil abgedeckt wurde.

1. <u>Student</u>

Tabelle 2: Anforderungen des Studenten werden zwischen JACK und ArTEMiS verglichen.

Anforderung	***JACK***	***ArTEMiS***
FR1.1 Code-Editor	+	+
FR1.2 Bearbeitung & Einreichung	+	++
FR1.3 Teamarbeit	-	++
FR1.4 Versionskontrolle	-	++
FR1.5 Feedback	++	++
FR1.5a Visuelles Feedback	+	-
FR1.5b Textuelles Feedback	++	++
FR1.6 Hinweise	+	++
FR1.7 Code Reviews	-	-
FR1.8 Sprache	-	-
FR1.9 Lösungsfortschritt	-	++
FR1.10 Anmeldung	++	++

2. Aufgabenentwickler

Tabelle 3: Anforderungen des Aufgabenentwicklers werden zwischen JACK und ArTEMiS verglichen.

Anforderung	***JACK***	***ArTEMiS***
FR2.1a Einzelaufgaben	++	++
FR2.1b Teamaufgaben	-	++
FR2.1c Wettbewerbsaufgaben	-	-
FR2.2 Kategorie	++	++
FR2.3 Aufgaben teilen	++	-
FR2.4 Material	++	++
FR2.5 Sprache	+	+
FR2.6 Tests	++	++

3. Lehrender

Tabelle 4: Anforderungen des Lehrenden werden zwischen JACK und ArTEMiS verglichen.

Anforderung	***JACK***	***ArTEMiS***
FR3.1 Kursmanagement	+	++
FR3.2a Zeitlimitierung	+	-
FR3.2b Bearbeitungsreihenfolge	-	-
FR3.2c Sprache	-	-
FR3.3 Belohnung	-	-
FR3.4 Lösungsfortschritt beobachten	+	++
FR3.5 Material	-	-

5.2 Unterschiede in der Umsetzung der Anforderungen

Nach der tabellarischen Übersicht, welche Anforderungen erfüllt wurden, werden diese als Vergleichskriterien genutzt. Dabei wird diskutiert, wie die Anforderungen von den jeweiligen Plattformen umgesetzt wurden. Die Systeme werden somit gegenübergestellt und verglichen.

Student (Bezug zu Tabelle 2)

In beiden Plattformen meldet sich der Student mit seinen Benutzerdaten aus der Universität an (s. FR1.10). In ArTEMiS kann er die Aufgaben dann im Web-Browser bearbeiten, sofern dies vom Aufgabenentwickler freigeschaltet wurde (s. Abbildung 1). In ArTEMiS können die Benutzerdaten mit anderen Rechten erweitert werden. Somit kann ein Student auch ein Lehrender sein und braucht nicht zwei Benutzerdaten. In JACK können nur Studentenkonten oder Lehrendenkonten existieren.

ArTEMiS stellt einen Online-Code-Editor zur Verfügung, wobei JACK dies nur für Programmieraufgaben anbietet, die in der Sprache R bearbeitet werden können (s. FR1.1). Der Student kann sich bei beiden Systemen nicht aussuchen, in welchen Sprachen er die Aufgabe lösen möchte (s. FR1.8), denn dies wird bereits bei der Erstellung der Aufgabe festgelegt.

Beide Editoren verfügen zwar über ein Terminal, das über Kompilierfehler informiert, jedoch fehlen ihnen Grundfunktionalitäten, wie Syntaxhervorhebung, Autovervollständigung oder „Suchen und Ersetzen“. In ArTEMiS greift die Autovervollständigung nur, wenn das Wort bereits einmal verwendet wurde. Es erkennt somit keine sprachspezifischen Bibliotheken. Bei JACK hingegen kann der Lehrende wahlweise ein Auswahlmenü mit verschiedenen Standardfunktionen bereitstellen, das jedoch nicht die Funktion einer Autovervollständigung erfüllt. Die Syntaxfehler werden in beiden Systemen nur durch das Terminal signalisiert. In ArTEMiS geschieht dies, wenn die Aufgabe abgegeben wurde. Es gibt hier also keine Möglichkeiten, seinen Code im Online-Editor vor der Einreichung zu überprüfen. Da beide Plattformen dafür gebaut sind, Unterricht zu betreiben, halte ich dies nicht für sinnvoll, denn als Student möchte ich nicht unbedingt, dass der Lehrende alle meine Zwischenlösungen sehen kann. In JACK hingegen kann der Code vor der Abgabe ausgewertet werden.

In ArTEMiS wirkt der Editor für das Auge ansprechender und moderner und erweckt den Eindruck, dass es größeres Potential hat. Der Student kann dort außerdem zwischen verschiedenen Dateien navigieren und die Lösung in mehreren Dokumenten abgeben (s. FR1.2). Zusätzlich kann er an einem Fortschrittsbalken sehen, welche Tests erfolgreich waren oder fehlgeschlagen sind (s. FR1.9). In JACK wird kein Lösungsfortschritt während der Aufgabenbearbeitung angezeigt. Das Einreichen der Lösung in mehreren Dateien ist nur bei Aufgabentypen möglich, für die es keinen Online-Editor gibt (Java, C++ etc.).

In ArTEMiS wird die Lösung in ein Repository hochgeladen, welches dem Studenten zur Verfügung gestellt wird (s. FR1.4). Auch wenn dieser im Online-Editor arbeitet, wird der Stand des Repositories aktualisiert, wenn er auf den Einreichungsbutton klickt. In JACK werden keine Repositories verwaltet und der Student muss alles lokal sichern. Die Lösungen muss er dann in der Plattform hochladen. Nach der Einreichung wird bei beiden Plattformen direkt ein Feedback erhalten (s. FR1.5). Dies erfolgt automatisch und relativ schnell. Als Vergleich zu

einer bekannteren Plattform, liefert Codingame[12] schnelleres Feedback. Ich kann jedoch nicht einschätzen, wie lange die Plattformen im Worst Case Szenario brauchen, beispielsweise wenn sehr viele Studenten in der Plattform aktiv sind, da ich die Tests nur auf Demoservern ausführen durfte. Ob das Feedback verständlich bzw. hilfreich ist, hängt vom Autor der Tests (In JACK sind es die Checker) ab. In ArTEMiS wird das Feedback immer textuell geliefert (s. FR1.5b). In JACK ist zusätzlich eine Visualisierung möglich (s. FR1.5a), jedoch ist dies immer in Form von Objektdiagrammen (erläutert in Kapitel 4.3) und nur für den Aufgabentyp Java verfügbar.

Während der Aufgabenbearbeitung werden dem Studenten bei beiden Plattformen Hinweise eingeblendet (s. FR1.6). In JACK ist dies jedoch nur bei der Bearbeitung von R-Aufgaben realisiert. In ArTEMiS wird die Funktion bei allen Aufgabentypen unterstützt. Es zeigt sich, dass beide Systeme das gleiche Verständnis darüber haben, was ein Hinweis ist. Denn sowohl bei ArTEMiS, als auch bei JACK können die Hinweise jederzeit, auch direkt zu Anfang einer Aufgabe, abgerufen werden. Eine andere Definition wäre, dass der Hinweis automatisch erscheint, wenn der Student mehrmals falsche Lösungen eingereicht hat. Ich finde die Umsetzung der beiden Systeme jedoch sehr gut, denn als Student habe ich auch den Ehrgeiz, die Aufgabe ohne Hilfestellung zu lösen. Selbst wenn ich die Lösung mehrmals falsch abgebe, möchte ich eventuell keine Tipps bekommen. Bei den Plattformen kann der Student dies für sich frei entscheiden.

In ArTEMiS können die Aufgaben im Team bearbeitet werden (s. FR1.3). Dies ist jedoch nur möglich, wenn die Aufgabe als Teamaufgabe erstellt wurde. Diese Art von Aufgaben können vom Studenten nur dann gestartet werden, wenn er Teil eines Teams ist und dieses wiederum der Aufgabe zugewiesen ist. Die Teams werden von den Lehrenden erstellt und verwaltet. Der einzige Unterschied zwischen einer Einzelaufgabe und Teamaufgabe ist, dass mehrere Studenten eine gemeinsame Lösung abgeben und sich ein einziges Repository teilen. Somit kann der Lehrende sehen, welcher Student welche Änderungen vorgenommen hat. In JACK gibt es keine Teamaufgaben.

[12] https://www.codingame.com/start

Die Lösungen können in beiden Plattformen nicht zwischen den Studenten bzw. Teams ausgetauscht werden, und somit ist auch kein gegenseitiges Code Review möglich (s. FR1.7).

Aufgabenentwickler (Bezug zu Tabelle 3)

ArTEMiS ist also in der Erstellung der Aufgaben vielfältiger, weil es im Gegensatz zu JACK, zusätzlich zu Einzelaufgaben, auch die Erstellung von Teamaufgaben ermöglicht (s. FR2.1a u. FR2.1b). Andere Aufgabentypen, wie Wettbewerbsaufgaben, werden in beiden Systemen nicht unterstützt (s. FR2.1c).

Wie die Aufgaben erstellt werden, wurde bereits in Kapitel 3.2 und 4.2 erläutert. In beiden Systemen können der Aufgabe Beschreibungen, Links etc. hinzugefügt werden (s. FR2.4). Während in JACK Materialien (PDF etc.), in Form von „Sheets“ zum Download zur Verfügung gestellt werden können, ist dies in ArTEMiS möglich, indem diese in das Template-Repository (erklärt in Kapitel 3.2) hochgeladen werden (s. FR3.5FR2.4).

Eine weitere Ähnlichkeit besteht darin, dass die Aufgaben erst erstellt werden und dann nochmals überarbeitet werden müssen, damit diese Sinn ergeben. Dies ist meiner Meinung nach umständlich durchdacht worden.

In JACK gibt es hierfür sogar zwei verschiedene Menüs. In dem Ersten (s. B.2) wird die Aufgabe erstellt und im Zweiten (s. B.3) wird die Aufgabe überarbeitet. Das erste Menü ist jedoch überflüssig, weil sich im Zweiten alle Konfigurationen befinden, die auch im ersten Menü enthalten sind.

In ArTEMiS gibt es zwar nur eine Oberfläche (s. A.1 u. A.2), jedoch muss der Entwickler nach dem Anlegen der Aufgabe diese nochmal aufrufen, weil einige Funktionen erst nach Erstellung der Aufgabenstellung freigeschaltet werden. Beispielsweise ist das Hinzufügen von Tests (s. FR2.6) in die Aufgabenbeschreibung erst dann möglich, wenn ein Repository für die Tests existiert. Dieses Repository wird jedoch erst nach dem Anlegen der Aufgabe erstellt.

Wie die Tests in JACK erstellt werden, wurde bereits in Kapitel 4.3 ausführlich besprochen. Hier werden für verschiedene Programmiersprachen verschiedene Checker bereitgestellt, wobei JACK nur Java, C++ und R unterstützt. In ArTEMiS sind mehr Sprachen verfügbar (s.

FR2.5), darunter auch modernere, wie OCaml oder Kotlin. Sprachen wie C++ oder R werden wiederum nicht unterstützt. Außerdem kann eine Aufgabe in beiden Plattformen nur in einer Sprache entwickelt werden.

Beim Erstellen der Aufgabe ist das Hinzufügen einer Kategorie in beiden Systemen obligatorisch (s. FR2.2). Jedoch zeigt sich, dass beide Systeme ein anderes Verständnis darüber haben, was eine Kategorie ist. In ArTEMiS können einer Aufgabe zwei Kategorien hinzugefügt werden und diese werden bei der Aufgabenauswahl im Kurs angezeigt (s. A.16). In JACK muss für eine Aufgabe nicht zwingend ein Kurs existieren. Hier werden Aufgaben und Kurse in Kategorien eingeteilt (s. B.7). Kurz zusammengefasst bedeutet es für JACK, dass sich Aufgaben und Kurse innerhalb einer Kategorie befinden und für ArTEMiS, dass sich Kategorien in Aufgaben befinden, welche ausschließlich in Kursen existieren können. Zusätzlich soll erwähnt werden, dass es in beiden Systemen keine Funktion gibt, die es ermöglicht, nach Kategorien zu filtern oder zu suchen. Außerdem gibt es in JACK noch eine weitere Art von Kategorie, nämlich das Hinzufügen von Tags. Wie bereits in Kapitel 4.2 erklärt, werden diese jedoch nur benutzt, um Aufgaben einem Kurs hinzuzufügen.

Bis hierhin wird deutlich, dass beide Systeme auch einen Kurs anders definieren. Denn während Aufgaben in ArTEMiS innerhalb von Kursen entwickelt werden, werden in JACK Aufgaben entwickelt, um sie Kursen hinzuzufügen. In ArTEMiS sind einem Kurs mehrere Lehrende und Aufgabenentwickler zugewiesen und alle haben Zugriff auf jede entwickelte Aufgabe innerhalb des Kurses. Sie teilen sich somit zwingend alle Aufgaben (s. FR2.3), d.h. alle in dem Kurs können die Aufgabenstellung überarbeiten. In JACK können die Entwickler ihre Aufgaben wahlweise für Andere freigeben. Jeder Berechtigte kann die Aufgabenstellung überarbeiten.

Lehrender (Bezug zu Tabelle 4)

Das Erstellen von Kursen ist also in beiden Systemen möglich (s. FR3.1). Im Gegensatz zu JACK können in ArTEMiS die Studenten und Teams verwaltet werden, d.h. sie können nur in Kursen aktiv sein, für die sie der Lehrende berechtigt. In JACK kann solch eine Einschränkung nicht eingestellt werden. Hier hat der Student Zugriff auf alle Aufgaben und Kurse.

Auch Limitierungen, wie die Bearbeitungsreihenfolge, können in beiden Systemen nicht festgelegt werden (s. FR3.2b). Und auch die Sprache, in der die Aufgabe gelöst werden soll, kann der Lehrende nicht verändern, weil dies bei der Entwicklung festgelegt wird (s. FR3.2c). In JACK kann er für eine einzelne Aufgabe keine Bearbeitungsdauer festlegen (s. FR3.2a). Wenn die Aufgabe jedoch einem Kurs zugewiesen wird, kann er für den Kurs eine zeitliche Begrenzung bestimmen. Nun wäre es denkbar, einen Kurs zu erstellen, welcher eine einzige Aufgabe enthält, damit diese Aufgabe eine zeitliche Bearbeitungsgrenze hat. Jedoch ist das Erstellen eines Kurses mit einer einzigen Aufgabe nicht ganz einfach (dies wurde in Kapitel 4.2 erläutert). In ArTEMiS kann für jede Aufgabe ein Ablaufdatum eingestellt werden, aber keine bestimmte Uhrzeit. Dieses Ablaufdatum wird wiederum bei der Erstellung der Aufgabe festgelegt. Der Lehrende kann somit keine eigene Ablaufzeit definieren.

In ArTEMiS können die Lehrenden nur Aufgaben aus Kursen in andere Kurse importieren, wenn sie Mitglied des Kurses sind. In JACK kann er hingegen alle Aufgaben, welche auf der Plattform existieren, in seinen Kurs integrieren. In JACK existiert somit ein Aufgabenpool, auf den jeder Zugriff hat. Mit Zugriff ist gemeint, dass jeder die Aufgabe benutzen, aber nicht verändern kann. Wie bereits im letzten Abschnitt erklärt, erteilt der Entwickler Berechtigungen, damit seine Aufgabenstellung von anderen überarbeitet werden kann.

In JACK ist das Bereitstellen von Zusatzmaterialien somit auch nicht ohne weiteres für jeden möglich (s. FR3.5). Denn der Lehrende kann nur Ressourcen hinzufügen, wenn er berechtigt ist, die Aufgabenstellung zu überarbeiten. Da in ArTEMiS jeder die Aufgabenstellung überarbeiten kann, gibt es hier solch ein Problem nicht. Nach meiner Meinung sollte der Lehrende nicht im Werk des Entwicklers arbeiten. Bei getrennter Betrachtung beider Stakeholder wäre es sinnvoll, wenn es in beiden Systemen eine besondere Ansicht für den Lehrenden gäbe, in der er auch Zusatzmaterial bereitstellen kann.

Ein Extramenü gibt es jedoch in beiden Systemen, um die Lösungen der Studenten anzusehen (s. FR3.4). In JACK ist dies jedoch auch nur für den Lehrenden möglich, welcher eine Berechtigung zum Überarbeiten der Aufgabenstellung hat. Denn die Lösungsübersicht kann nur über das Menü erreicht werden, in dem die Aufgabenstellung überarbeitet werden kann. In ArTEMiS hingegen, kann jeder Lehrende im Kurs alle Lösungen einsehen.

Leider gibt es in beiden Plattformen keine Möglichkeit, um eine besonders gute Lösung des Studenten zu belohnen (s. FR3.3). In ArTEMiS gibt es jedoch ein Menü für den Lehrenden, in dem er einzelne Tests gewichten kann. Das heißt, er kann unabhängig vom Entwickler bestimmen, in welchem Teil der Aufgabe der Student mehr Punkte bekommen soll. Zwar kann dadurch nicht die Schönheit oder Effizienz der Lösung belohnt werden, aber dafür wird der Student angeregt, besonders schwierige Teile der Aufgabe zu lösen, um ein besseres Ergebnis zu erlangen. In JACK ist es ebenso möglich, die Tests zu gewichten, jedoch nur in der Entwicklersicht.

6 Fazit

In dieser Arbeit ging es um zwei Hauptaspekte, die Analyse und den Vergleich von JACK und ArTEMiS.

Zu Beginn wurden einige Definitionen festgelegt. Die Stakeholder dienten dazu, um die Systeme aus verschiedenen Blickwinkeln betrachten zu können. Dabei spielten der Aufgabenentwickler, Lehrende und Student eine tragende Rolle. Die Anforderungen wurden benutzt, damit die Systeme unter bestimmten Kriterien miteinander verglichen werden konnten.

Die Plattformen wurden zunächst getrennt voneinander betrachtet. bei der Funktionsanalyse wurde die Erstellung (durch Aufgabenentwickler), Verwaltung (durch Lehrenden) und Bearbeitung (durch Studenten) von Programmieraufgaben erläutert. Es hat sich gezeigt, dass beide Systeme einen großen Umfang an Funktionen anbieten. Mithilfe von Use-Case Diagrammen konnten alle Möglichkeiten übersichtlich dargestellt werden.

Als nächstes wurde jeweils der technische Hintergrund betrachtet und gezeigt, wie der Quelltext des Systems aufgebaut ist und welche architektonischen Konzepte eingesetzt wurden. Bei der Quelltextanalyse wurden die Beziehungen der Klassen dargestellt. Und bei der System Architektur wurde gezeigt, welche internen und externen Komponenten benutzt wurden.

Die zu Beginn festgelegten Anforderungen kamen im letzten Kapitel zum Einsatz. Dabei wurde verglichen, welche Plattform welche Anforderungen erfüllt und wie diese jeweils umgesetzt wurden. Dadurch wurde deutlich, dass die Systeme teilweise Dinge ganz anders definieren und dass sie teilweise verschiedene Ziele verfolgen.

Literaturverzeichnis

[1] N. Grandraß, T. Hinrichs und A. Schmolitzky, „Towards An Online Programming Platform Complementing Software Engineering Education,“ in *Tagungsband 17. Workshop “Software Engineering im Unterricht der Hochschulen"*, 2020, [Online]. Available: http://ceur-ws.org/Vol-2531/paper05.pdf.

[2] M. Wauligmann, „Team-based Exercises in Artemis,“ Master Thesis, Technical University Munich, June 2020.

[3] M. Striewe und M. Goedicke, „Visualizing Data Structures in an E-Learning System,“ in *Proceedings of the 2nd International Conference on Computer Supported Education (CSEDU) , Volume 1*, Valencia, Spain , 07 - 10 April 2010, pp. 172-179. [Online] Available: https://www.researchgate.net/publication/221130376_Visualizing_Data_Structures_in_an_e-Learning_System.

