Erfahrungen bei der Integration des Autograding-Systems CodeOcean in die universitäre Programmierausbildung

Peter Amthor[1], Ulf Döring[1], Daniel Fischer[1], Jonas Genath[1] und Gunther Kreuzberger[1]

Abstract: Eine effektive und effiziente universitäre Programmierausbildung erfordert zunehmend den Einsatz automatisierter Bewertungssysteme. Im Rahmen des Projekts examING[2] erprobt das Teilprojekt AutoPING den Einsatz des quelloffenen Autograding-Systems CodeOcean für übergreifende Lehrangebote und Prüfungen an der TU Ilmenau mit dem Ziel, selbstgesteuertes und kompetenzorientiertes Lernen zu ermöglichen und zu fördern. Der Beitrag gibt einen Überblick über erste Projekterfahrungen bei der Adaption didaktischer Szenarien in der Programmierausbildung hin zu testgetriebener Softwareentwicklung sowie der Generierung von Feedback. Es werden wesentliche Erkenntnisse aus Sicht der Studierenden und Lehrenden erörtert, Herausforderungen und Lösungsansätze zur Integration und Erweiterung von CodeOcean für neue Anwendungsfelder diskutiert sowie zukünftige Perspektiven eröffnet.

Keywords: Autograder; Programmieren; automatisches Feedback; CodeOcean

1 Einleitung

Das Vermitteln von Programmiersprachen, das Verstehen von Algorithmen und das Erlernen von Entwicklungsmethoden erfordert Zeit, Engagement und intensive Übung. Möglichst direktes und qualitativ hochwertiges Feedback ist für Lernende hierbei von großer Bedeutung. Dieses kontinuierlich und zum richtigen Zeitpunkt bereitzustellen, stellt noch immer eine Herausforderung dar [Je22]. Mit testgetriebener Softwareentwicklung kann Feedback automatisch generiert werden. Eines dieser Systeme ist das Autograding-System CodeOcean. Dank seiner Flexibilität und Skalierbarkeit lässt es sich disziplinübergreifend einsetzen und fördert Qualitätssicherung sowie allseitigen Wissensaustausch in der Programmierausbildung.

In diesem Beitrag geben wir einen Einblick in unsere Erfahrungen bei der Umstellung didaktischer Szenarien in der Programmierausbildung auf die testgetriebene Softwareentwicklung und stellen den dafür umgesetzten, disziplin- und themenübergreifenden Einsatz von CodeOcean in der digitalen Hochschullehre vor. Anhand von vier Anwendungsfällen aus verschiedenen Lehrangeboten der TU Ilmenau erörtern wir wesentliche Erkenntnisse, beschreiben Herausforderungen und diskutieren Lösungsansätze.

[1] Technische Universität Ilmenau, PF 100565, 98684 Ilmenau, Deutschland
{peter.amthor,ulf.doering,daniel.fischer,jonas.genath,gunther.kreuzberger}@tu-ilmenau.de

[2] gefördert durch die Stiftung Innovation in der Hochschullehre

doi:10.18420/abp2023-9

2 CodeOcean als einheitliche Plattform

Angesichts steigender Anforderungen an die Qualität der Hochschullehre und den Einsatz digitaler Bildungstechnologien suchten wir zu Beginn des Projekts nach einer Alternative zur bislang heterogenen Landschaft digitaler Werkzeuge in Programmierkursen an der TU Ilmenau. Ausgehend von unseren langjährigen, durchaus gemischten Erfahrungen zum Einsatz von Moodle-Tests, dem Moodle-Plugin *Virtual Programming Lab* [RRH12] sowie mehreren Eigenentwicklungen wollten wir eine im Produktivbetrieb bewährte Lösung zur automatischen Bewertung von Programmieraufgaben und Quelltext finden, die sich in das hochschulweit eingesetzte Lernmanagement-System (LMS) Moodle[3] integrieren lässt, leicht auf neue Anwendungsszenarien adaptierbar ist und so kursübergreifend sowie studiengangsübergreifend etabliert werden kann. Im Ergebnis einer Recherche (vgl. [SS22]) wählten wir das Autograding-System CodeOcean zur intensiven Erprobung aus.

