Automatisierte Bewertung von Java-Programmieraufgaben im Rahmen einer Moodle E-Learning-Plattform

Niels Gandraß[1], Axel Schmolitzky[1]

Abstract: Die Programmiersprache Java wird an zahlreichen Hochschulen gelehrt, um Studierende mit grundlegenden Programmierkonzepten vertraut zu machen. Zur Integration von Online-Java-Programmieraufgaben in ein Moodle LMS wurde ein Fragetyp entwickelt, welcher die parallele Ausführung sowie die automatisierte Bewertung von Quellcode auf Basis von JUnit-Tests ermöglicht. Studierende erhalten hierbei ein sofortiges und individuelles Feedback, welches dynamisch schon während der Bearbeitung einer Aufgabe erzeugt wird. In diesem Beitrag werden sowohl die technischen Details des entwickelten Fragetyps als auch erste Erfahrungen mit seinem Einsatz in der Programmierlehre an der Hochschule für Angewandte Wissenschaften Hamburg thematisiert.

Keywords: automated code evaluation; feedback generation; grading; Java; Java class isolation; JUnit; learning management system; Moodle; signature validation; web service

1 Einleitung

Die an der Hochschule für Angewandte Wissenschaften (HAW) Hamburg entwickelte Lernplattform viaMINT [La18], welche auf dem Learning Management System (LMS) Moodle[2] basiert, ermöglicht sowohl angehenden Studierenden als auch solchen in den ersten Semestern ihr MINT-Wissen aufzufrischen und zu erweitern. Neben interaktivem Lernmaterial aus der Mathematik, Physik und Chemie erhalten Inhalte der Informatik vermehrt Einzug. Den Studierenden werden hierbei unter anderem Programmieraufgaben im Rahmen freiwilliger, studienbegleitender Selbsttests angeboten. Da in der Programmierlehre an vielen Hochschulen die objektorientierte Programmiersprache Java zum Einsatz kommt [Sc17], liegt hier derzeit der Fokus bei der Entwicklung neuer Fragen und Inhalte.

Aufgaben zur Schulung der *Lesekompetenz für Quellcode* sind mit den von Moodle bereitgestellten Fragetypen (u. a. Multiple-Choice, Lückentext, Kurzantwort) bereits gut umsetzbar, indem Quellcodefragmente vorgegeben und Verständnisfragen zu diesen gestellt werden. Neben der Beantwortung solcher als auch theoretischer Fragen zu Programmierkonzepten ist deren praktische Anwendung für den Lernerfolg der Studierenden ebenfalls von großer Bedeutung. Die Fähigkeit, zu einer gegebenen Anforderung passende Java-Quellcodefragmente zu formulieren, bezeichnen wir als *Schreibkompetenz für Quellcode*. Hierbei ist unter anderem

[1] Hochschule für Angewandte Wissenschaften Hamburg, Department Informatik, Berliner Tor 7, 20099 Hamburg, {Niels.Gandrass, Axel.Schmolitzky}@haw-hamburg.de
[2] Moodle LMS Website: https://moodle.org/ (Stand: 03.06.2019)

doi:10.18420/abp2019-1

eine Bewertung der Lösung auf logischer Ebene, welche über den rein textuellen Vergleich hinaus geht, essentiell [Ei03]. Da dies mit den klassischen Moodle-Fragetypen (siehe oben) bisher nicht möglich ist, wurde ein neuer Fragetyp entwickelt. Dieser ermöglicht eine effektive Ausführung, Analyse sowie Bewertung von Java-Quellcodefragmenten. Hierbei erhalten die Studierenden ein sofortiges und interaktives Feedback zu ihrem aktuellen Lösungsansatz und können dies somit noch während der Bearbeitung der Aufgabe direkt mit einbeziehen.

Neben einer Vorstellung des entwickelten Fragetyps wird auf dessen Nutzung auf der viaMINT-Plattform und an der Hochschule für Angewandte Wissenschaften Hamburg, im Rahmen der Programmierlehre für Studierende der ersten zwei Semester, eingegangen.