[4] B. Bruegge und A. H. Dutoit, Object-Oriented Software EngineeringUsing UML, Patterns, and Java, One Lake Street Upper Saddle River, NJ, United States: Prentice Hall Press, ISBN: 978-0-13-606125-0, August 2009. [Online] Available: https://iran-lms.com/images/images/Books/PDF/Object-Oriented-Software-Engineering-Using-UML-Patterns-and-Java-Prentice-Hall-2010-Bernd-Bruegge-Allen-H.-Dutoit.pdf.

[5] D. Münch, „Conducting Interactive Programming Exercises in Large Lectures," Master Thesis, Technical University Munich, November 2016.

[6] S. Krusche und A. Seitz, „Artemis-An Automatic Assessment Management System for Interactive Learning," in *49th Technical Symposium on Computer Science Education (SIGCSE 2018)*, Baltimore - USA, February 2018. [Online]. Available: https://ase.in.tum.de/lehrstuhl_1/research/paper/krusche2018artemis.pdf.

[7] J. T. Behnke, „Extension of Programming Exercises in ArTEMiS," Master Thesis, Technical University Munich, October 2019.

[8] T. U. Munich, „artemis platform readthedocs," [Online]. Available: https://artemis-platform.readthedocs.io.

[9] „artemis-platform.readthedocs," [Online]. Available: https://artemis-platform.readthedocs.io. [Zugriff am 23 04 2021].

[10] A. S. Tanenbaum, Verteile Systeme – Prinzipien und Paradigmen, 2 aktualisierte Auflage Hrsg., München: Pearson Studium, ISBN: 978-3-8273-7293-2, 2008.

[11] O. J. Bott, P. Fricke, U. Priss und M. Striewe, „Automatisierte Bewertung in der Programmierausbildung," *Digitale Medien in der Hochschullehre*, Bd. 6, pp. 143-156, ISBN 978-3-8309-3606-0, 2017.

[12] B. Otto, T. Massing, N. Schwinning, N. Reckmann, A. Blasberg, S. Schumann, C. Hanck und M. Goedicke, „Evaluation einer Statistiklehrveranstaltung mit dem JACK R-Modul," in *Die 15. e-Learning Fachtagung Informatik*,, Gesellschaft für Informatik, Bonn, 2017.

[13] M. Striewe, „An architecture for modular grading and feedback generation for complex exercises,“ in *Science of Computer Programming*, Elsevier, pp. Volume 129, 1 November 2016, Pages 35-47 [Online] Available: https://www.sciencedirect.com/science/article/pii/S0167642316300260.

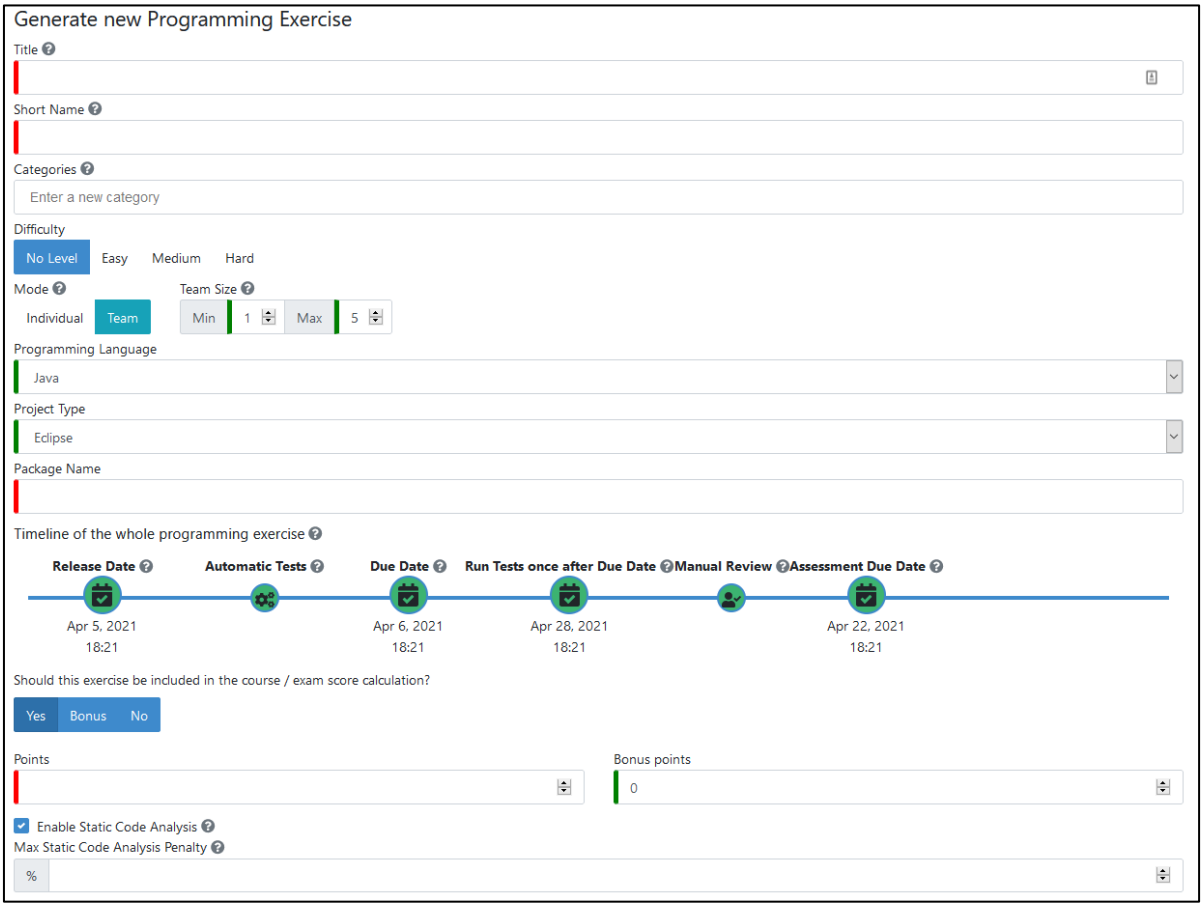
[14] J. Ebert, D. Bildhauer, T. Horn und V. Riediger, „Uni Koblenz Landau,“ [Online]. Available: https://www.uni-koblenz-landau.de/de/koblenz/fb4/ist/rgebert/research/graph-technology/GReQL. [Zugriff am 12 04 2021].

[15] M. Striewe, „Automated Analysis of Software Artefacts– A Use Case in E-Assessment,“ Dissertation, Universität Duisburg Essen, December 2014. [Online]. Available: https://duepublico2.uni-due.de/servlets/MCRFileNodeServlet/duepublico_derivate_00038402/Dissertation.pdf .

Anhang

A Anhang: ArTEMiS Benutzeroberfläche

A.1 Aufgabe erstellen Teil 1

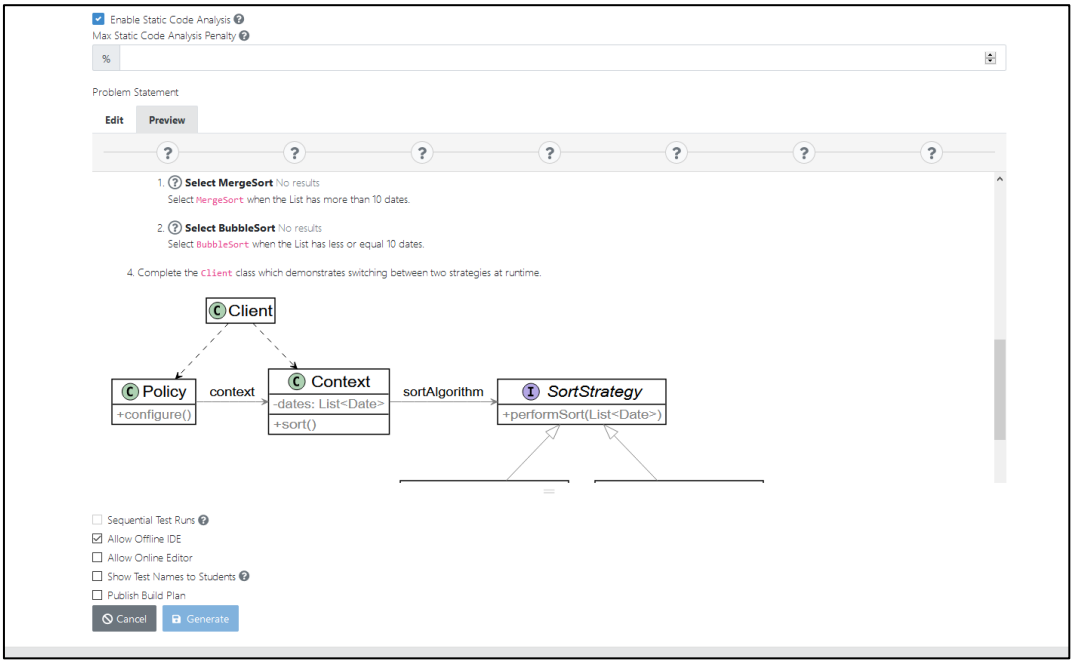


Anhang A.1 In dem Zeitplan wurden alle möglichen Fristen gesetzt und alle Funktionen aktiviert, weshalb diese grün angezeigt werden. Mit Ausnahme von Automatic Tests, dieses ist immer gesetzt und kann nicht ausgeschaltet werden.

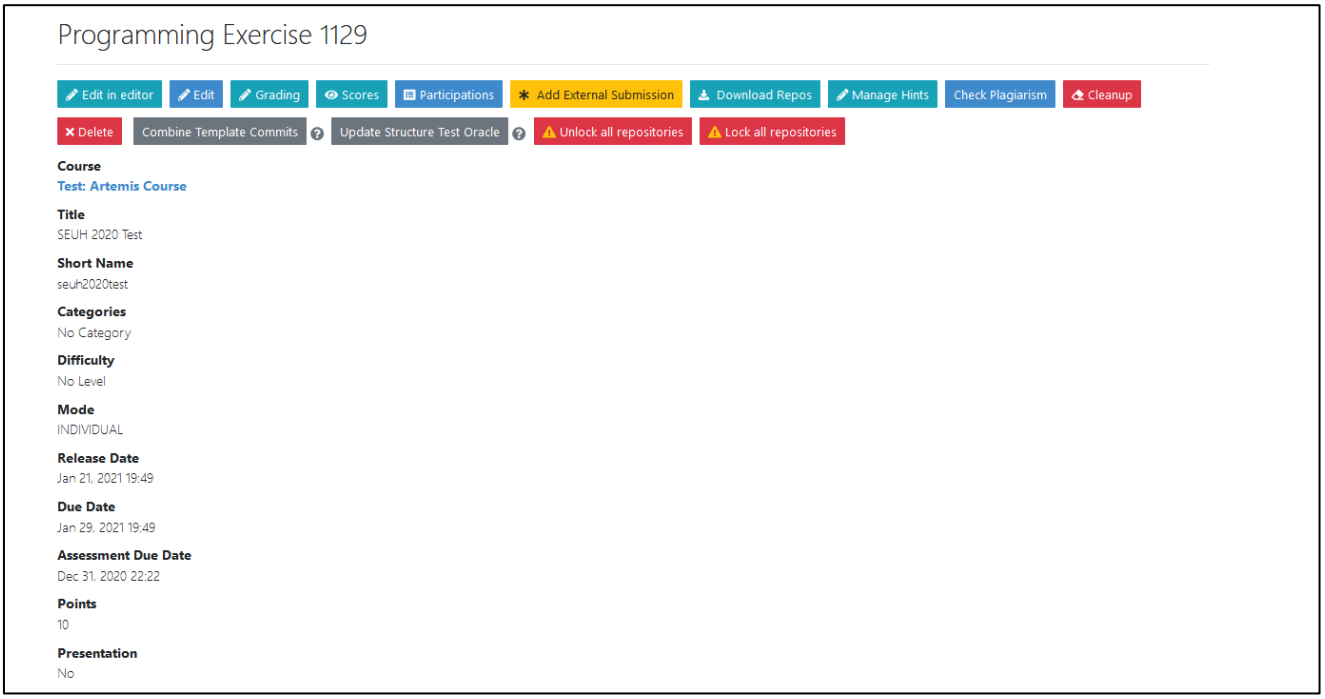
A.2 Aufgabe erstellen Teil 2: MarkDown-Editor

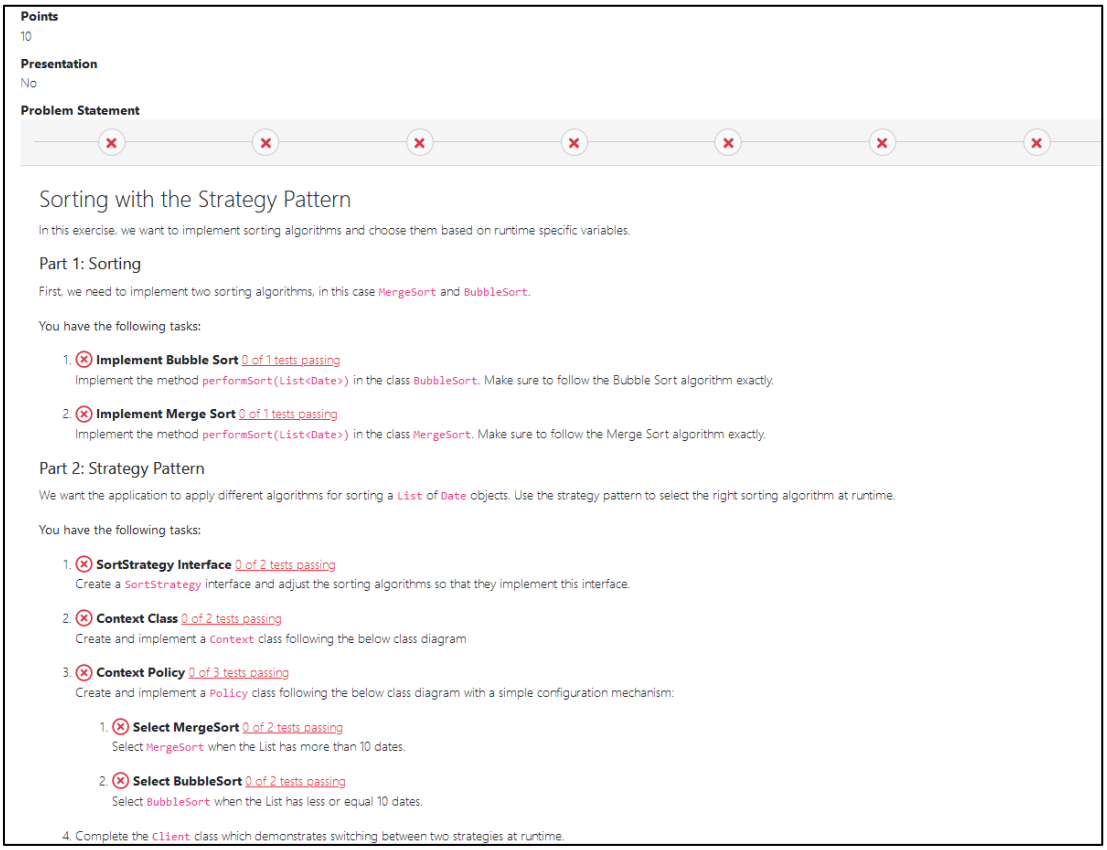
Enable Static Code Analysis
Max Static Code Analysis Penalty
%
Problem Statement
Edit
Preview
Style
Color
Formula
[task] Task
Insert Test Case
Add task specific hint
2. [task][Context Class](testAttributes[Context],testMethods[Context])
Create and implement a `Context` class following the below class diagram
3. [task][Context Policy](testConstructors[Policy],testAttributes[Policy],testMethods[Policy])
Create and implement a `Policy` class following the below class diagram with a simple configuration mechanism:
1. [task][Select MergeSort](testClass[MergeSort],testUseMergeSortForBigList)
Select `MergeSort` when the list has more than 10 dates.
2. [task][Select BubbleSort](testClass[BubbleSort],testUseBubbleSortForSmallList)
Select `BubbleSort` when the list has less or equal 10 dates.
4. Complete the `Client` class which demonstrates switching between two strategies at runtime.
@startuml
skinparam ArrowFontSize 16
skinparam shadowing false
skinparam classAttributeIconSize 0
skinparam class {
FontSize 20
BackgroundColor white
AttributeFontSize 16
BorderColor black
ArrowColor black
}
class Client {
}
Attach files by dragging & dropping or selecting them.
Sequential Test Runs
Allow Offline IDE
Allow Online Editor
Show Test Names to Students
Publish Build Plan
Cancel
Generate

A.3 Aufgabenbeschreibung



A.4 Aufgabenübersicht (View)





Points
10

Presentation
No

Problem Statement

Sorting with the Strategy Pattern

In this exercise, we want to implement sorting algorithms and choose them based on runtime specific variables.

Part 1: Sorting

First, we need to implement two sorting algorithms, in this case `MergeSort` and `BubbleSort`.

You have the following tasks:

1. **Implement Bubble Sort** 0 of 1 tests passing
   Implement the method `performSort(List<Date>)` in the class `BubbleSort`. Make sure to follow the Bubble Sort algorithm exactly.
2. **Implement Merge Sort** 0 of 1 tests passing
   Implement the method `performSort(List<Date>)` in the class `MergeSort`. Make sure to follow the Merge Sort algorithm exactly.

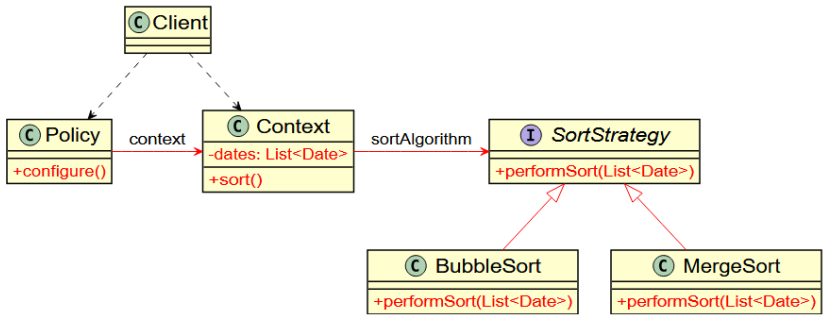
Part 2: Strategy Pattern

We want the application to apply different algorithms for sorting a `List` of `Date` objects. Use the strategy pattern to select the right sorting algorithm at runtime.

You have the following tasks:

1. **SortStrategy Interface** 0 of 2 tests passing
   Create a `SortStrategy` interface and adjust the sorting algorithms so that they implement this interface.
2. **Context Class** 0 of 2 tests passing
   Create and implement a `Context` class following the below class diagram
3. **Context Policy** 0 of 3 tests passing
   Create and implement a `Policy` class following the below class diagram with a simple configuration mechanism:
   1. **Select MergeSort** 0 of 2 tests passing
      Select `MergeSort` when the List has more than 10 dates.
   2. **Select BubbleSort** 0 of 2 tests passing
      Select `BubbleSort` when the List has less or equal 10 dates.
4. Complete the `Client` class which demonstrates switching between two strategies at runtime.

4. Complete the `Client` class which demonstrates switching between two strategies at runtime.



Part 3: Optional Challenges

(These are not tested)

1. Create a new class `QuickSort` that implements `SortStrategy` and implement the Quick Sort algorithm.
2. Make the method `performSort(List<Dates>)` generic, so that other objects can also be sorted by the same method. **Hint:** Have a look at Java Generics and the interface `Comparable`.
3. Think about a useful decision in `Policy` when to use the new `QuickSort` algorithm.