Die Open-Source-Software CodeOcean[4] wurde vom Hasso-Plattner-Instituts (HPI) für die Durchführung von Programmierübungen in Massive Open Online Courses entwickelt. Das Autograding-System bietet nicht nur zahlreiche Funktionen zur automatischen Bewertung von Programmieraufgaben, sondern stellt auch eine umfassende, webbasierte Ausführungs- und Entwicklungsumgebung mit vielfältigen Protokollierungs-, Hilfs- und Kollaborationsfunktionen zur Verfügung. Durch die Implementierung der Learning-Tools-Interoperability-Schnittstelle kann es mit LMS wie z. B. Moodle kommunizieren. Diese Funktionen ermöglichen es, den gesamten Lehr- und Bewertungsprozess bei der Durchführung von Übungen und Prüfungen für verschiedene Programmiersprachen in einer bestehenden IT-Infrastruktur zu unterstützen. Das HPI verwendet das System für über 60 000 Nutzer und hat damit die Skalierbarkeit in der Praxis nachgewiesen [Se21].

CodeOcean dient als Frontend für die Studierenden und organisiert deren Einreichungen. Docker[5]-Container weisen den Nutzern individuelle, je nach Programmiersprache und Aufgabenstellung verschiedene, spezialisierte Ausführungsumgebungen (Kompilier- und Ausführungstools, Bibliotheken, etc.) zu. Indem jedem Nutzer temporär ein separater Container zugewiesen wird, können die Einreichungen untereinander nicht interferieren und es wird keine spezielle Hard- oder Software für Studierende vorausgesetzt [Se21].

3 Python-Programmierübungsreihe für Studienanfänger

Erste Erfahrungen mit CodeOcean sammelten wir in einem Erstsemesterkurs für Studierende aus verschiedenen Studiengängen. Ziel in diesem Kurs ist es, Studierenden mit wenig bis keiner Programmiererfahrung erste Programmierfähigkeiten zu vermitteln. Hierfür entwickelten wir eine Python-Programmierübungsreihe und setzten diese in CodeOcean um. Der

[3] siehe: `https://moodle.org`
[4] Quellcode auf GitHub: `https://github.com/openHPI/codeocean`
[5] siehe: `https://www.docker.com`

verfolgte Lehransatz basiert auf dem Konzept der testgetriebenen Entwicklung und legt den Fokus darauf, dass Studierende das Programmieren in Python schrittweise erlernen, indem sie eigenständig Aufgaben bearbeiten. Notwendige grundlegende Programmierkonzepte werden durch Kick-off-Seminare und einen Online-Kurs des openHPI[6] vermittelt.

Die Übungsreihe ist in ein praktisches Anwendungsszenario eingebettet und erfordert einen iterativen Entwicklungsansatz. Sie besteht aus fünf Teilaufgaben, die verteilt über das Semester hinweg zu bearbeiten sind. Begleitet werden die Studierenden durch Tutoren. Das übergeordnete Ziel besteht darin, ein Konsolenprogramm zu entwickeln, das wesentliche Funktionen eines Fahrkartenautomaten umsetzt. Zu jeder Teilaufgabe erhalten die Studierenden einen vorgegebenen Entwicklungsstand des Programms und eine textuelle Beschreibung der Anforderungen, die das Programm zusätzlich erfüllen soll. Alle Informationen inkl. des Quellcodes werden über CodeOcean zur Verfügung gestellt. Während der einzelnen Implementierungsphasen haben die Studierenden jederzeit die Möglichkeit, ihre Lösungen zu testen. Für dieses Autograding haben wir aufgabenspezifische Unittests entwickelt, die individuelles Feedback liefern, einschließlich detaillierter Meldungen, wenn Anforderungen nicht oder unzureichend erfüllt werden [Pa22]. Die erwarteten Ausgaben des Konsolenprogramms werden getrennt von den Tests im JSON-Format definiert. Dadurch lassen sich Aufgabenvariationen des Fahrkartenautomaten, die keine funktionalen Änderungen vornehmen, leicht umsetzen. Um sicherzustellen, dass die Studierenden wesentliche Good Practices zur Codeformatierung und -strukturierung einhalten und erlernen, erfolgen mit Hilfe eines Linters zusätzlich aufgabenunabhängige Syntaxtests und Tests des Programmierstils.