2 Online-Java-Programmieraufgaben im Moodle LMS

Zur Umsetzung interaktiver Java-Programmieraufgaben wurde ein Moodle-Fragetyp entwickelt, der die Korrektheit einer Lösung mithilfe von Java-Unittests bewertet. Dieser ist anfangs als eine Erweiterung des Plugins `qtype_javaunittest` [Be16] entstanden, wurde jedoch später durch eine komplette Neuentwicklung ersetzt. Nichtsdestotrotz finden einige Konzepte des an der Technischen Universität Berlin entwickelten ursprünglichen Fragetyps hier weiterhin Verwendung.

2.1 Bisherige Softwarelösungen

Neben einer Vielzahl eigenständiger Grader wie beispielsweise dem Praktomat [KSZ02], AuDoscore [OKP17], Graja [Ga16] und auch JACK[3] [GS17] existieren bereits Plugins für das Moodle LMS, welche die Ausführung und Bewertung von Java-Programmen erlauben. Diese können entweder zur Beurteilung eingereichter Komplettlösungen [Ba13; Lü17] oder zur dynamischen Auswertung einzelner Java-Aufgaben [Be16; Öz08] eingesetzt werden. Im Rahmen des hier beschriebenen Szenarios sind die Plugins des letzteren Typs relevant.

Bisherige Entwicklungen in diesem Bereich ermöglichen keine vollständige Realisation der benötigten Funktionalitäten und bieten lediglich eingeschränkte Feedbackmöglichkeiten (siehe Abschnitt 2.2). Ferner werden die bereits vorhandenen Plugins zur Zeit nicht mehr aktiv gepflegt, was zu Kompatibilitätsproblemen mit neuen Moodle-Versionen führt.

2.2 Anforderungen und Zielsetzung

Die bisher verfügbaren Fragetypen bieten bereits eine gute Basisfunktionalität, die weiterhin erhalten bleiben soll. Nichtsdestotrotz werden für den Einsatz auf der viaMINT-Plattform zusätzliche Funktionalitäten benötigt. Diese beinhalten unter anderem:

[3] JACK erlaubt eine Einbindung in Moodle als sogenannte ”Externes Tool”-Aktivität.

- Unterstützung mehrerer virtueller Dateien / Eingabepuffer
- Statische Prüfung der Code-Signatur inklusive generischer Typparameter
- Detaillierteres, auch für Programmieranfänger leicht verständliches Feedback
- Sperren einzelner Quellcode-Abschnitte (engl. *read only*)
- Verbergen einzelner Quellcode-Abschnitte, um Studierende gegen Schnittstellendefinitionen (z. B. generiertes Javadoc[4]) programmieren zu lassen (engl. *black box*)
- Reduktion der Auswertungszeit sowie parallele Bewertung mehrerer Aufgaben
- Vollständige Isolation der ausgeführten Aufgaben
- Zugriff auf Aufgaben-Metadaten (u. a. Quellcode) aus der Testklasse heraus

2.3 Aufbau des Fragetyps

Der entwickelte Fragetyp besteht aus einem Moodle-Plugin (`qtype_junittest`) sowie einem zusätzlichen Webservice (`JUnitQuestionServer`), welcher die Bewertung eines eingereichten Lösungsvorschlags mithilfe von JUnit[5] durchführt. Ersteres stellt lediglich erweiterte Eingabemöglichkeiten bereit und übernimmt die Erzeugung des Feedbacks. Hierbei kommt CodeMirror[6] als Editor zum Einsatz. Dieser erlaubt die komfortable Eingabe von Quellcode und bietet eine umfangreiche Schnittstelle für Erweiterungen, welche die Realisation der benötigten Funktionalitäten (siehe Abschnitt 2.2) erlaubt.