Run Tests once after Due Date

Enable Static Code Analysis
false

Template Repository Url
https://bitbucket.ase.in.tum.de/scm/ATCSEUH2020TEST/atcseuh2020test-exercise.git

Solution Repository Url
https://bitbucket.ase.in.tum.de/scm/ATCSEUH2020TEST/atcseuh2020test-solution.git

Test Repository Url
https://bitbucket.ase.in.tum.de/scm/ATCSEUH2020TEST/atcseuh2020test-tests.git

Template Build Plan Id
ATCSEUH2020TEST-BASE

Template Build Plan Id
ATCSEUH2020TEST-BASE

Solution Build Plan Id
ATCSEUH2020TEST-SOLUTION

Sequential Test Runs
false

Publish Build Plan
false

Allow Offline IDE
true

Allow Online Editor
true

Show Test Names to Students
false

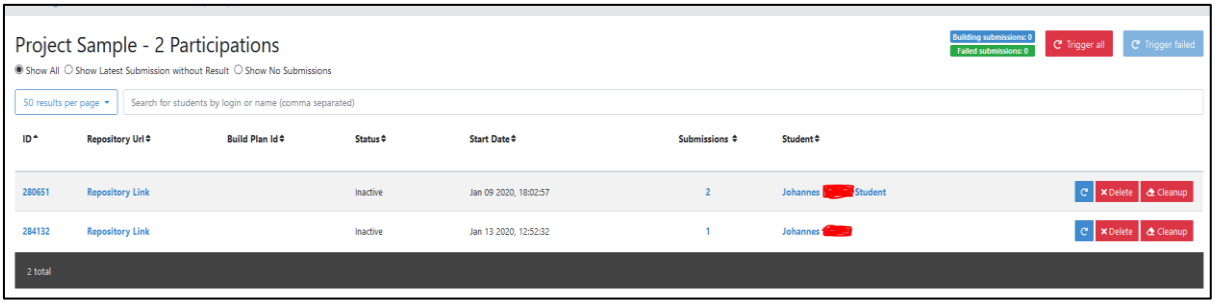
Programming Language
Java

Package Name
de.tum.in

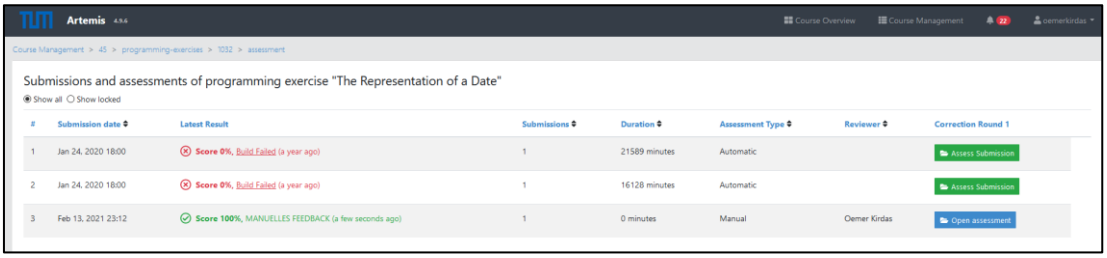
Template Result
Score 0%, 13 of 13 failed (a year ago)

Solution Result
Score 100%, 13 of 13 passed (a year ago)