Die Python-Übungsreihe haben wir in semesterbegleitenden Übungen in zwei Studierendenjahrgängen erprobt, an denen insgesamt etwa 170 Studierende teilnahmen. Zur Evaluation führten wir Befragungen durch. Aus diesen sowie aus unseren Beobachtungen im Lernprozess wurde deutlich, dass die Verwendung von CodeOcean einen maßgeblichen Beitrag zur Zufriedenheit der Studierenden geleistet hat. Die Möglichkeit, kontinuierlich automatisches Feedback zum Quellcode abzurufen, wurde als äußerst positiv bewertet. Mithilfe des Feedback konnten die Studierenden Fehler selbst erkennen und ihre Lösungen kontinuierlich verbessern. Allerdings gab es auch Kritikpunkte, die sich vor allem auf ungenaues, verwirrendes oder nicht gut nachvollziehbares Feedback sowie unklare Aufgabenstellungen bezogen. Einige Studierende bezeichneten das Feedback als überladen, da es teils viele oder zu detaillierte Fehlermeldungen enthielt. Um diese Probleme anzugehen, fand eine Überarbeitung der Aufgabenstellungen und aller Unittests statt. In den Aufgaben wurden u. a. mehr Vorgaben eingeführt und diese präziser erläutert. Zudem haben wir das individuelle Feedback übersichtlicher und transparenter gestaltet, z. B. durch eine eindeutige Darstellung der In- und Outputs der Testfunktionen und die Ergänzung exemplarischer Lösungshinweise. Dadurch war eine einfachere Fehleranalyse und -behebung für die Studierenden möglich.

Mit dem zweiten Studierendenjahrgang führten wir als zusätzliche Überprüfung des Lernfortschritts nach dem Absolvieren der Übungsreihe einen Abschlusstest durch. Dadurch

[6] https://open.hpi.de/courses/pythonjunior-schule2022

sammelten wir erste Erfahrungen mit dem Einsatz von CodeOcean in einer prüfungsnahen Situation. Die Studierenden mussten dazu unter Prüfungsbedingungen, d. h. in Präsenz und unter Aufsicht von Lehrenden, ihre Programmierfähigkeiten durch das Lösen von zwei zufällig ausgewählten Aufgaben aus einer Aufgabensammlung nachweisen. Dafür wurden weniger komplexe Aufgaben als bei der semesterbegleitenden Übungsreihe verwendet, wie z. B. das Potenzieren zweier Zahlen. CodeOcean erwies sich auch hier als effektives Werkzeug, die Lösungen der Studierenden zu bewerten und ihnen Feedback zu geben. Die anschließenden Rückmeldungen der Studierenden zu dem Einsatz von CodeOcean in dieser prüfungsnahen Situation waren sehr positiv. Daher planen wir, diese Art des Einsatzes weiter auszubauen und in den kommenden Semestern eine digitale Abschlussprüfung mit CodeOcean zu entwickeln und zu erproben.

4 Integration in den Kurs Algorithmen und Programmierung (AuP)

In AuP erwerben Studierende verschiedener Ingenieurstudiengänge Grundlagen der Anwendung ausgewählter Algorithmen und Datenstrukturen sowie der Programmierung mit Java. Dabei arbeiten sie oft selbständig mit Programmier- und Verständnisaufgaben sowie entsprechenden Beispiellösungen. Da Anfängern das kritische Beurteilen eigener Programmierlösungen meist schwer fällt, integrierten wir CodeOcean in den Lehrbetrieb. Nun steht Feedback auch Studierenden zur Verfügung, die nicht an Präsenzübungen/ -tutorien teilnehmen. Zudem werden Betreuende von Kontrollaufgaben entlastet und gewinnen Zeit zur Klärung von Verständnisfragen oder für individuelle Lösungshinweise. Links im LMS sowie den PDF-Aufgabenblättern sollen zum Einreichen der Lösung in CodeOcean anregen. Insgesamt hat etwa die Hälfte der ca. 100 aktiv am Kurs teilnehmenden Studierenden CodeOcean regelmäßig benutzt.