Studierende können, analog zu den im Praktikum eingesetzten Entwicklungsumgebungen, mithilfe einer Tab-Leiste zwischen mehreren virtuellen Dateien beziehungsweise Eingabepuffern wechseln. Dies ermöglicht die Einhaltung der Java-Konvention, je Datei maximal eine öffentliche Klasse zu definieren, und erhöht zudem die Übersichtlichkeit von Aufgaben, für deren Lösung mehr als eine Klasse benötigt wird. Ferner kann jeder dieser Tabs sowohl Quellcode-Abschnitte (jeweils durch den Studierenden bearbeitbar oder gesperrt) als auch beliebige HTML-Elemente beinhalten. Dies kann beispielsweise dazu genutzt werden, Studierende gegen eine gegebene und mithilfe von Javadoc dokumentierte Schnittstelle programmieren zu lassen, ohne ihnen Einblick in die genauen Implementationsdetails oder sogar den vollständigen Quellcode zu geben.

Die Komponenten des entwickelten Fragetyps sowie der Ablauf einer einzelnen Bewertung sind in Abbildung 1 schematisch dargestellt. Hierbei sendet die Moodle-Instanz über das Plugin eine Anfrage, bestehend aus dem zu testenden Java-Quellcode sowie Metadaten (u. a. die gewünschte maximale Ausführungszeit), an den Webservice, welcher die Bewertung der Lösung durchführt und ein Testergebnis zurückliefert. Basierend auf diesem Ergebnis generiert das Moodle-Plugin anschließend ein Feedback, welches dem Studierenden

[4] Siehe: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html (Stand: 03.06.2019)
[5] Java testing framework JUnit, Website: https://junit.org/ (Stand: 03.06.2019)
[6] CodeMirror Text Editor, Website: https://codemirror.net/ (Stand: 04.06.2019)

präsentiert wird. Die einzelnen Teilschritte sowie die bewerteten Aspekte der eingereichten Java-Programme werden detailliert in Abschnitt 2.4 beschrieben.

Abb. 1: Kommunikation der Komponenten zur Bewertung einer Lösung

Die Auskopplung der Ausführung des Java-Programms ermöglicht einige Effizienzoptimierungen sowie eine einfache horizontale Skalierbarkeit des Systems. Für kleinere Szenarien ist es jedoch ebenfalls möglich, den `JUnitQuestionServer` auf demselben Server zu betreiben, auf welchem sich auch die Moodle-Instanz befindet. Ferner wird es explizit unterstützt, den Webservice, inklusive aller für den Betrieb benötigten Bibliotheken, als eigenständigen Docker[7]-Container zu betreiben.

2.4 Bewertung einer eingereichten Lösung

Die gesamte Analyse sowie Bewertung der eingereichten Lösungen findet im `JUnitQuestionServer` statt. Hierbei handelt es sich um einen Java-Webservice, welcher eine JSON-basierte API für das Moodle-Plugin bereitstellt. Der schematische Aufbau des Servers ist in Abbildung 2 grafisch dargestellt. Eintreffende Anfragen werden in eine Warteschlange eingereiht und von einem verfügbaren Worker-Thread bearbeitet. Die Ausführungszeit einer jeden Anfrage ist begrenzt und überfällige Threads werden konsequent terminiert.

Jeder Thread des Worker-Threadpools bearbeitet genau eine der Anfragen und besitzt jeweils einen eigenen `ClassLoader`[8], in den die kompilierten Klassen geladen werden. Dieser wird stets nach Bearbeitung einer Anfrage geleert. Somit wird eine Isolation der einzelnen parallel bearbeiteten Anfragen erreicht. Hierbei führt jeder Worker folgende Schritte aus:

2.1-2 Kompilieren des Quellcodes der Lösung
2.3 Statische Validierung der Code-Signatur (optional)
2.4-5 Kompilieren der JUnit-Testklasse
2.6 Ausführen des JUnit-Runners in gesicherter Ausführumgebung