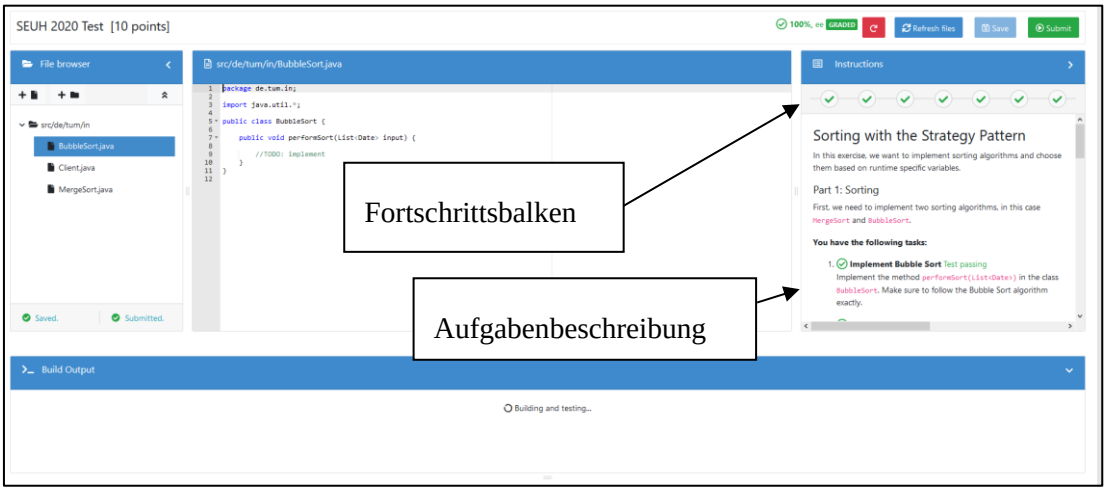
A.5 Teilnahmen an einer Aufgabe



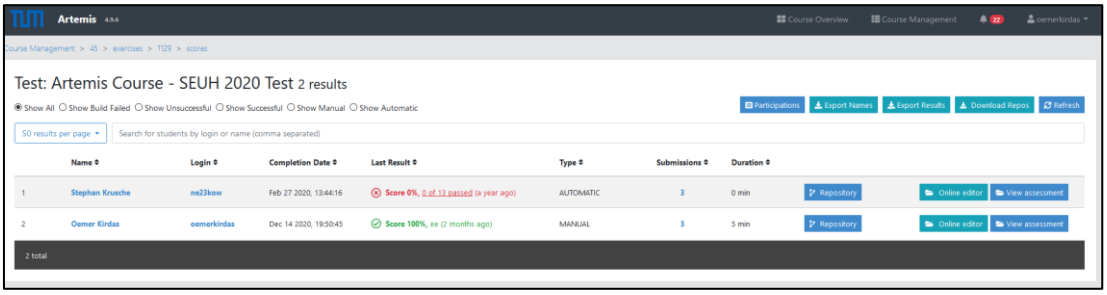
A.6 Abgaben ansehen



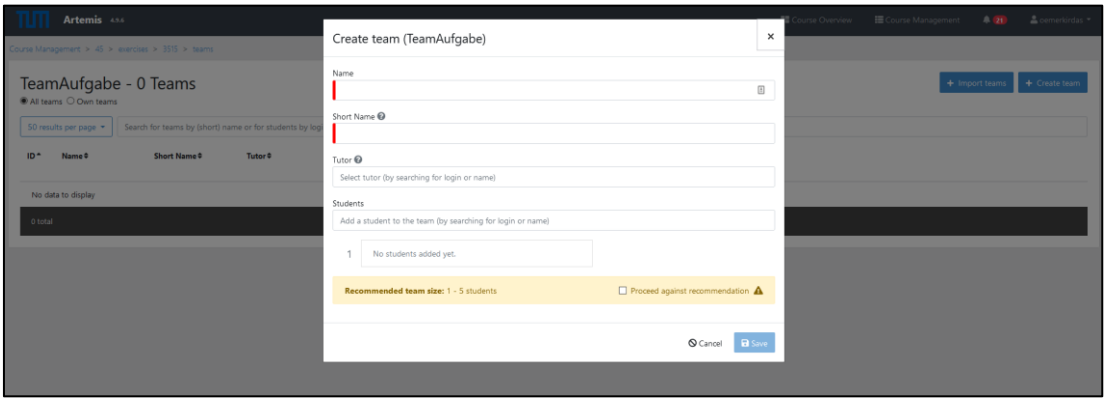
A.7 Online-Editor



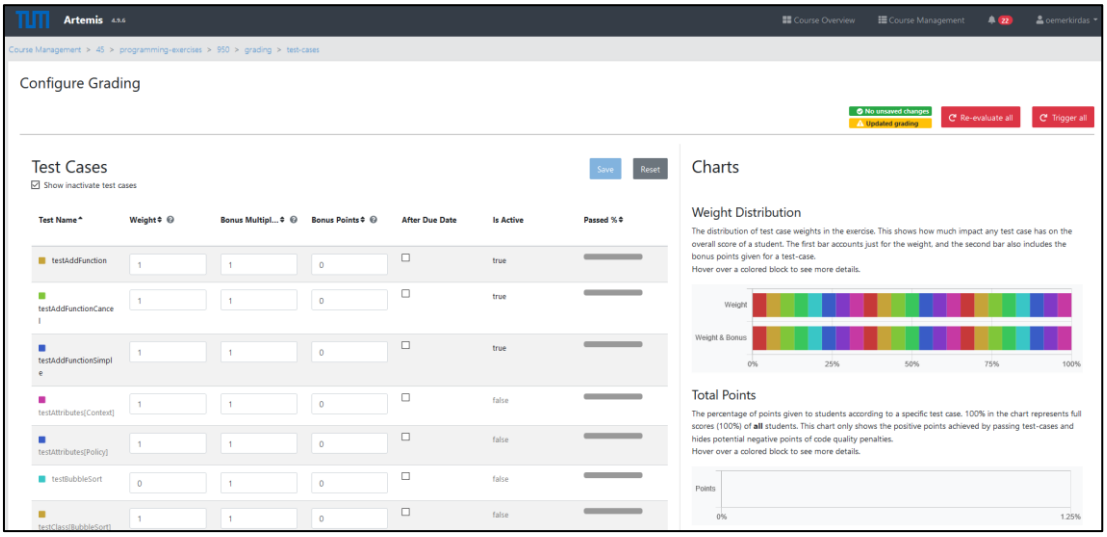
A.8 Ergebnisübersicht



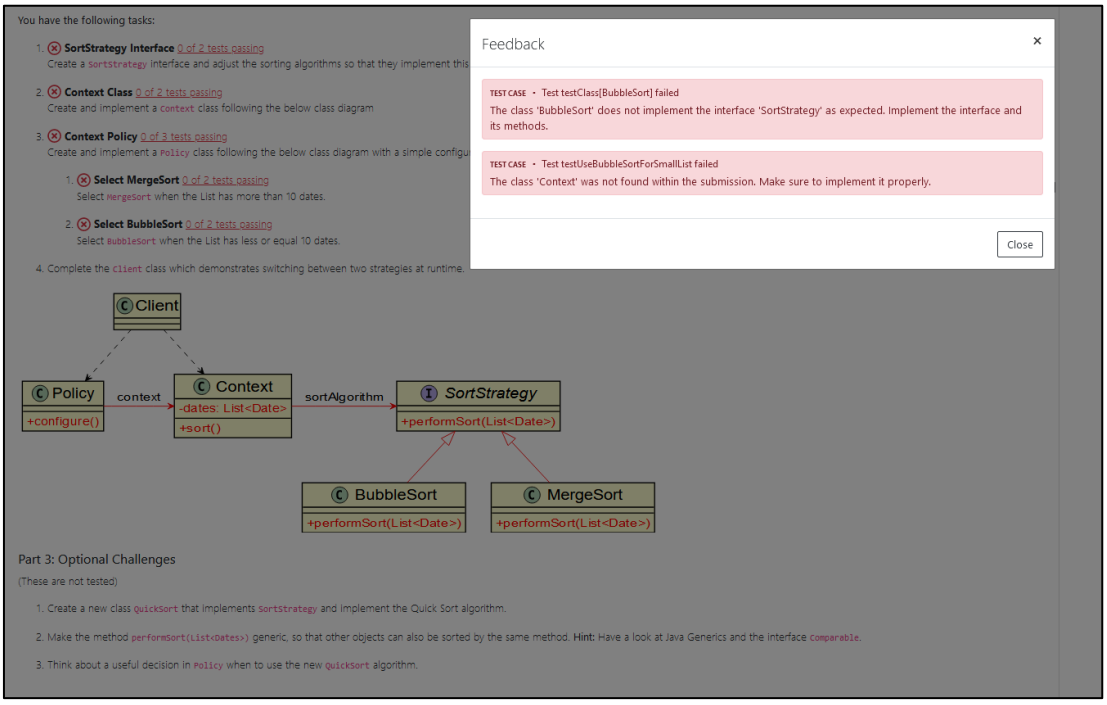
A.9 Teams erstellen



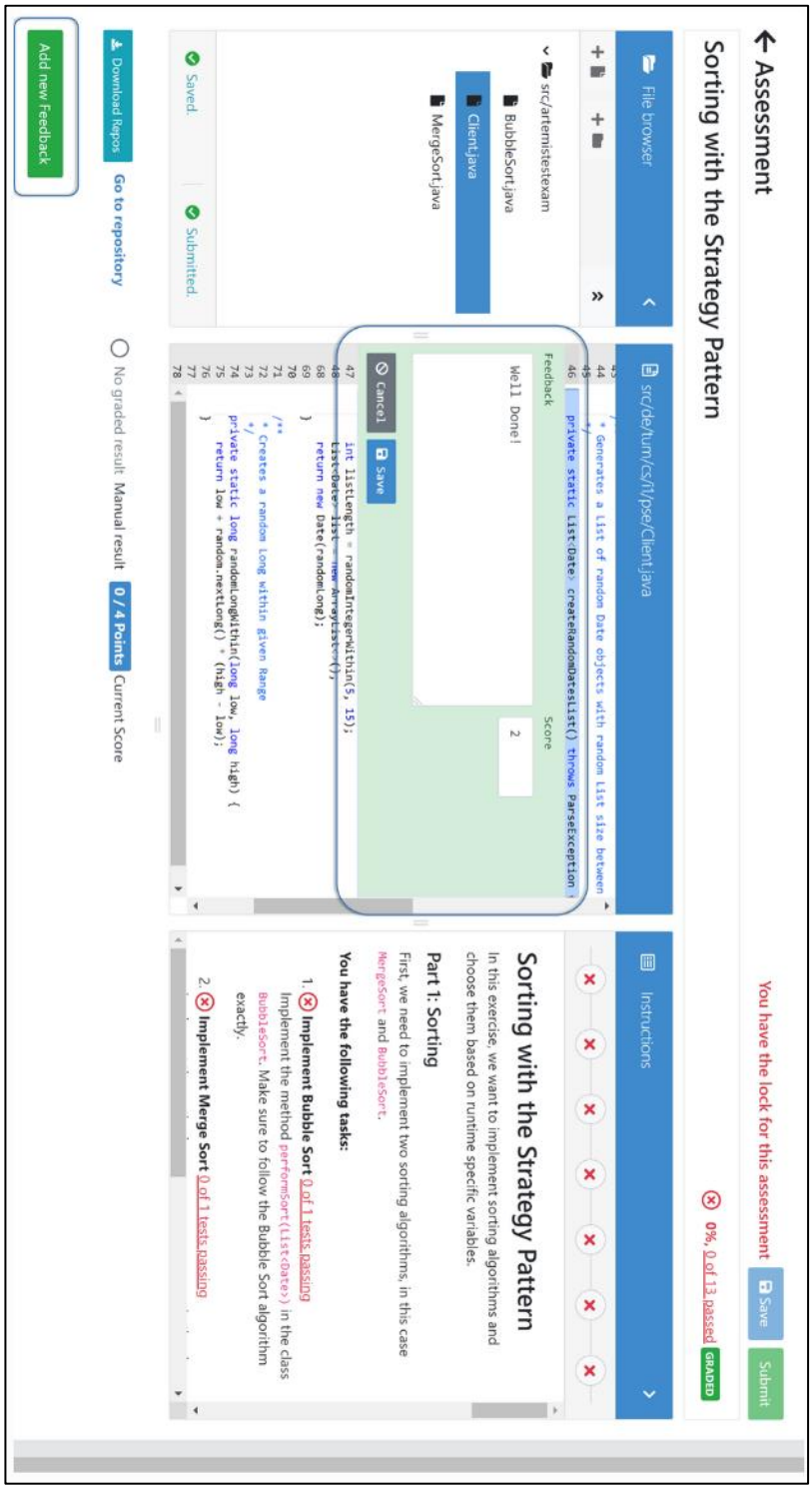
A.10Testfälle gewichten



A.11Feedback

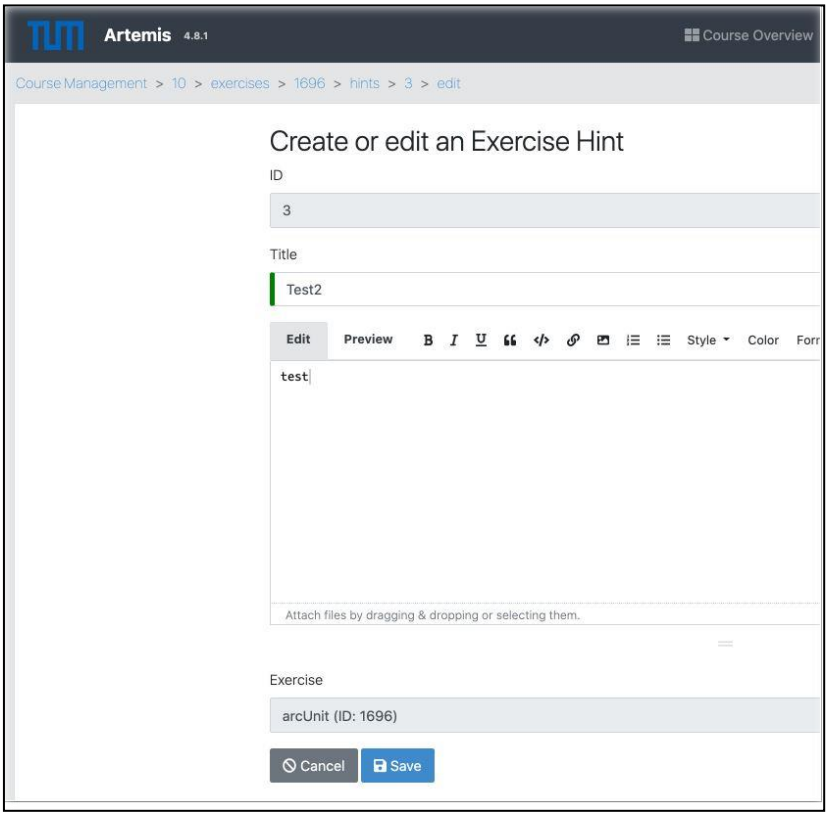


A.12Manuelles Feedback vergeben

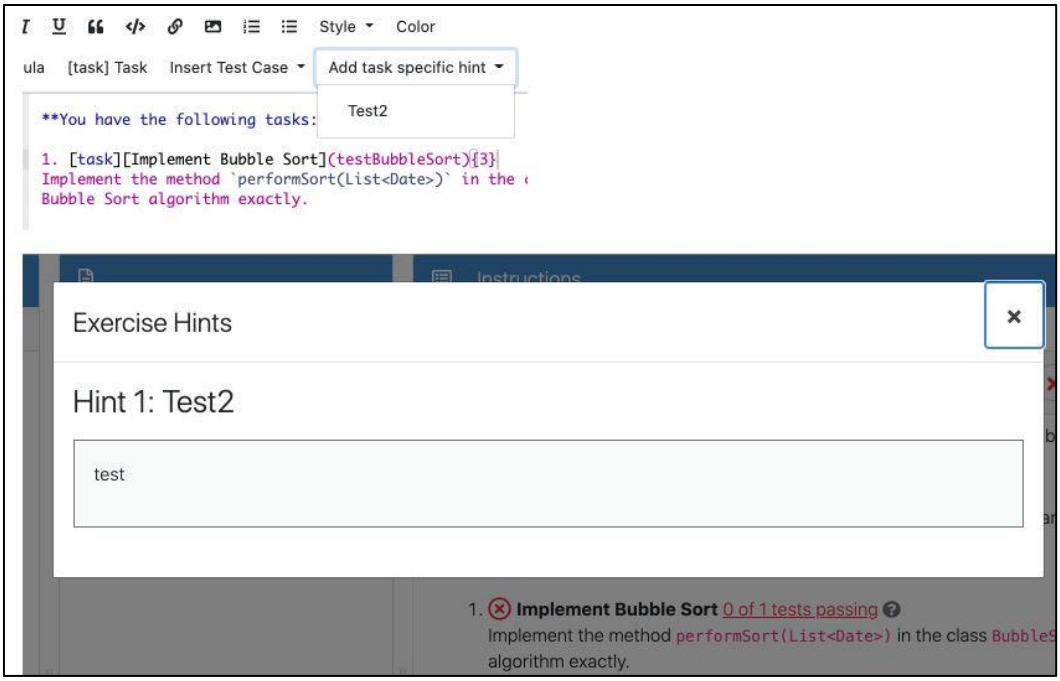


A.13Hinweise

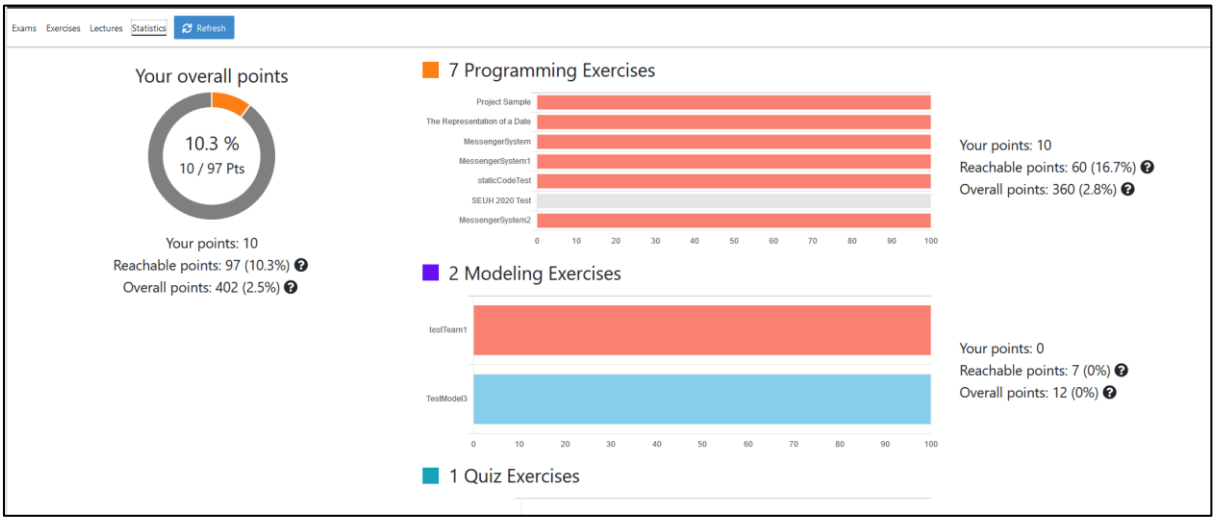
A.13.1 Hinweis erstellen



A.13.2 Hinweis abrufen

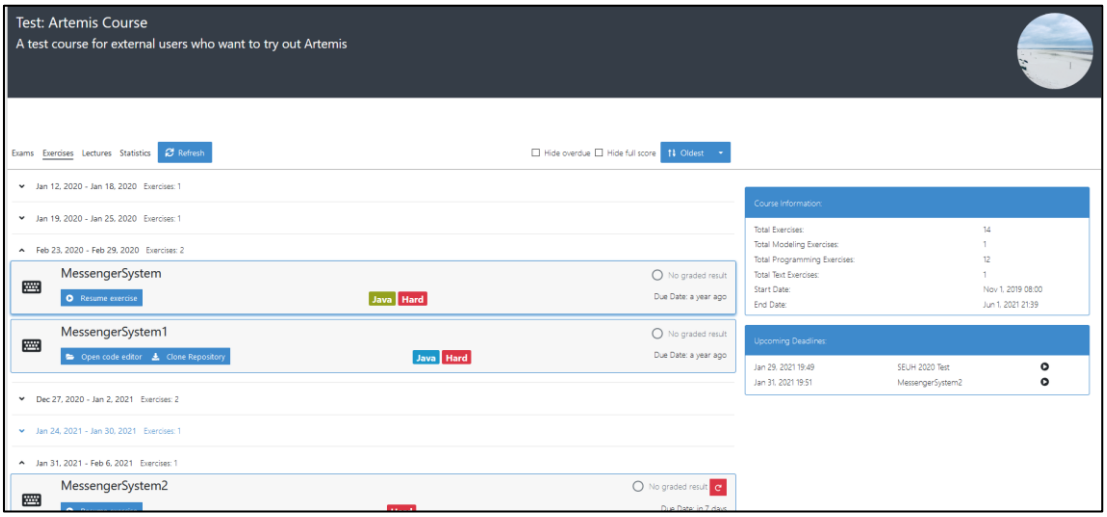


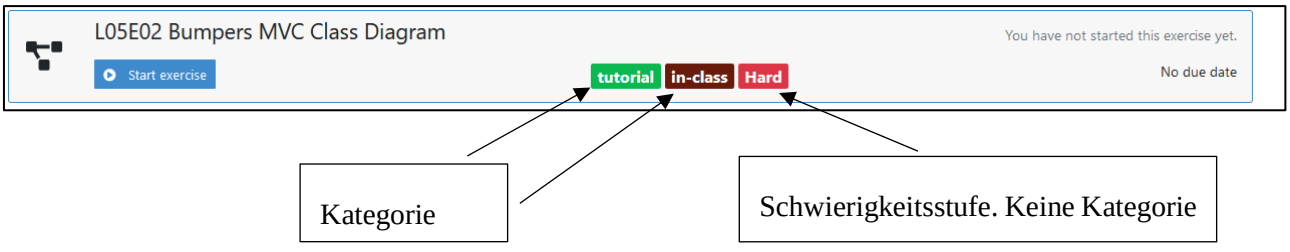
A.14Kursstatistik



A.15 Kursübersicht

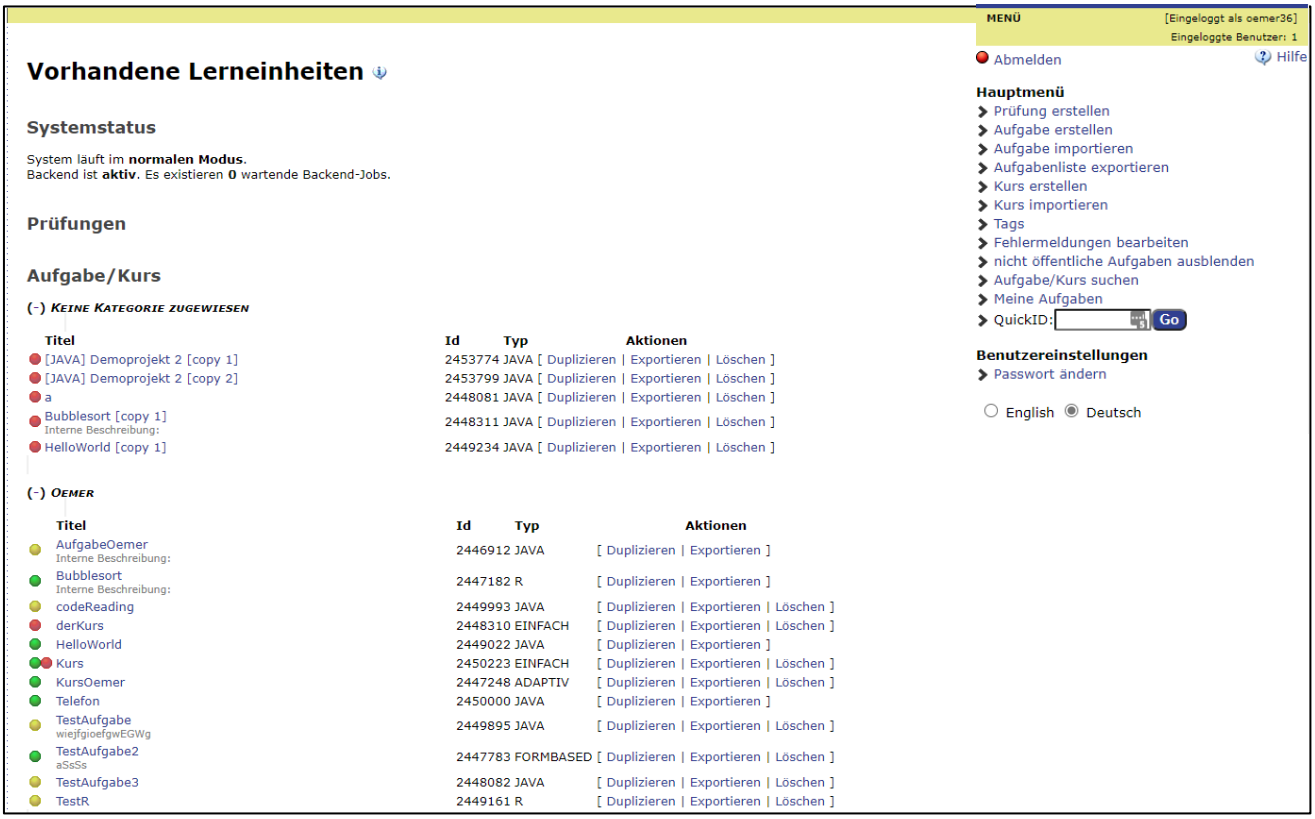
A.16Aufgabenübersicht



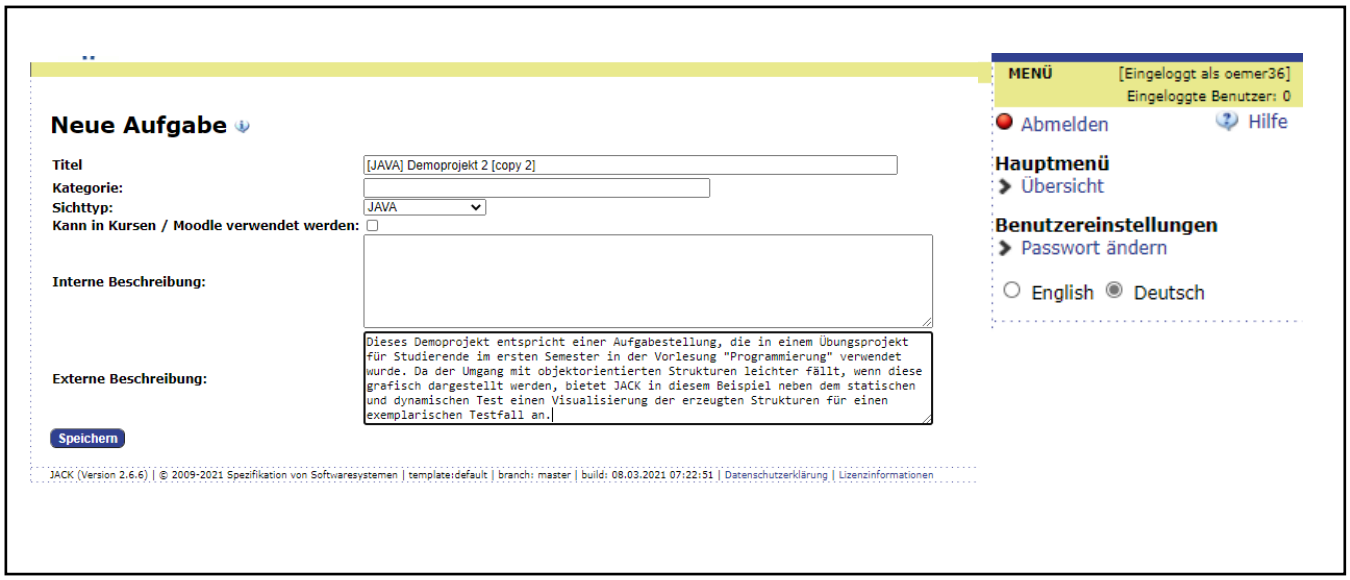


B Anhang: JACK Benutzeroberfläche

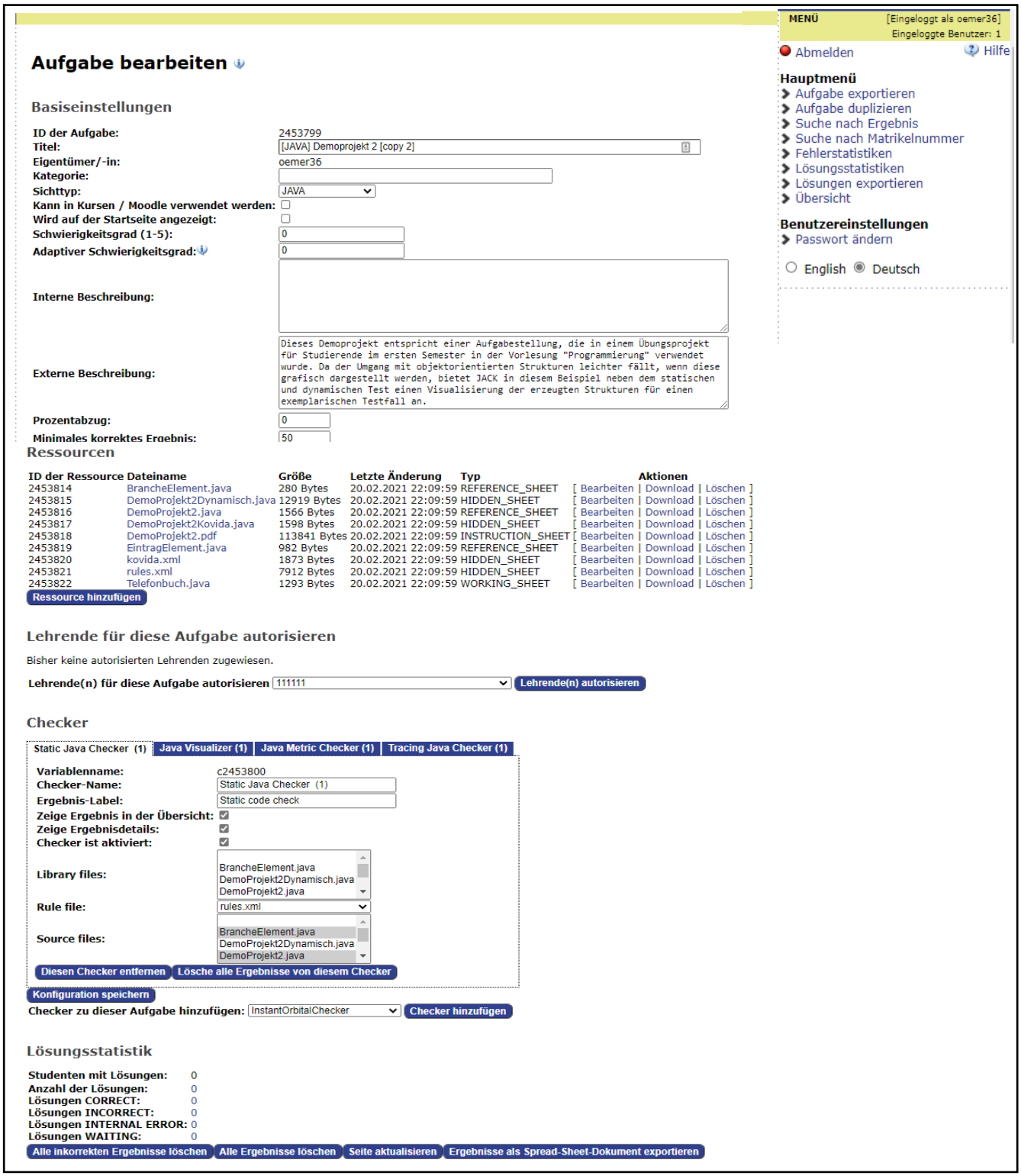
B.1 Hauptmenü – Lehrendensicht



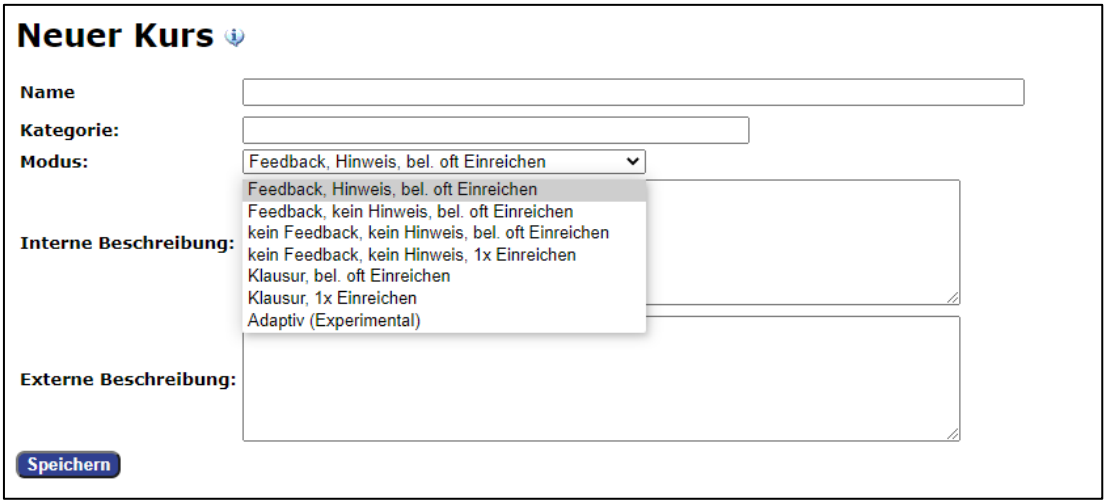
B.2 Aufgabe erstellen Teil 1



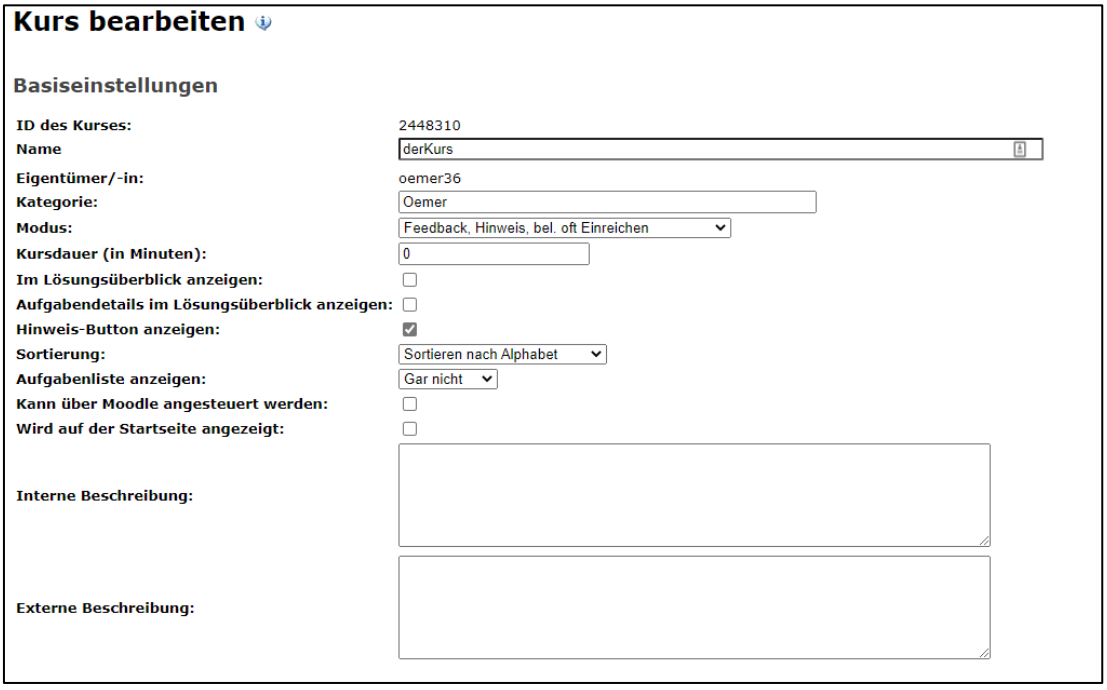
B.3 Aufgabe erstellen Teil 2: Aufgabenstellung bearbeiten