[7] Docker Container Platform, Website: https://www.docker.com/ (Stand: 04.06.2019)
[8] Siehe: https://docs.oracle.com/javase/8/docs/api/java/lang/ClassLoader.html (Stand: 03.06.2019)

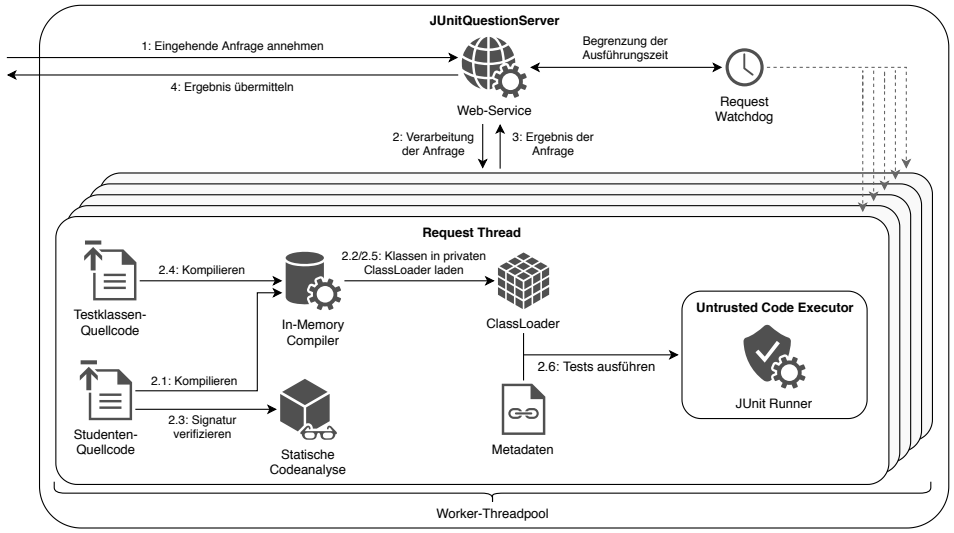

Abb. 2: Architekturdetails des entwickelten JUnitQuestionServers

Im ersten Schritt wird der Inhalt aller Quellcode-Eingabepuffer kompiliert und die hierbei entstandenen Klassen werden geladen. Zur Vermeidung zusätzlicher I/O-Operationen werden alle Quellcode-Teile direkt aus dem Arbeitsspeicher kompiliert, ohne zuvor auf die Festplatte geschrieben zu werden. Kommt es beim Übersetzen zu einem Fehler, so wird dem Studierenden eine Fehlermeldung inklusive Problembeschreibung des Compilers angezeigt.

Um sicherzustellen, dass der eingereichte Quellcode der von der Testklasse erwarteten Schnittstellenspezifikation entspricht, kann optional eine statische Signaturprüfung durchgeführt werden. Hierbei wird geprüft, ob die eingereichte Lösung allen vom Aufgabenersteller spezifizierten Eigenschaften genügt. Die erwartete Signatur entspricht der vom Java Disassembler `javap`[9] generierten Signaturbeschreibung bzw. einer Untermenge dieser und kann somit automatisiert aus der Musterlösung erzeugt werden. Zur Verifikation der Signatur wird mithilfe von ANTLR[10] und einer passenden Java-Grammatik ein abstrakter Syntaxbaum (AST) aus dem Code sowie der erwarteten Signatur erzeugt und verglichen. Dies ermöglicht es, komplexen Fehlermeldungen beim Kompilieren der Testklasse vorzubeugen und stattdessen eine für Programmieranfänger leicht verständliche Fehlermeldung zu generieren. Außerdem erlaubt dieses Vorgehen das Verifizieren generischer Typparameter, welche durch die *Type Erasure*[11] zur Laufzeit nicht mehr geprüft werden können.