B.4 Kurs erstellen Teil 1

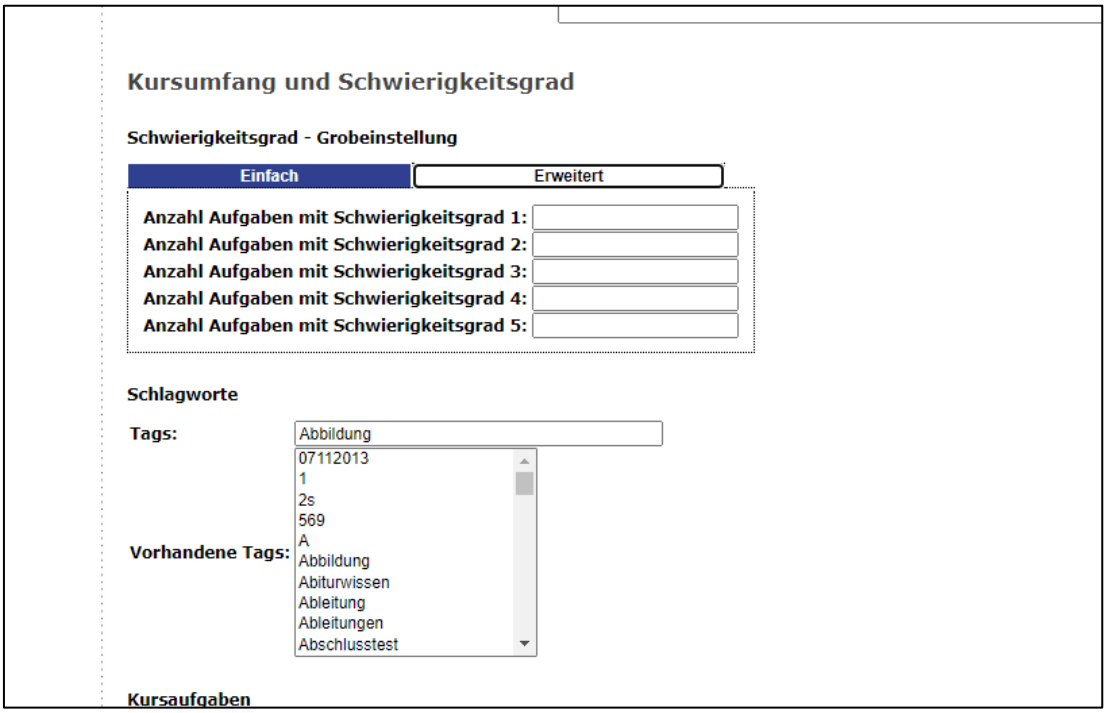


B.5 Kurs erstellen Teil 2: Kurs bearbeiten

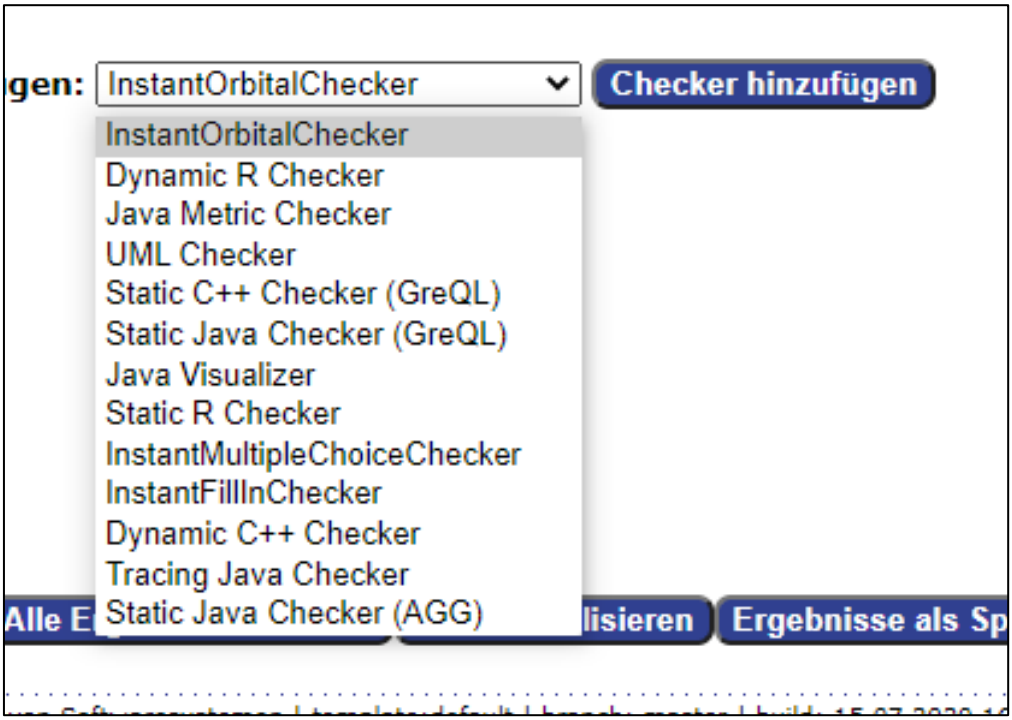



Kursumfang und Schwierigkeitsgrad
Schwierigkeitsgrad - Grobeinstellung
Einfach
Erweitert
Maximale Anzahl von Aufgaben:
0
Aufgaben haben mindestens Schwierigkeitsgrad:
0
Aufgaben haben höchstens Schwierigkeitsgrad:
0
Schlagworte
Tags:
Analysis
Vorhandene Tags:
07112013
1
2s
569
A
Abbildung
Abiturwissen
Ableitung
Ableitungen
Abschlusstest
Kursaufgaben
Zu Ihrer Konfiguration passen derzeit 19 Aufgaben. Diese sind die folgenden Aufgaben:
- Anzahl der Nullstellen
- Anzahl der Nullstellen 2
- euklidischer Algorithmus
16 mehr...
Einstellungen speichern
Lehrende für diesen Kurs autorisieren
Bisher keine autorisierten Lehrenden zugewiesen.
Lehrende(n) für diese Aufgabe autorisieren
111111
Lehrende(n) autorisieren
Ressourcen
Bisher keine Ressourcen hochgeladen.
Ressource hinzufügen

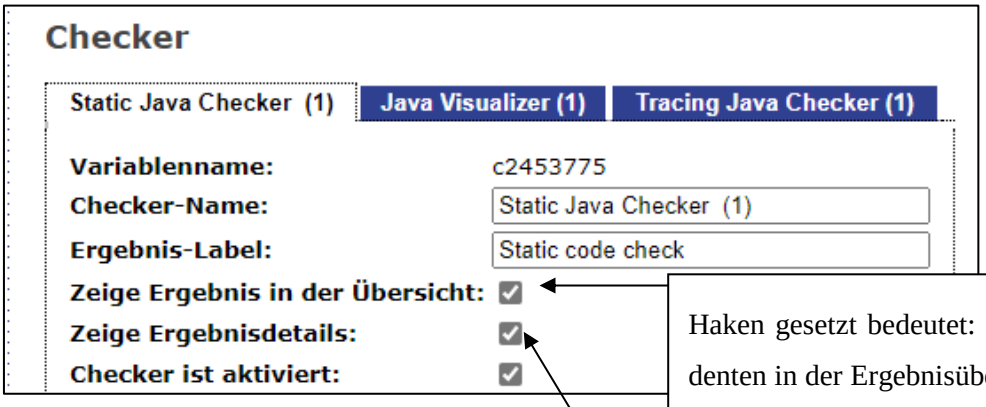
Erweiterte Grobeinstellung



B.6 Checker – Aufgabenentwicklersicht



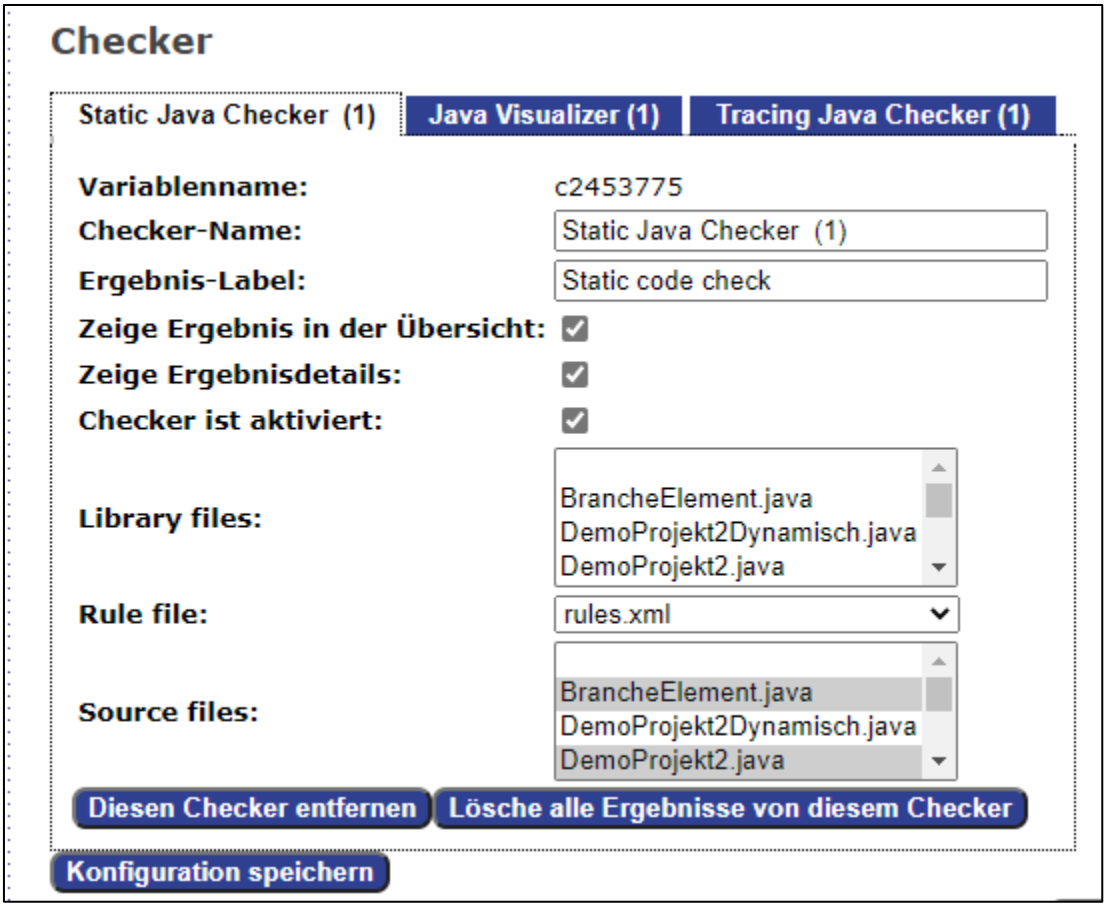
Standardkonfigurationen



Haken gesetzt bedeutet: Punkte werden dem Studenten in der Ergebnisübersicht angezeigt, die von dem Checker ausgehen

Haken gesetzt bedeutet: Feedback wird dem Studenten in der Ergebnisübersicht angezeigt, die von dem Checker ausgeht

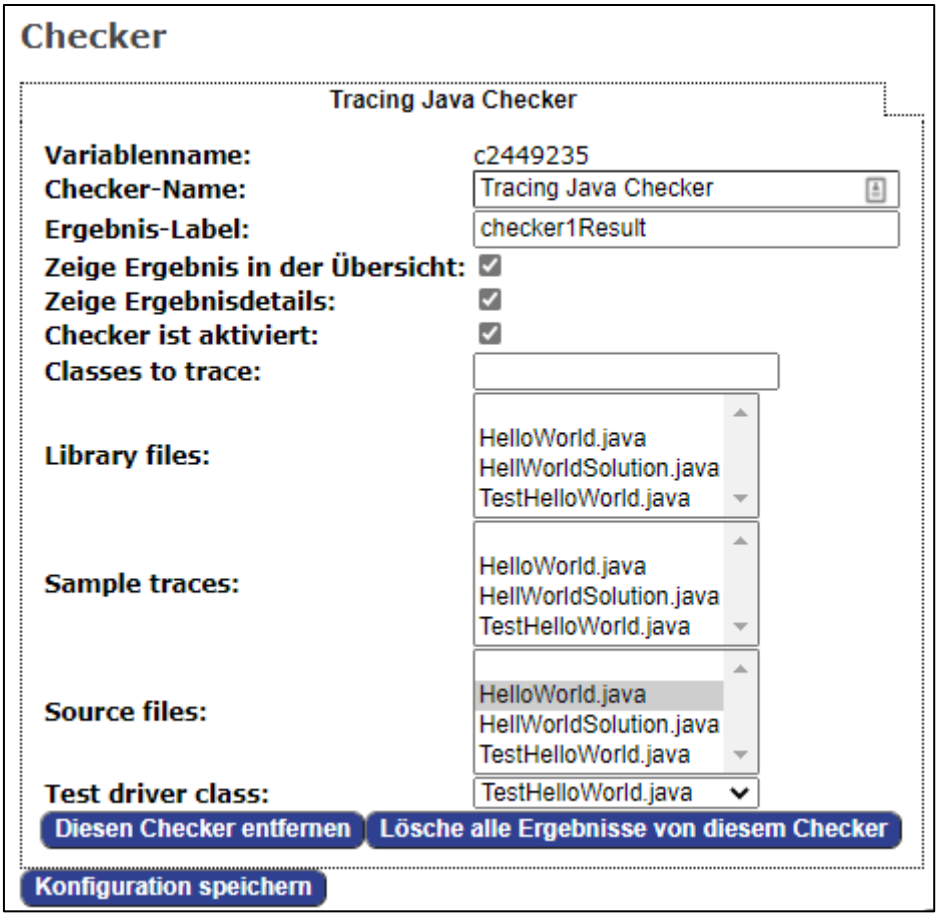
B.6.1 Static Java Checker



Rules.xml

```
<checkerrules>
    <!-- #### Stil-Regeln #### -->
    <rule type="absence" points="0">
        <query>from x : V{CompilationUnit}, y : V{PackageDeclaration}
                with x --&gt; y report x.name, y.name end</query>
        <feedback prefix="Hinweis (ohne Punktabzug)">Du verwendest eine Package-Deklaration,
anstatt die Datei im "default package" ohne Deklaration abzulegen.</feedback>
    </rule>
    <rule type="absence" points="0">
        <query>from x : V{VariableDeclarationFragment} with x.name=capitalizeFirst(x.name)
report x.name as "name" end</query>
        <feedback prefix="Hinweis (ohne Punktabzug)">Du verwendest Variablennamen, die mit
einem Großbuchstaben beginnen. Das ist möglich, aber es entspricht nicht dem üblichen
Programmierstil für Java.</feedback>
    </rule>
    undefined
</checkerrules>
```

B.6.2 Java Tracing Checker



Testklasse

```java
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import de.uni_due.s3.jack2.backend.checkers.tracingchecker.framework.TracingFramework;
import de.uni_due.s3.jack2.backend.checkers.tracingchecker.framework.TracingFramework.Test;

public class TestHelloWorld{

    private final ByteArrayOutputStream outContent = new ByteArrayOutputStream();
    private int punkte = 0;

    public int getResult() {

        return punkte;

    }

    @Test(name="Test 1")
    public void testHelloWorld(){
        System.setOut(new PrintStream(outContent));
        HelloWorld helloWorld = new HelloWorld();
        helloWorld.main(null);

        if (outContent.toString().equals("HelloWorld")) {
          punkte += 50;
        }
        else{
            TracingFramework.printError("HelloWorld does not sort correctly");
        }
    }
}
```

Tracing ist hier aktiviert, weil es nicht explizit abgeschaltet wurde. Würde hier `TracingFramework.switchOffTracing()` stehen würde die Tracing-Tabelle leer

Ergebnis

Ausführungstrace für Testfall "Test 1"

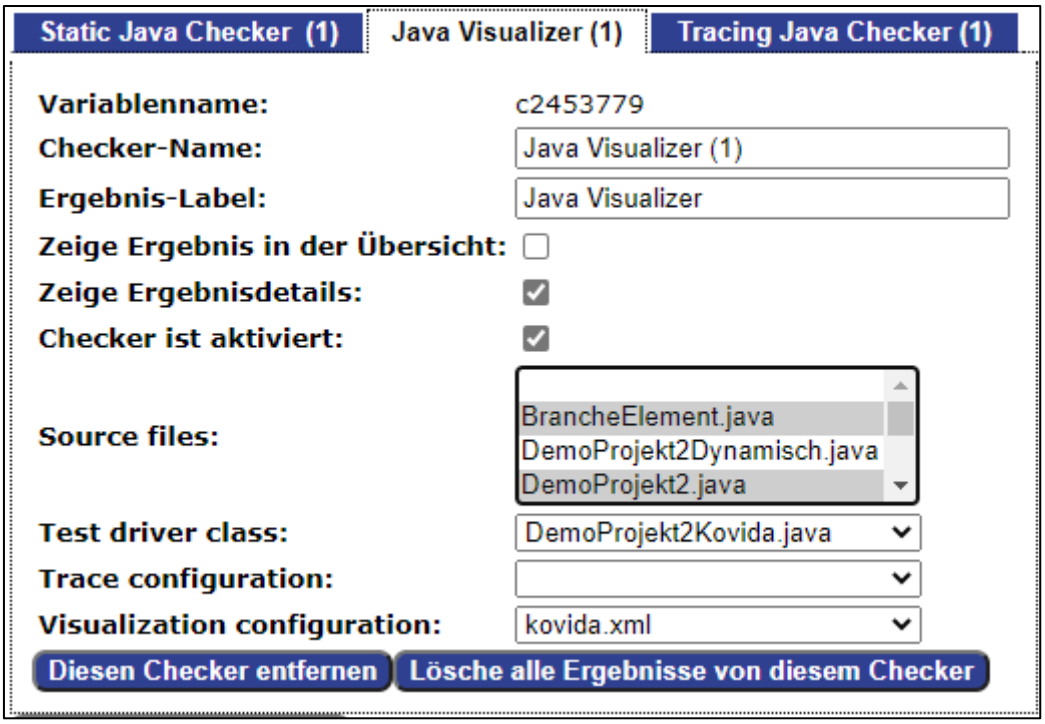
Dieser Trace gibt eine schrittweise Übersicht darüber, welche Programmzeilen ausgeführt wurden und welche Werte die beteiligten Variablen hatten.