Im Anschluss wird die Testklasse kompiliert und ebenfalls in den `ClassLoader` des Threads geladen. Zusammen mit einer Menge an Metadaten (u. a. kompletter Quellcode, Kompilierzeit, Timeouts) werden die Klassen an den JUnit-Runner übergeben und die zugehörige Testklasse

[9] Siehe: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javap.html (Stand: 04.06.2019)
[10] **AN**other **T**ool for **L**anguage **R**ecognition, Website: https://www.antlr.org/ (Stand: 03.06.2019)
[11] Siehe: https://docs.oracle.com/javase/tutorial/java/generics/erasure.html (Stand: 03.06.2019)

ausgeführt. Zur Absicherung des Systems geschieht dies in einem durch den Java-eigenen *Security Manager*[12] abgesicherten Kontext. Hierdurch wird beispielsweise der Zugriff auf System- sowie Netzwerkressourcen unterbunden und ebenfalls die Menge der verfügbaren Programmbibliotheken eingeschränkt. Diese Sicherheitsrichtlinien können vom Serveradministrator nach Bedarf individuell angepasst, sollten jedoch so restriktiv wie möglich gewählt werden.

Ist die Durchführung aller definierten Testfälle abgeschlossen, wird ein Testbericht generiert und von dem Webservice zurück an das Moodle-Plugin übergeben. Hierbei wird zu jedem der Testfälle eine individuelle Fehlermeldung erzeugt, die dem Studierenden anschließend angezeigt wird. Diese kann sowohl eine interne Java-Fehlermeldung inklusive Stacktrace als auch eine extra aufbereitete Meldung mit zusätzlichen Hinweisen sein. Dies erlaubt es dem Aufgabenersteller, eine dem Zielpublikum angemessene Fehlermeldung zu spezifizieren. Gerade Programmieranfängern kann somit ein einfach verständliches Feedback gegeben werden, welches ohne komplexe Java-Interna auf den Grund des Fehlschlagens eines Tests hinweisen kann. Kommt es ferner während der Ausführung einer Testklasse zu einem Timeout, so wird die Ausführung abgebrochen und ein entsprechendes Feedback mit dem Hinweis auf eventuelle Endlosschleifen oder Deadlocks erzeugt.

2.5 Integrationsmöglichkeiten des Webservices

Die bereitgestellte JSON-Schnittstelle des `JUnitQuestionServers` ist agnostisch gegenüber dem eingesetzten Lernmanagementsystem und kann somit potenziell von einer Vielzahl weiterer Softwarelösungen genutzt werden. Für eine Auswertungs-Anfrage an den Webservice werden lediglich die folgenden Daten benötigt: 1) Lösungs-Quellcodefragmente 2) Erwartete Signatur 3) Testklassen-Quellcode 4) Vom Client gewünschter Timeout[13]

Durch die Aufteilung in Plugin- und Serverkomponente können somit Erweiterungen für alternative LMS entwickelt werden. Ferner kann der Webservice so auch zur automatisierten Bewertung von abgegebenen Lösungen im Rahmen von Prüfungsvorleistungen oder Klausuren eingesetzt werden.

3 Einsatz an der HAW Hamburg

An der HAW Hamburg werden bereits seit mehreren Semestern Java-Programmieraufgaben über die Onlineplattform viaMINT angeboten. Die Bearbeitung ist für die Studierenden freiwillig und kann zeitlich unabhängig von laufenden Lehrveranstaltungen erfolgen. Die Aufgaben richten sich an Programmieranfänger der unterschiedlichen Bachelorstudiengänge und thematisieren grundlegende Konzepte der Programmierlehre. Das zur Verfügung

[12] Siehe: https://docs.oracle.com/javase/tutorial/essential/environment/security.html (Stand: 03.06.2019)

[13] Der durch den Client spezifizierte Timeout kann aus Sicherheitsgründen höchstens der im Server hinterlegten maximalen Ausführungszeit entsprechen.

gestellte Angebot beinhaltet sowohl theoretische und *Lesekompetenz*-Fragen als auch praktische Programmieraufgaben zur Schulung der *Schreibkompetenz von Quellcode*.