Aufruf des Konstruktors der Klasse HelloWorld durch JACK
Aufruf der Methode 'main' durch JACK

Klasse und Zeile	Variablenwerte		Nächste ausgeführte Codezeile
	a	args	
HelloWorld:6		null	String a = "HelloWorld ";
HelloWorld:7	"HelloWorld "	null	System.out.print(a);
HelloWorld:8	"HelloWorld "	null	}

```
1 public class HelloWorld {
2
3     // TODO Change the text "Hello Jack" to "HelloWorld"
4
5     public static void main(String[] args) {
6         String a = "HelloWorld  ";
7         System.out.print(a);
8     }
9 }
```

B.6.3 Java Visualizer



Kovida.xml

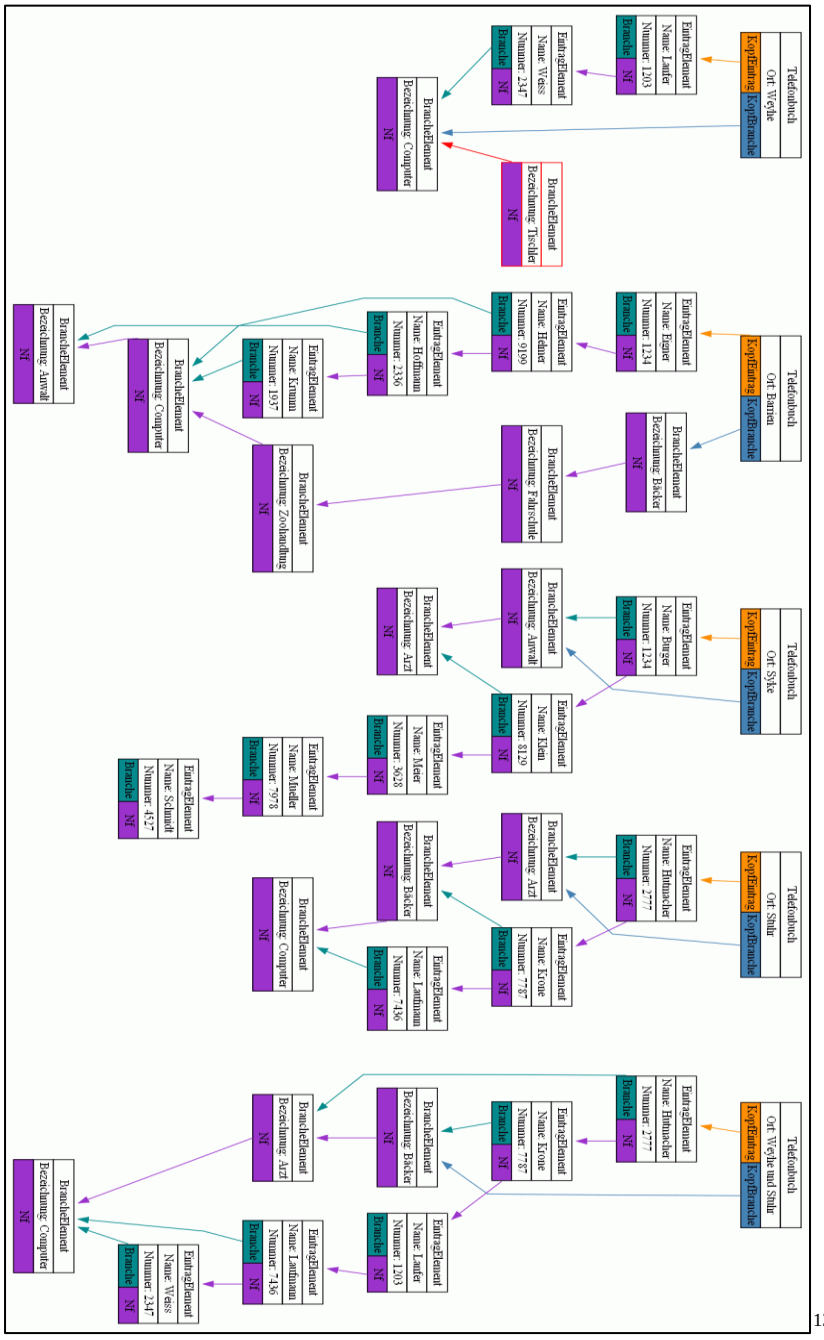
```
<project>
   <classes>
      <class name="BrancheElement" style="list_element" />
      <class name="EintragElement" style="list_element" />
      <class name="Telefonbuch" style="list" />
   </classes>
   <breakpoints name="DemoProjekt2Kovida.java">
      <breakpoint orderId="1">29</breakpoint>
      <breakpoint orderId="2">38</breakpoint>
      <breakpoint orderId="3">44</breakpoint>
      <breakpoint orderId="4">50</breakpoint>
      <breakpoint orderId="5">54</breakpoint>
   </breakpoints>
   <info />
</project>
```

Testtreiber: DemoProjekt2Kovida.java

rules.xml | kovida.xml | DemoProjekt2Kovida.java

```
1  public class DemoProjekt2Kovida {
2
3      private Telefonbuch t1;
4      private Telefonbuch t2;
5      private Telefonbuch t3;
6      private Telefonbuch t4;
7      private Telefonbuch t5;
8
9      private boolean gleicheNummern1;
10     private boolean gleicheNummern2;
11
12     public static void main (String[] args) {
13         DemoProjekt2Kovida kovida = new DemoProjekt2Kovida();
14         kovida.start();
15     }
16
17     public void start() {
18         DemoProjekt2 demo = new DemoProjekt2();
19         demo.Landkreis = new Telefonbuch[4];
20
21         t1 = demo.Landkreis[0] = new Telefonbuch("Syke");
22         t1.neu("Mueller",   7978);
23         t1.neu("Meier",     3628);
24         t1.neu("Schmidt",   4527);
25         t1.neu("Klein",     8129,  "Arzt");
26         t1.neu("Burger",    1234,  "Anwalt");
27
28         t2 = demo.Landkreis[1] = new Telefonbuch("Barrien");
29         t2.neu("Hoffmann",  2336,  "Anwalt");
30         t2.neu("Krumm",     1937,  "Computer");
31         t2.neu("Eigner",    1234);
32         t2.neu("Helmer",    9199,  "Computer");
33         t2.neu("Zoohandlung");
34         t2.neu("Fahrschule");
35         t2.neu("Bäcker");
36
37         t3 = demo.Landkreis[2] = new Telefonbuch("Weyhe");
38         t3.neu("Weiss",     2347,  "Computer");
39         t3.neu("Lanfer",    1203);
40         t3.neu("Tischler");
41         t3.reduziereBranchen();
42
43         t4 = demo.Landkreis[3] = new Telefonbuch("Stuhr");
44         t4.neu("Hutmacher", 2777,  "Arzt");
45         t4.neu("Krone",     7787,  "Bäcker");
46         t4.neu("Laufmann",  7436,  "Computer");
47         t4.sortiereBranchen();
48
49         gleicheNummern1 = demo.Landkreis[0].gleicheNummern(demo.Landkreis[1]);
50         gleicheNummern2 = demo.Landkreis[0].gleicheNummern(demo.Landkreis[3]);
51
52         t5 = demo.Landkreis[2].vereinige(demo.Landkreis[3]);
53     }
54 }
55
```

Ergebnis

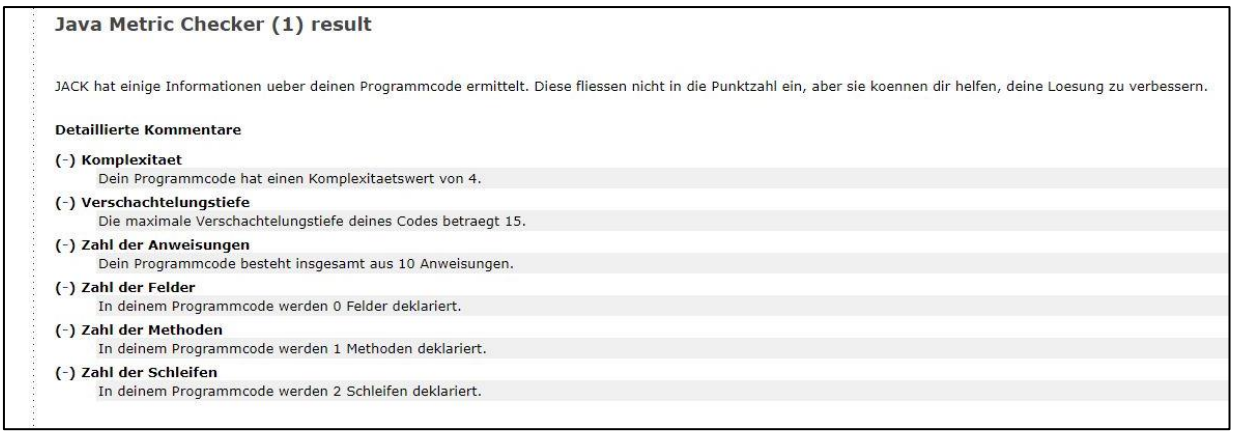


[13]

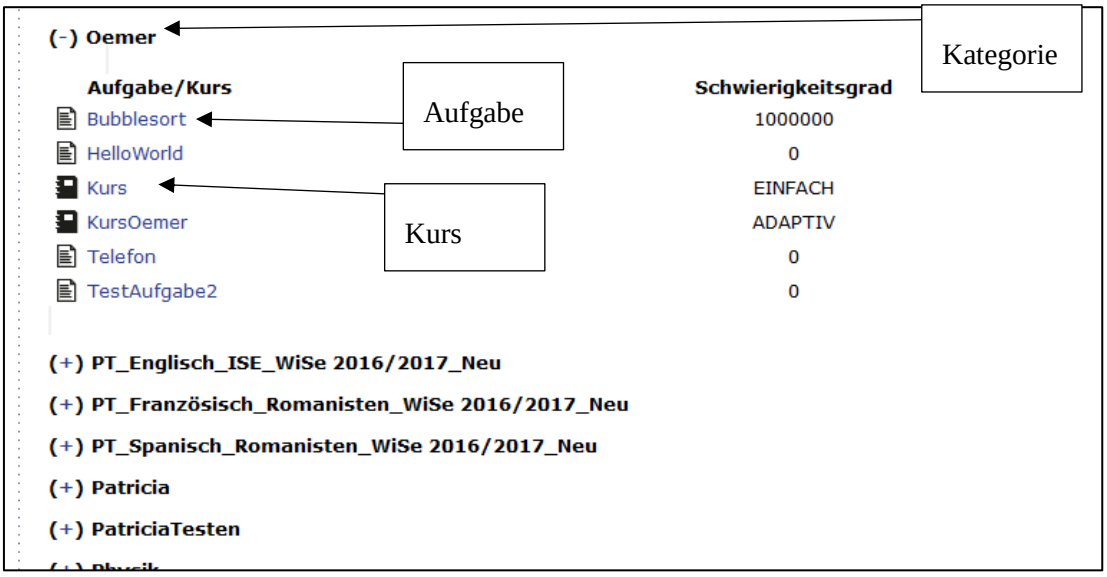
[13] https://jack-community.org/wiki/index.php/KovidaChecker

B.6.4 Java Metric Checker

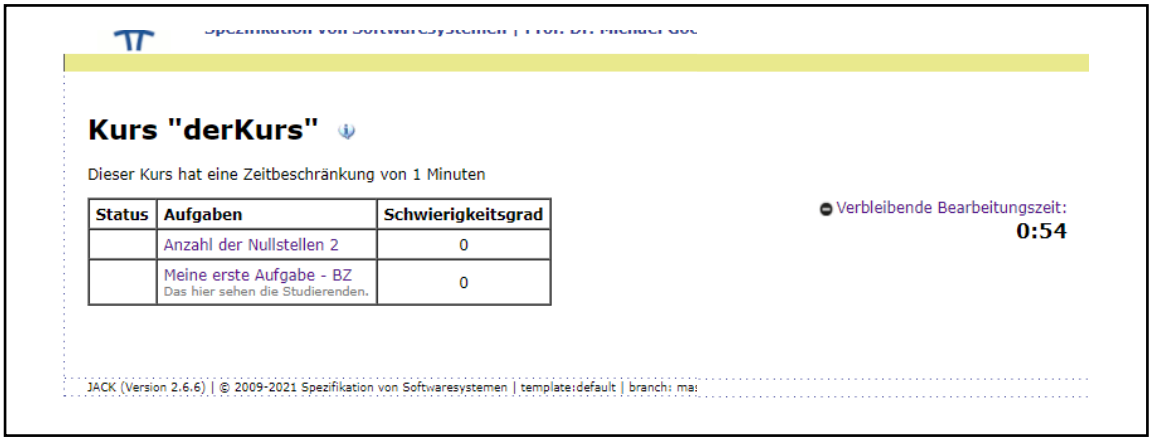
Ergebnis



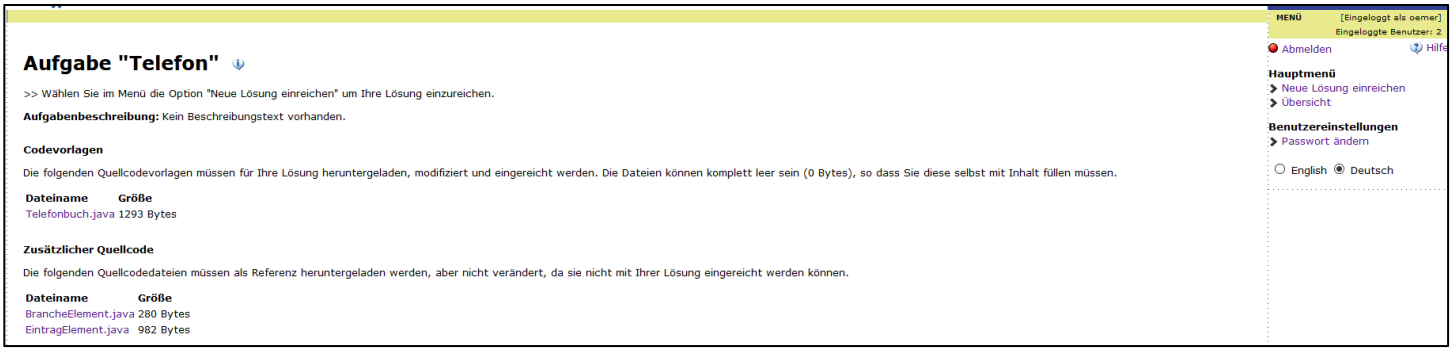
B.7 Aufgabenübersicht - Studentensicht



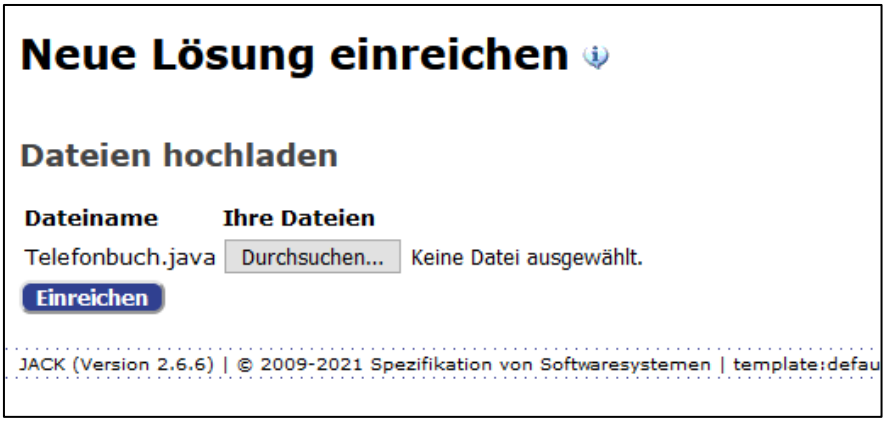
B.7.1 Kursübersicht



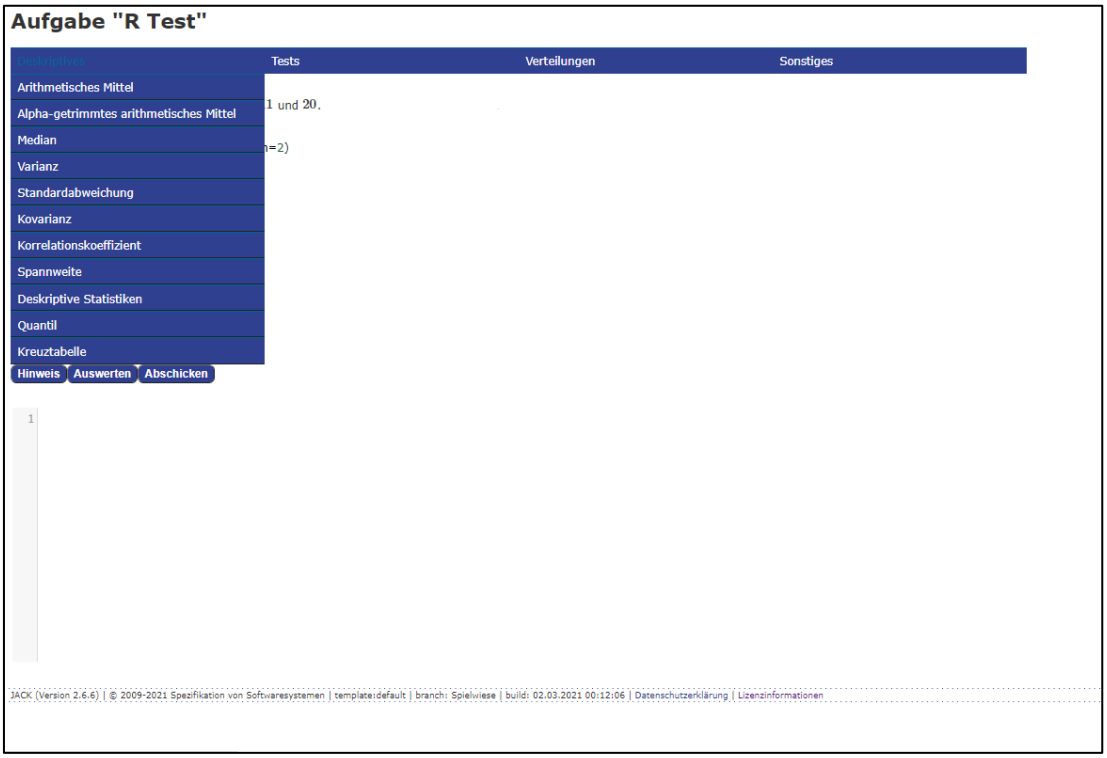
B.7.2 Aufgabe



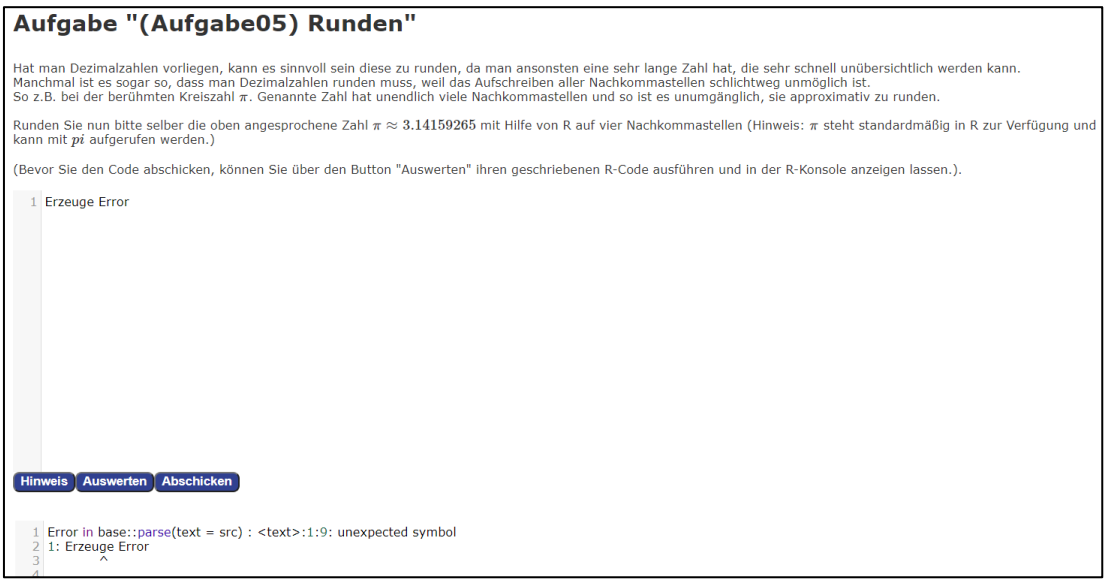
B.7.3 Aufgabe einreichen



B.7.4 Funktionsmenü bei R-Aufgaben



B.7.5 Editor für R-Aufgaben



B.8 Lösungsüberblick - Lehrendensicht

B.8.1 Kurs Lösungsstastik

Lösungsstatistiken

Nutzungsstatistik

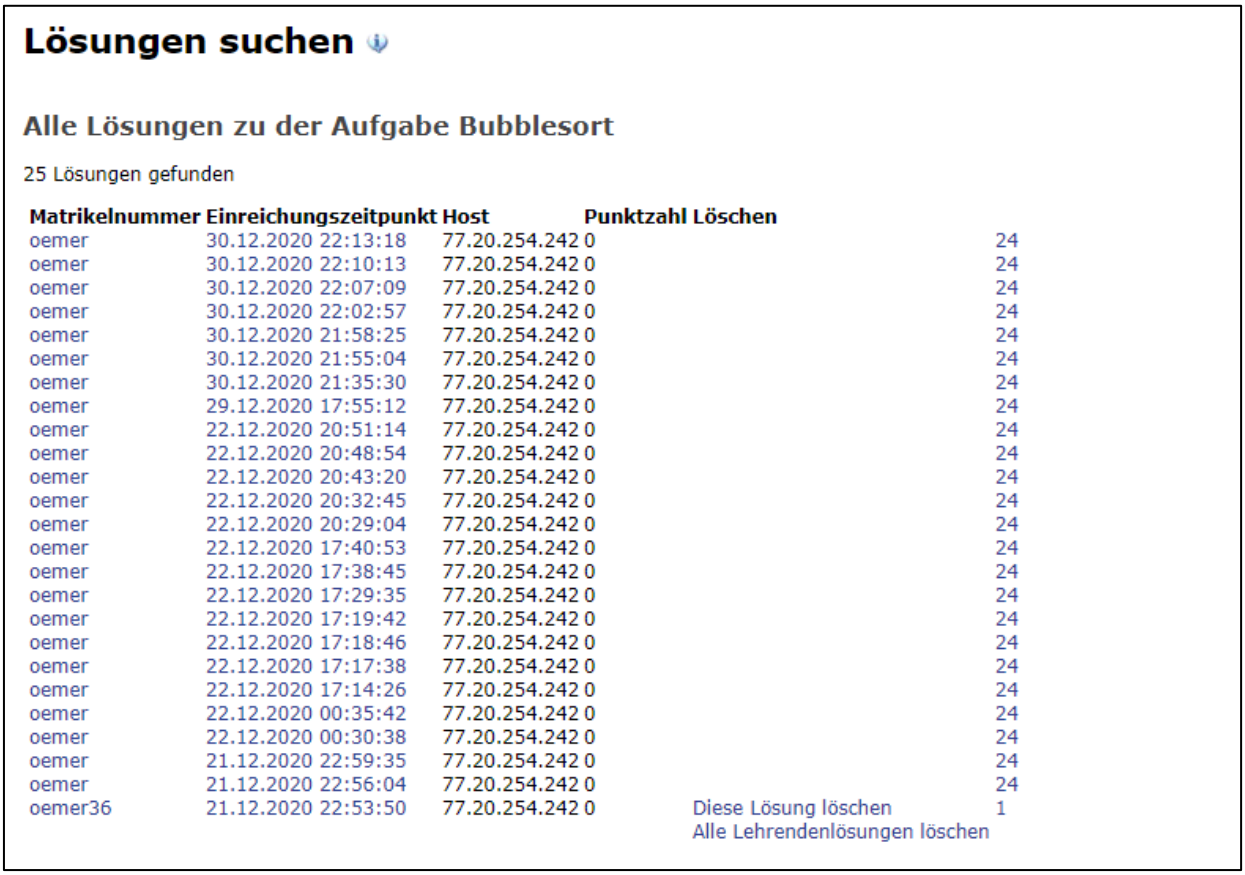
Studenten mit Bearbeitungen:	1
Anzahl der Bearbeitungen:	9
Anzahl der Bearbeitungen ohne Lösungen:	8
Anzahl der Lösungen:	2
Lösungen CORRECT:	0
Lösungen INCORRECT:	1
Lösungen INTERNAL ERROR:	0
Lösungen WAITING:	1
Höchster erreichter Durchschnitt:	0.0
Niedrigster erreichter Durchschnitt:	0.0
Arithmetisches Mittel aller erreichten Durchschnitte:	0.0
Größte # an Lösungen:	9
Kleinste # an Lösungen:	9

Seite aktualisieren | Ergebnisse als Spread-Sheet-Dokument exportieren | Ranking | Kursdurchschnitte neu berechnen | Leere Kurse löschen

Aufgabe	Lösungen	richtige Lösungen	falsche Lösungen
Nullstellen genügen Gleichung 2	1	0	1
Meine erste Aufgabe - BZ	1	0	1
Anzahl der Nullstellen	0	0	0
Anzahl der Nullstellen 2	0	0	0
euklidischer Algorithmus	0	0	0
komplexe Zahlen	0	0	0
Konvergenz von Folgen	0	0	0
Konvergenz von Folgen - MC	0	0	0
Konvergenz von Folgen- Wahr/Falsch	0	0	0
L'Hospital	0	0	0
Nullstellen genügen Gleichung 1	0	0	0
Nullstellen und Asymptoten bestimmen	0	0	0
Polarkoodinaten	0	0	0
quadratische Funktion	0	0	0
quadratische Funktion-Schrittweise	0	0	0
Stammfunktion gesucht	0	0	0
Stammfunktion gesucht2	0	0	0
Ungleichung lösen	0	0	0
Ungleichung lösen-schrittweise	0	0	0
Summe	2	0	2

B.8.2 Einreichungsstatistik einer Aufgabe – alle Teilnehmer

B.8.3 Einreichungssuche



Lösungen suchen

Alle Lösungen zu der Aufgabe Bubblesort

25 Lösungen gefunden

Matrikelnummer	Einreichungszeitpunkt	Host	Punktzahl	Löschen	
oemer	30.12.2020 22:13:18	77.20.254.242	0		24
oemer	30.12.2020 22:10:13	77.20.254.242	0		24
oemer	30.12.2020 22:07:09	77.20.254.242	0		24
oemer	30.12.2020 22:02:57	77.20.254.242	0		24
oemer	30.12.2020 21:58:25	77.20.254.242	0		24
oemer	30.12.2020 21:55:04	77.20.254.242	0		24
oemer	30.12.2020 21:35:30	77.20.254.242	0		24
oemer	29.12.2020 17:55:12	77.20.254.242	0		24
oemer	22.12.2020 20:51:14	77.20.254.242	0		24
oemer	22.12.2020 20:48:54	77.20.254.242	0		24
oemer	22.12.2020 20:43:20	77.20.254.242	0		24
oemer	22.12.2020 20:32:45	77.20.254.242	0		24
oemer	22.12.2020 20:29:04	77.20.254.242	0		24
oemer	22.12.2020 17:40:53	77.20.254.242	0		24
oemer	22.12.2020 17:38:45	77.20.254.242	0		24
oemer	22.12.2020 17:29:35	77.20.254.242	0		24
oemer	22.12.2020 17:19:42	77.20.254.242	0		24
oemer	22.12.2020 17:18:46	77.20.254.242	0		24
oemer	22.12.2020 17:17:38	77.20.254.242	0		24
oemer	22.12.2020 17:14:26	77.20.254.242	0		24
oemer	22.12.2020 00:35:42	77.20.254.242	0		24
oemer	22.12.2020 00:30:38	77.20.254.242	0		24
oemer	21.12.2020 22:59:35	77.20.254.242	0		24
oemer	21.12.2020 22:56:04	77.20.254.242	0		24
oemer36	21.12.2020 22:53:50	77.20.254.242	0	Diese Lösung löschen Alle Lehrendenlösungen löschen	1

B.8.4 Lösungsdetails für eine Java-Aufgabe

Lösungsdetails

Allgemeine Informationen

Matrikelnummer: owner
Host-Adresse: 77.20.254.122
Aufgabentitel: HelloWorld
Einreichung: 5.03.2021 15:00:25
Öffentliche Aufgabenbeschreibung: Printe HelloWorld

Diese Lösung löschen

Ergebnisübersicht

Gesamtergebnis: noch nicht komplett bearbeitet
Java Visualizer (1) result: 100 [Ergebnis löschen]
Tracechecker: 50 [Ergebnis löschen]
Static Java Checker (AGC) (1) result: Job wartet

Seite aktualisieren | Alle Ergebnisse löschen

Lösungsressourcen

Dateiname	Größe	Hash-Code	Aktionen
coverageMap	0 Bytes	0	Löschen
HelloWorld.java	274 Bytes	1998088810	

Alle Ressourcen herunterladen | Lösung für Workspace exportieren

Weitere Attribute

Checker-Name	Schlüssel	Wert
Java Visualizer (1)	Executor for checker config 2449686	backend01-A
Tracechecker	Executor for checker config 2449028	backend08
Tracechecker	test coverage	100.0% (5 of 5 lines)
Tracechecker	tracing checker error processing time	0 msec.
Tracechecker	tracing checker run time	2051 msec.

Manuelles Ergebnis

Interne Beschreibung zum manuellen Ergebnis:

Punktzahl: 0 — Manuelles Ergebnis speichern

Fehlereintrag zum manuellen Ergebnis:

Position 1

Nachricht MANUELLS FEEDBACK

Sichtbarkeit

Nachricht hinzufügen

Java Visualizer (1) result

Tracechecker

Folgende Ausgabe wurde auf System.out.println geschrieben:

```
TEST HAT GESTARTET
```

Eingereichter Quellcode

HelloWorld.java (Download ,274 Bytes)

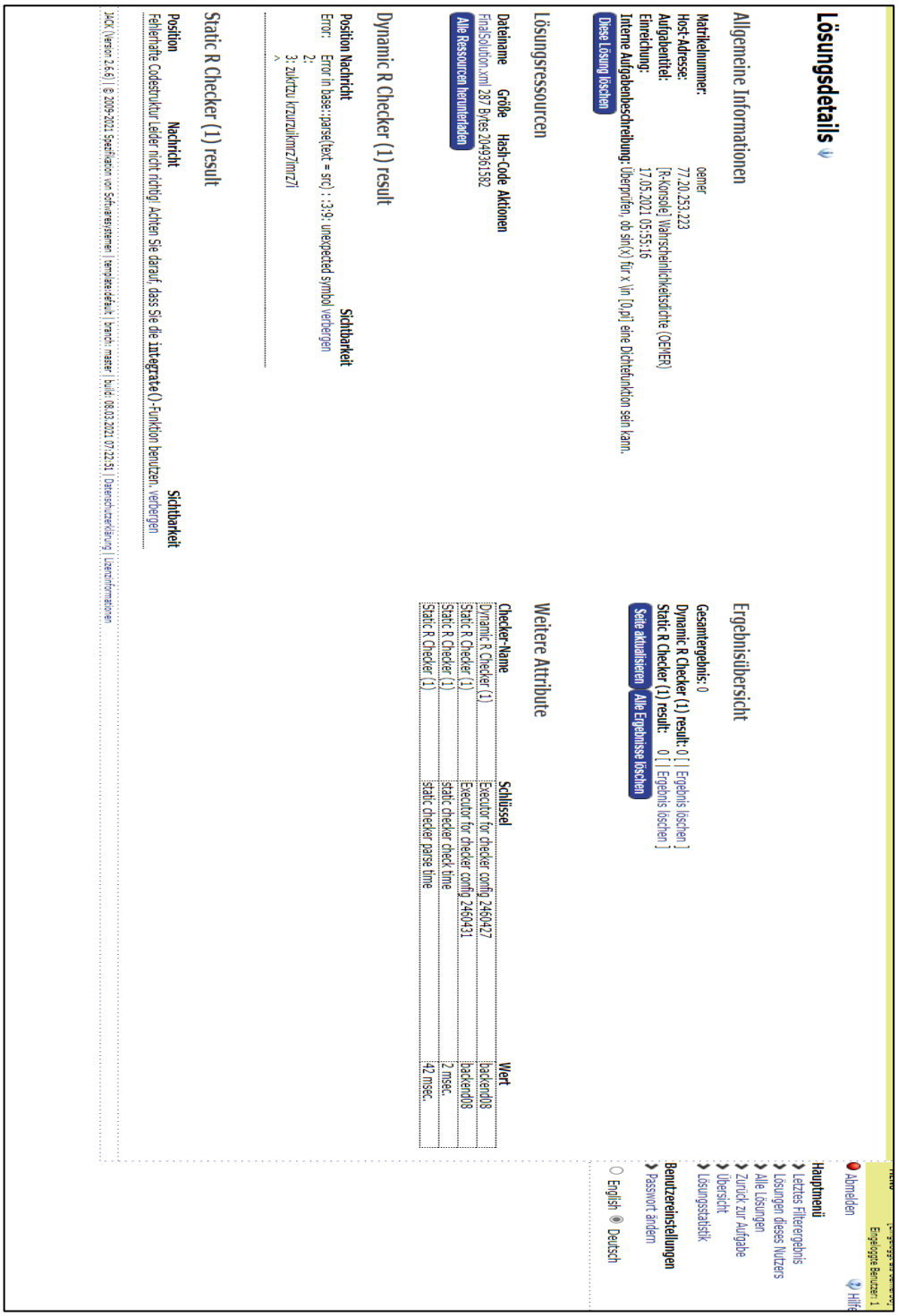
```
1 public class HelloWorld {
2
3     // TODO Change the text "Hallo Jack" to "Hallo World"
4     private String a = "HelloWorld";
5     public static void main(String[] args) {
6         HelloWorld helloWorld = new HelloWorld();
```

Hauptmenü
- Letztes Filterergebnis
- Lösungen dieses Nutzers
- Alle Lösungen
- Zurück zur Aufgabe
- Zurück zur Übersicht
- Lösungsstatistik

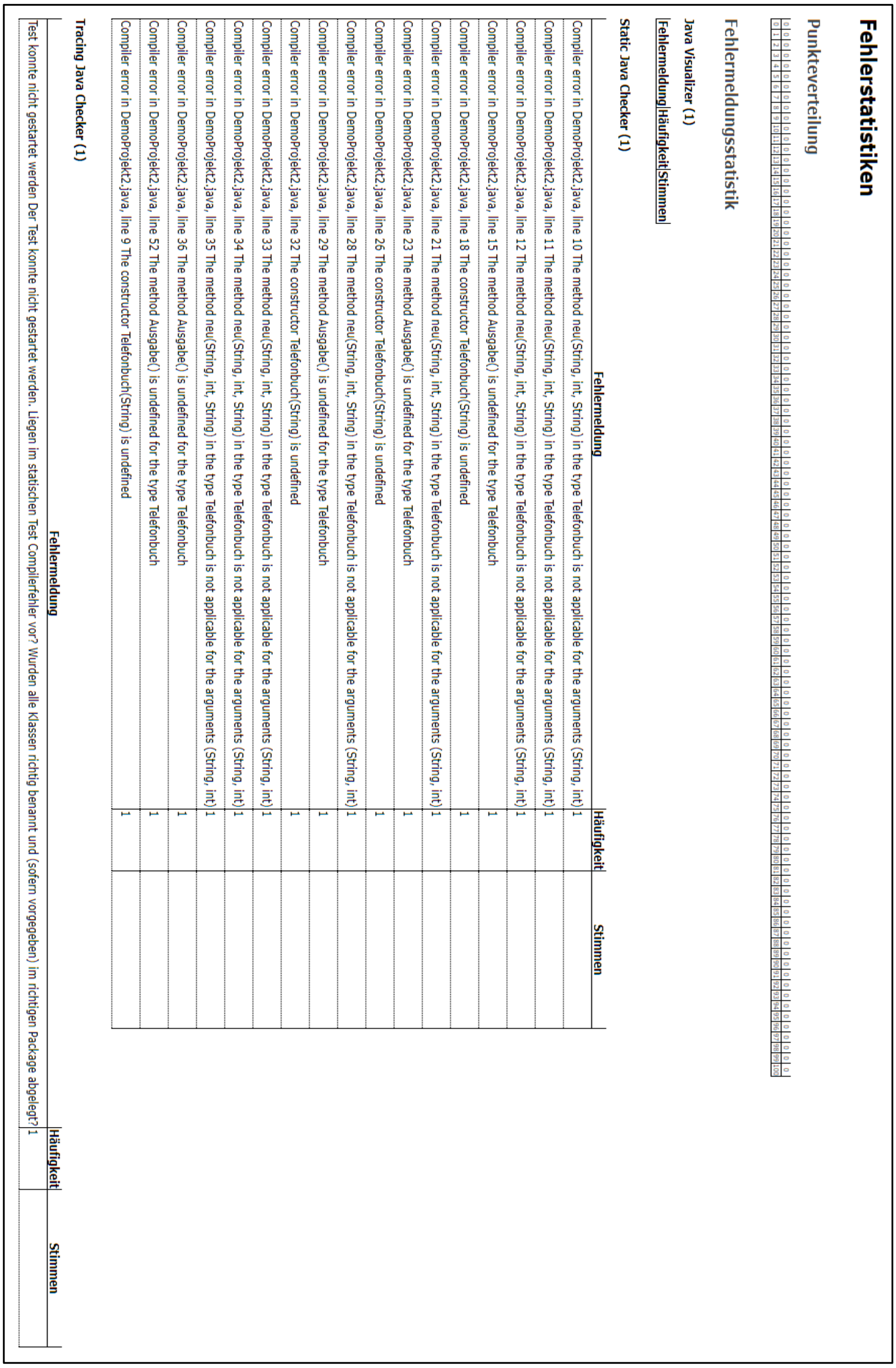
Benutzereinstellungen
- Passwort ändern

English / Deutsch

B.8.5 Lösungsdetails für eine R Aufgabe



B.8.6 Fehlerstatistik einer Aufgabe



Fehlerstatistiken

Punkteverteilung

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fehlermeldungsstatistik

Java Visualizer (1)

Fehlermeldung	Häufigkeit	Stimmen

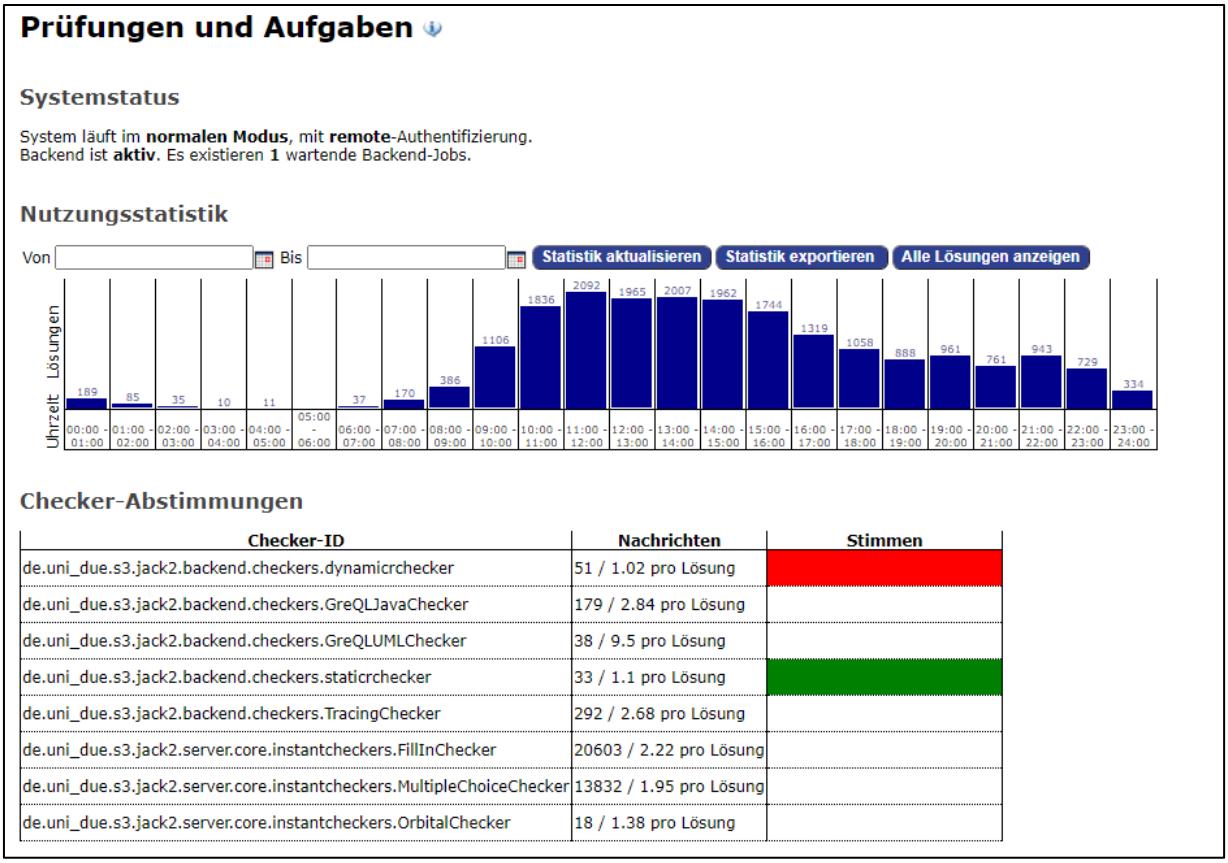
Static Java Checker (1)

Fehlermeldung	Häufigkeit	Stimmen
Compiler error in DemoProjekt2.java, line 10 The method neu(String, int, String) in the type Telefonbuch is not applicable for the arguments (String, int)	1	
Compiler error in DemoProjekt2.java, line 11 The method neu(String, int, String) in the type Telefonbuch is not applicable for the arguments (String, int)	1	
Compiler error in DemoProjekt2.java, line 12 The method neu(String, int, String) in the type Telefonbuch is not applicable for the arguments (String, int)	1	
Compiler error in DemoProjekt2.java, line 15 The method Ausgabe() is undefined for the type Telefonbuch	1	
Compiler error in DemoProjekt2.java, line 18 The constructor Telefonbuch(String) is undefined	1	
Compiler error in DemoProjekt2.java, line 21 The method neu(String, int, String) in the type Telefonbuch is not applicable for the arguments (String, int)	1	
Compiler error in DemoProjekt2.java, line 23 The method Ausgabe() is undefined for the type Telefonbuch	1	
Compiler error in DemoProjekt2.java, line 26 The constructor Telefonbuch(String) is undefined	1	
Compiler error in DemoProjekt2.java, line 28 The method neu(String, int, String) in the type Telefonbuch is not applicable for the arguments (String, int)	1	
Compiler error in DemoProjekt2.java, line 29 The method Ausgabe() is undefined for the type Telefonbuch	1	
Compiler error in DemoProjekt2.java, line 32 The constructor Telefonbuch(String) is undefined	1	
Compiler error in DemoProjekt2.java, line 33 The method neu(String, int, String) in the type Telefonbuch is not applicable for the arguments (String, int)	1	
Compiler error in DemoProjekt2.java, line 34 The method neu(String, int, String) in the type Telefonbuch is not applicable for the arguments (String, int)	1	
Compiler error in DemoProjekt2.java, line 35 The method neu(String, int, String) in the type Telefonbuch is not applicable for the arguments (String, int)	1	
Compiler error in DemoProjekt2.java, line 36 The method Ausgabe() is undefined for the type Telefonbuch	1	
Compiler error in DemoProjekt2.java, line 52 The method Ausgabe() is undefined for the type Telefonbuch	1	
Compiler error in DemoProjekt2.java, line 9 The constructor Telefonbuch(String) is undefined	1	