Letztere behandeln im ersten Semester der zweisemestrigen Programmiergrundausbildung lediglich *Objekt-basierte* [We87] Aspekte von Java. Der Umfang einzelner Aufgaben variiert zwischen einfachen Einzeiler-Lösungen und grundlegenden Interaktionen zweier Objekte beziehungsweise Klassen. Hierbei ermöglicht die vorgelagerte statische Signaturprüfung es, den Studierenden, anstelle komplexer interner Compilerfehler der für sie nicht einsehbaren Testklasse, leicht verständliche Fehlermeldungen zu präsentieren. Besonders Programmieranfänger werden somit vor einigen Stolpersteinen bewahrt, was gerade bei der individuellen Bearbeitung der Aufgaben dem Erhalt des Lernflusses dienlich ist.

Im zweiten Semester der Programmiergrundausbildung werden fortgeschrittene Themen der *Objekt-orientierten* [We87] Programmierung behandelt. Dies beinhaltet unter anderem die Trennung von Typ- und Implementationsvererbung sowie Generizität. Einzelne Aufgaben umfassen nun mindestens eine, meist jedoch mehrere Klassen sowie die sich hieraus ergebenden komplexeren Klassengeflechte inklusive Vererbungsstrukturen. Die Unterstützung mehrerer virtueller Dateien beziehungsweise Eingabepuffer ist für diese Art der Aufgaben unabdingbar. Die optionale Signaturprüfung dient nun weniger dem Abfangen komplexer Fehlermeldungen als der Ermöglichung von Korrektheitstests der generischen Typparameter, welche zur Laufzeit aufgrund der *Type Erasure* nicht zu prüfen sind.

Obwohl die angebotenen Inhalte von den Studierenden freiwillig und außerhalb der Präsenzzeiten bearbeitet werden, konnte eine hohe Beteiligung sowie eine, wie auch schon bei vergleichbaren Angeboten [St14] aufgetretene, überaus positive Resonanz auf das Zusatzangebot festgestellt werden. Gerade auch die vorlesungs- sowie praktikumsbegleitende Nutzung des Onlineangebots wurde von den Studierenden im Rahmen der semesterweise durchgeführten Lehrevaluationen häufig als positiv hervorgehoben.

Momentan beschäftigt sich eine Arbeitsgruppe an der HAW Hamburg mit der Entwicklung weiterführender Programmieraufgaben. Hierbei werden erstmals auch fortgeschrittene Themen der Java-Programmierung wie beispielsweise Generizität oder Nebenläufigkeit im Rahmen von Online-Programmieraufgaben behandelt. Ferner ist ein Einsatz des entwickelten Fragetyps im Kontext von Prüfungsvorleistungen oder E-Klausuren in der Zukunft denkbar.

4 Zusammenfassung und Ausblick

Der vorgestellte Moodle-Fragetyp ist erfolgreich seit mehreren Semestern an der HAW Hamburg im Einsatz und wird seither erweitert. Das entwickelte Plugin erlaubt unter anderem die Umsetzung von Java-Programmieraufgaben im Moodle LMS, welche auch fortgeschrittene Programmierkonzepte thematisieren. Studierende erhalten die Möglichkeit, ein sofortiges und interaktives Feedback auf Basis ihrer aktuellen Lösung einzuholen, noch bevor die Aufgabe abgegeben wird. Hierbei wurden die zur Verfügung stehenden Feedbackmöglichkeiten erweitert, wovon sowohl Programmieranfänger als auch Studierende der

höheren Semester profitieren. Ferner konnte das Abbilden von Klassengeflechten, was gerade bei der objektorientierten Programmierung unabdingbar ist, durch die Einführung mehrerer virtueller Dateien beziehungsweise Eingabepuffer ermöglicht werden. Schlussendlich war es durch die Entwicklung eines optimierten Java-Webservices zur Bewertung der eingereichten Lösungen möglich, die benötigte Auswertungszeit deutlich zu reduzieren.

Potenzielle Erweiterungen des entwickelten Fragetyps beinhalten unter anderem die Integration eines Frameworks zur statischen Codeanalyse. Somit können sowohl weitere Eigenschaften der eingereichten Lösung mit in die Bewertung einfließen als auch den Studierenden ein Feedback zur Qualität des eingereichten Quellcodes gegeben werden. Ferner ist die Integration von Testmöglichkeiten für GUI-Anwendungen ein Themengebiet, welches in späteren Versionen Einzug erhalten kann.

Dieser Beitrag wurde im Rahmen der hochschulübergreifenden MINTFIT E-Assessment Initiative durch die Behörde für Wissenschaft, Forschung und Gleichstellung (BWFG) Hamburg gefördert.

Literatur

[Ba13] Becker, S.; et al.: Prototypische Integration automatisierter Programmbewertung in das LMS Moodle. In: 1. Workshop Autom. Bewertung von Programmieraufgaben. 2013.

[Be16] Bertalan, G.; Rumler, M.: Moodle Fragetyp: javaunittest (`qtype_javaunittest`), Version 2.03, Technischen Universität Berlin, März 2016, URL: `https://moodle.org/plugins/view.php?plugin=qtype_javaunittest`, Stand: 31. 05. 2019.

[Ei03] Eichelberger, H.; Fischer, G.; Grupp, F.; von Gudenberg, J. W.: Programmierausbildung Online. In: DeLFI. 2003.

[Ga16] Garmann, R.: Graja - Autobewerter für Java-Programme, Techn. Ber., 2016, S. 20.

[GS17] Goedicke, M.; Striewe, M.: 10 Jahre automatische Bewertung von Programmieraufgaben mit JACK. In: INFORMATIK 2017. Gesellschaft für Informatik, Bonn, S. 279–283, 2017.

[KSZ02] Krinke, J.; Störzer, M.; Zeller, A.: Web-basierte Programmierpraktika mit Praktomat. Softwaretechnik-Trends 22/3, 2002.

[La18] Landenfeld, K.; Göbbels, M.; Hintze, A.; Priebe, J.: A Customized Learning Environment and Individual Learning in Mathematical Preparation Courses. In: Distance Learning, E-Learning and Blended Learning in Mathematics Education: International Trends in Research and Development. Springer International Publishing, Cham, S. 93–111, 2018.

[Lü17] Lückemeyer, G.: Moodle Plugin: JUnit Exercise Corrector (`assignsubmission_mojec`), Version 1.0, Hochschule für Technik Stuttgart, Jan. 2017, URL: `https://moodle.org/plugins/assignsubmission_mojec`, Stand: 31. 05. 2019.

[OKP17] Oster, N.; Kamp, M.; Philippsen, M.: AuDoscore: Automatic Grading of Java or Scala Homework. In: 3. Workshop Automatische Bewertung von Programmieraufgaben. 2017.

[Öz08] Özcan, S.: Moodle Fragetyp: sojunit (`qtype_sojunit`), Sep. 2008, URL: `https://moodle.org/mod/forum/discuss.php?d=102690`, Stand: 31. 05. 2019.

[Sc17] Schmolitzky, A.: Zahlen, Beobachtungen und Fragen zur Programmierlehre. In: Tagungsband 15. Workshop "Software Engineering im Unterricht der Hochschulen". 2017.

[St14] Stöcker, A.; Becker, S.; Garmann, R.; Heine, F.; Kleiner, C.; Werner, P.; Grzanna, S.; Bott, O. J.: Die Evaluation generischer Einbettung automatisierter Programmbewertung am Beispiel von Moodle und aSQLg. In: DeLFI. 2014.

[We87] Wegner, P.: Dimensions of Object-based Language Design. In: Conference Proceedings. OOPSLA '87, ACM, Orlando, Florida, USA, S. 168–182, 1987.