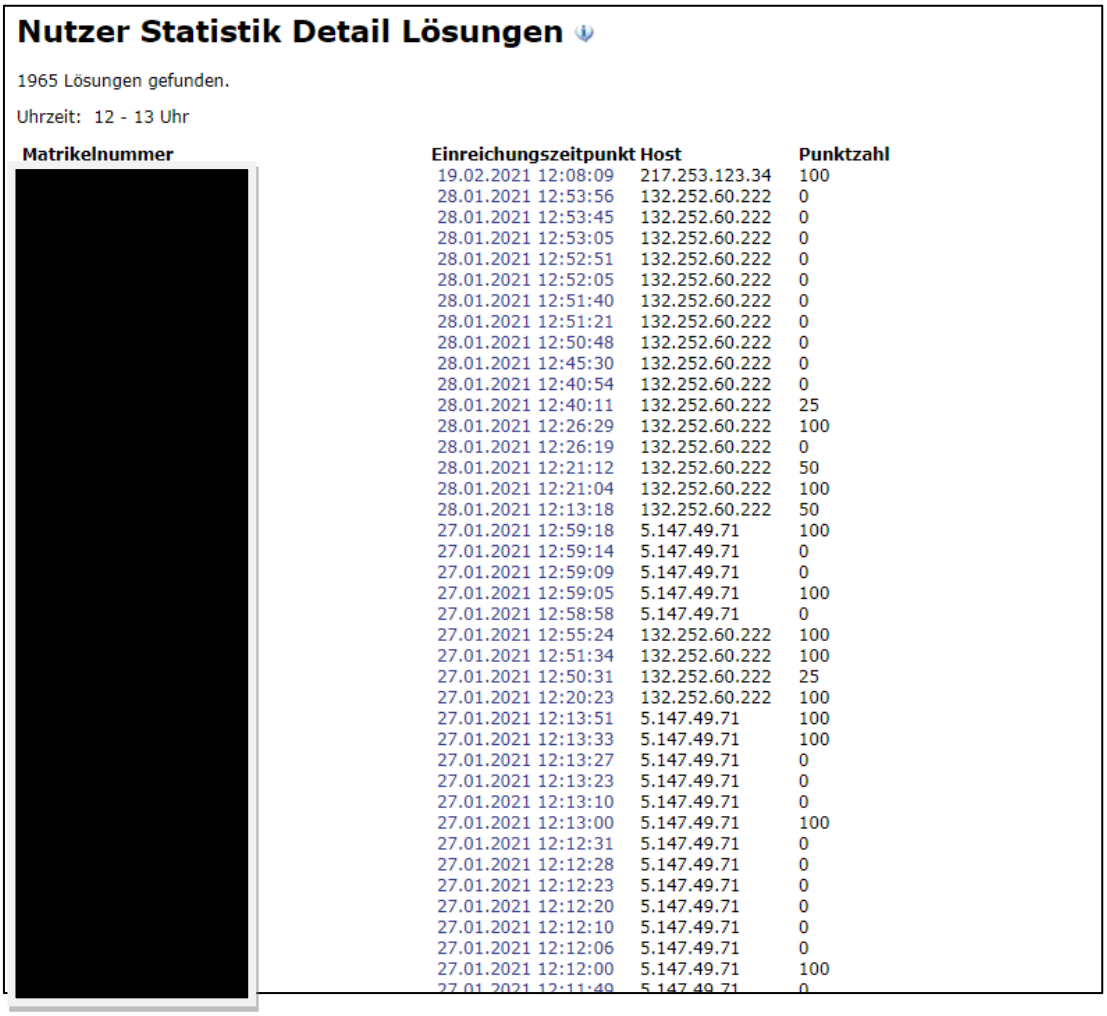
Tracing Java Checker (1)

Fehlermeldung	Häufigkeit	Stimmen
Test konnte nicht gestartet werden Der Test konnte nicht gestartet werden. Liegen im statischen Test Compilerfehler vor? Wurden alle Klassen richtig benannt und (sofern vorgegeben) im richtigen Package abgelegt?	1	

B.8.7 Nutzungsstatistik



Checker-ID	Nachrichten	Stimmen
de.uni_due.s3.jack2.backend.checkers.dynamicrchecker	51 / 1.02 pro Lösung	
de.uni_due.s3.jack2.backend.checkers.GreQLJavaChecker	179 / 2.84 pro Lösung	
de.uni_due.s3.jack2.backend.checkers.GreQLUMLChecker	38 / 9.5 pro Lösung	
de.uni_due.s3.jack2.backend.checkers.staticrchecker	33 / 1.1 pro Lösung	
de.uni_due.s3.jack2.backend.checkers.TracingChecker	292 / 2.68 pro Lösung	
de.uni_due.s3.jack2.server.core.instantcheckers.FillInChecker	20603 / 2.22 pro Lösung	
de.uni_due.s3.jack2.server.core.instantcheckers.MultipleChoiceChecker	13832 / 1.95 pro Lösung	
de.uni_due.s3.jack2.server.core.instantcheckers.OrbitalChecker	18 / 1.38 pro Lösung	



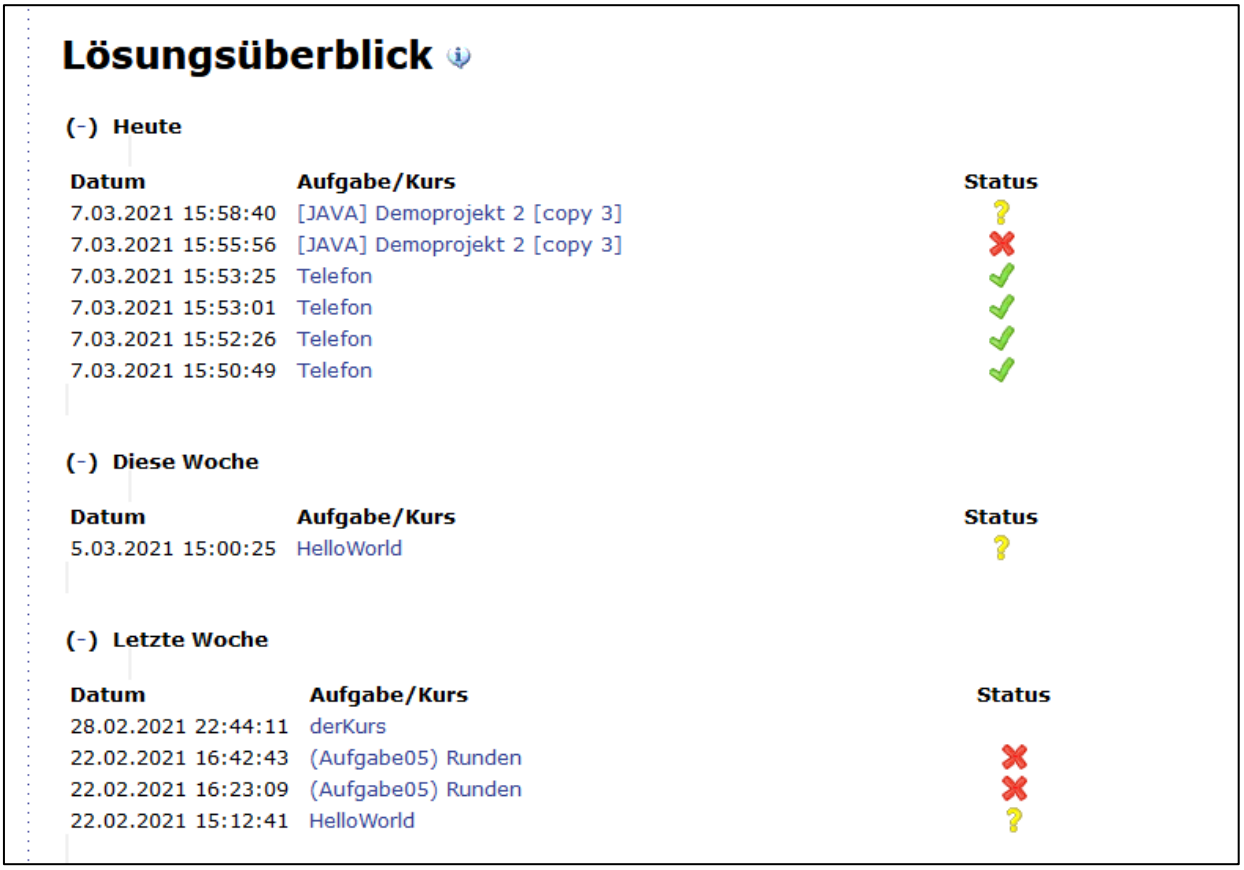
Nutzer Statistik Detail Lösungen

1965 Lösungen gefunden.

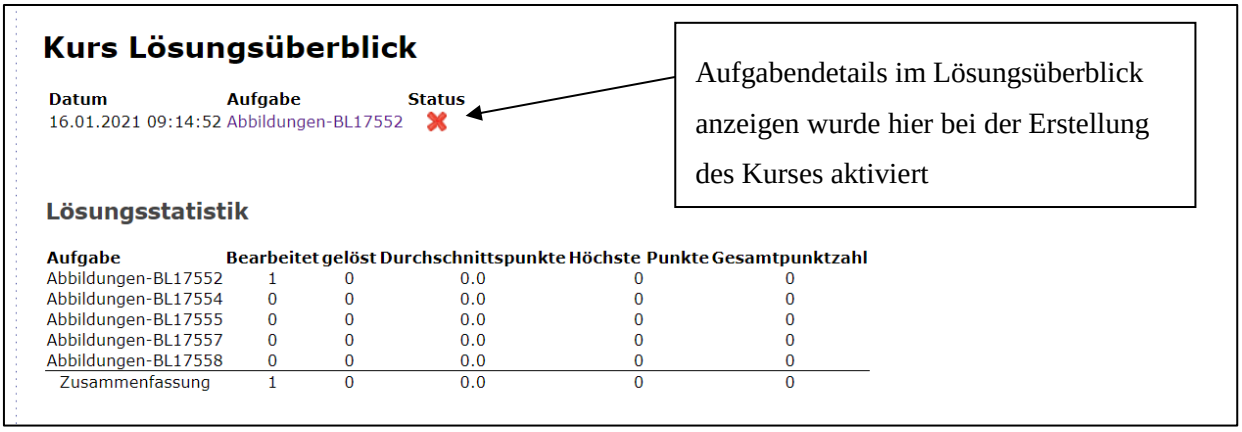
Uhrzeit: 12 - 13 Uhr

Matrikelnummer	Einreichungszeitpunkt	Host	Punktzahl
[redacted]	19.02.2021 12:08:09	217.253.123.34	100
	28.01.2021 12:53:56	132.252.60.222	0
	28.01.2021 12:53:45	132.252.60.222	0
	28.01.2021 12:53:05	132.252.60.222	0
	28.01.2021 12:52:51	132.252.60.222	0
	28.01.2021 12:52:05	132.252.60.222	0
	28.01.2021 12:51:40	132.252.60.222	0
	28.01.2021 12:51:21	132.252.60.222	0
	28.01.2021 12:50:48	132.252.60.222	0
	28.01.2021 12:45:30	132.252.60.222	0
	28.01.2021 12:40:54	132.252.60.222	0
	28.01.2021 12:40:11	132.252.60.222	25
	28.01.2021 12:26:29	132.252.60.222	100
	28.01.2021 12:26:19	132.252.60.222	0
	28.01.2021 12:21:12	132.252.60.222	50
	28.01.2021 12:21:04	132.252.60.222	100
	28.01.2021 12:13:18	132.252.60.222	50
	27.01.2021 12:59:18	5.147.49.71	100
	27.01.2021 12:59:14	5.147.49.71	0
	27.01.2021 12:59:09	5.147.49.71	0
	27.01.2021 12:59:05	5.147.49.71	100
	27.01.2021 12:58:58	5.147.49.71	0
	27.01.2021 12:55:24	132.252.60.222	100
	27.01.2021 12:51:34	132.252.60.222	100
	27.01.2021 12:50:31	132.252.60.222	25
	27.01.2021 12:20:23	132.252.60.222	100
	27.01.2021 12:13:51	5.147.49.71	100
	27.01.2021 12:13:33	5.147.49.71	100
	27.01.2021 12:13:27	5.147.49.71	0
	27.01.2021 12:13:23	5.147.49.71	0
	27.01.2021 12:13:10	5.147.49.71	0
	27.01.2021 12:13:00	5.147.49.71	100
	27.01.2021 12:12:31	5.147.49.71	0
	27.01.2021 12:12:28	5.147.49.71	0
	27.01.2021 12:12:23	5.147.49.71	0
	27.01.2021 12:12:20	5.147.49.71	0
	27.01.2021 12:12:10	5.147.49.71	0
	27.01.2021 12:12:06	5.147.49.71	0
	27.01.2021 12:12:00	5.147.49.71	100
	27.01.2021 12:11:49	5.147.49.71	0

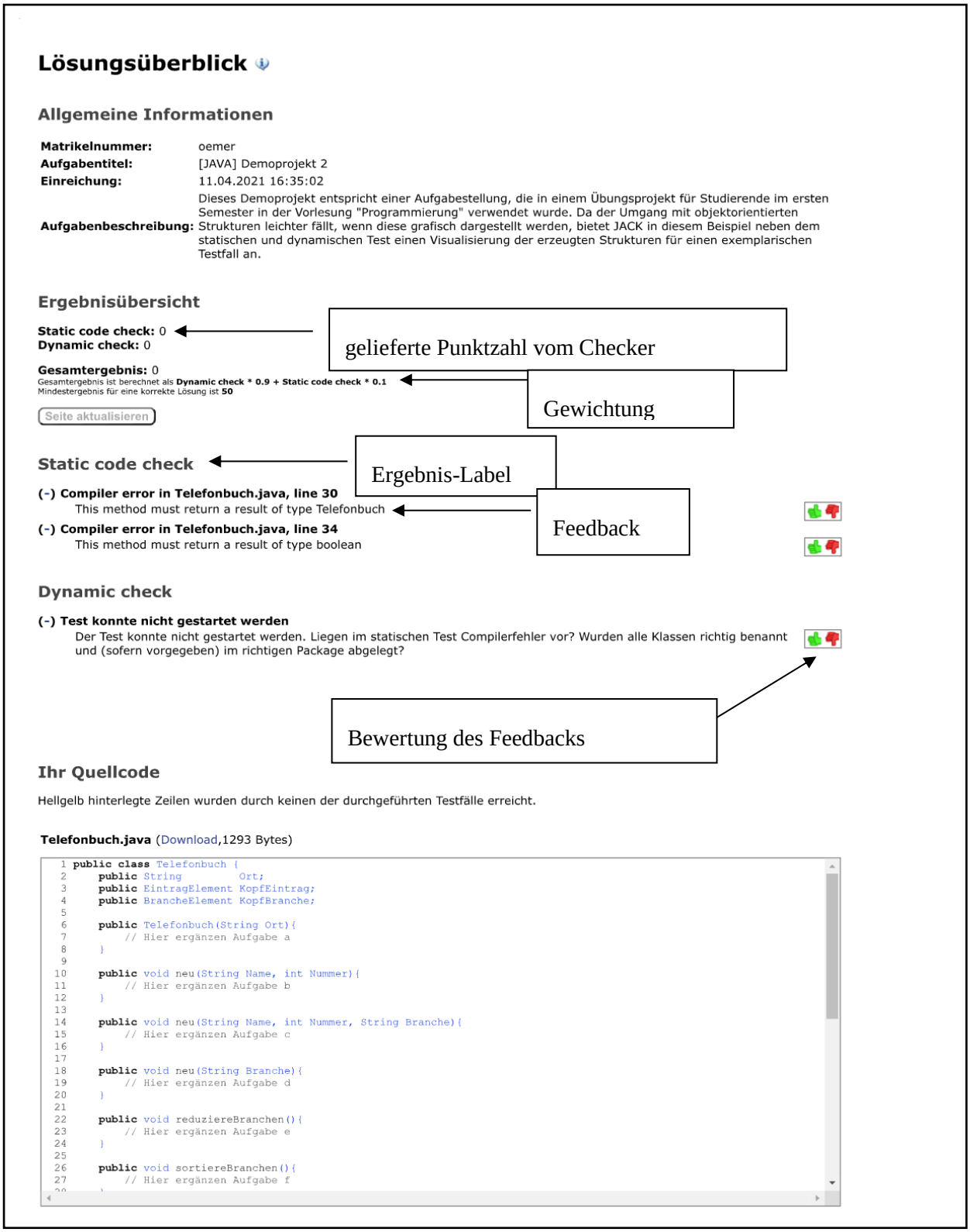
B.9 Lösungsüberblick - Student



B.9.1 Kurslösungsüberblick



B.9.2 Aufgabenergebnis


Lösungsüberblick
Allgemeine Informationen
Matrikelnummer: oemer
Aufgabentitel: [JAVA] Demoprojekt 2
Einreichung: 11.04.2021 16:35:02
Aufgabenbeschreibung: Dieses Demoprojekt entspricht einer Aufgabestellung, die in einem Übungsprojekt für Studierende im ersten Semester in der Vorlesung "Programmierung" verwendet wurde. Da der Umgang mit objektorientierten Strukturen leichter fällt, wenn diese grafisch dargestellt werden, bietet JACK in diesem Beispiel neben dem statischen und dynamischen Test einen Visualisierung der erzeugten Strukturen für einen exemplarischen Testfall an.
Ergebnisübersicht
Static code check: 0
Dynamic check: 0
Gesamtergebnis: 0
Gesamtergebnis ist berechnet als Dynamic check * 0.9 + Static code check * 0.1
Mindestergebnis für eine korrekte Lösung ist 50
Seite aktualisieren
gelieferte Punktzahl vom Checker
Gewichtung
Static code check
Ergebnis-Label
(-) Compiler error in Telefonbuch.java, line 30
This method must return a result of type Telefonbuch
(-) Compiler error in Telefonbuch.java, line 34
This method must return a result of type boolean
Feedback
Dynamic check
(-) Test konnte nicht gestartet werden
Der Test konnte nicht gestartet werden. Liegen im statischen Test Compilerfehler vor? Wurden alle Klassen richtig benannt und (sofern vorgegeben) im richtigen Package abgelegt?
Bewertung des Feedbacks
Ihr Quellcode
Hellgelb hinterlegte Zeilen wurden durch keinen der durchgeführten Testfälle erreicht.
Telefonbuch.java (Download,1293 Bytes)
1 public class Telefonbuch {
2 public String Ort;
3 public EintragElement KopfEintrag;
4 public BrancheElement KopfBranche;
5
6 public Telefonbuch(String Ort){
7 // Hier ergänzen Aufgabe a
8 }
9
10 public void neu(String Name, int Nummer){
11 // Hier ergänzen Aufgabe b
12 }
13
14 public void neu(String Name, int Nummer, String Branche){
15 // Hier ergänzen Aufgabe c
16 }
17
18 public void neu(String Branche){
19 // Hier ergänzen Aufgabe d
20 }
21
22 public void reduziereBranchen(){
23 // Hier ergänzen Aufgabe e
24 }
25
26 public void sortiereBranchen(){
27 // Hier ergänzen Aufgabe f

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original