Die Umsetzung der Analyse- und Bewertungsansätze für das Autograding wird durch die von CodeOcean bereitgestellte Infrastruktur sehr erleichtert. Dies ermöglichte uns die Implementierung von Tests, welche auf spezielle Anforderungen von Programmieranfängern ausgerichtet sind. Gleichzeitig gelang es, den Aufwand für die Testerstellung und -pflege gering zu halten. Wichtigste Anforderung an den AuP-Autograder war, dass er auch für Teillösungen (kompilierbar, aber einige geforderte Member fehlen) einsetzbar ist. Dies impliziert, dass herkömmliche JUnit-Tests, welche zur Laufzeit komplette studentische Lösungen benötigen, nicht geeignet sind. Der von uns gewählte Ansatz beruht auf der *reflection*-basierten Analyse studentischer Lösungen, siehe [Dö23]. Wenn Tests aufgrund fehlender Lösungsteile nicht ausgeführt werden können, dann wird dies mitgeteilt. Aus lernmethodischer Sicht ist dies sinnvoll, denn die Studierenden können weiterhin in ihrer gewohnten IDE (z. B. Eclipse) arbeiten und bei Bedarf (Teil-) Lösungen via CodeOcean bewerten lassen. Dabei ist eine ihrem Wissensstand angemessene Formulierung der Fehlermeldungen sehr wichtig. Insbesondere bei Funktionstests sind zur Nachvollziehbarkeit auch die verwendeten Testdaten (Methodenparameter und initialer Zustand von Member-Variablen) sowie die festgestellten und erwarteten Ergebnisse (Ausgaben, Rückgaben, Variablenwerte) mitzuteilen.

Aktuell verwendet der AuP-Autograder eine Java-Einbindung (in CodeOcean „Adapter“) für JUnit. Dazu emuliert er die JUnit-Fehlerausgaben. Positives Feedback und Verbesserungshinweise lassen sich so aber nicht zurückgeben. Deshalb streben wir die Entwicklung eines Adapters an, der die vom Autograder vergebenen Punkte und das textuelle Feedback entgegen nimmt. Zudem sollte der Autograder Formatierungen (insbesondere Hervorhebungen) ins Feedback integrieren können, um so die Lesbarkeit für Studierende zu verbessern.

5 Erprobung automatisierter E2E-Tests in einem WebDev-Kurs

In einem studiengangübergreifenden Grundlagenkurs zur Entwicklung web-basierter Kommunikationsangebote erlernen die Studierenden, Inhalte mit HTML5 strukturiert zu beschreiben, mittels CSS3 zu gestalten sowie mit Hilfe von Javscript zu manipulieren und Nutzereingaben persistent zu halten. Anhand von Übungsaufgaben erarbeitete Kompetenzen sollen sie abschließend in einem eigenen Web-Projekt zusammenführen. Dieser Anwendungsfall erweitert das in [Au21] beschriebene Szenario, indem bereits die strukturierte Beschreibung von Webseiten-Inhalten und auch die (serverseitige) Datenhaltung als Testfall betrachtet wird. Wir adaptierten den dafür vorgeschlagenen Ansatz, den Kompetenzerwerb mittels end-to-end (E2E)-Tests zu überprüfen (vgl. [Le16]), und realisierten ihn in CodeOcean mithilfe einer speziellen Ausführungsumgebung.

Das von CodeOcean empfohlene Container-Basis-Image[7] erweiterten wir zu einer für automatisierte E2E-Tests geeigneten Single-Container-Ausführungsumgebung. Konkret ergänzten wir einen lokalen Webserver zur Bereitstellung der zu prüfenden Lösung sowie einen über das Selenium-Webdriver-Protokoll gesteuerten Webbrowser zur Ausführung der einzelnen Unit-Tests (vgl. [Pa22]). Die zu testende Lösung wird durch CodeOcean als Ordner im Dateisystem der Ausführungsumgebung bereitgestellt. Den lokalen Webserver konfigurierten wir deshalb so, dass er diesen Ordner als Basisverzeichnis verwendet.

Für den produktiven Einsatz im nächsten Lehrzyklus mit ca. 150 Studierenden übertragen wir folgende Beispielfälle in CodeOcean. In Use Case 1 soll ein HTML-Grundgerüst gemäß eines in der Aufgabenstellung beschriebenen Wireframe-Modells um typische Inhaltsbereiche mittels HTML5-Container-Elementen ergänzt werden. Der Funktionstest traversiert die DOM-Struktur und gleicht sie mit einem Musterdatensatz im JSON-Format ab. Das Feedback adressiert fehlende oder überzählige Bereiche sowie ungünstig gewählte Container-Elemente. In Use Case 2 sollen komplex strukturierte Inhaltsformen, wie Wort-Bild-Kombinationen, Tabellen oder Aufzählungen, in eine elementar strukturierte Webseite eingefügt werden. Der Funktionstest vergleicht die DOM-Teilstruktur mit einem Musterdatensatz im JSON-Format. Das Feedback adressiert Fehler in der Wahl und Verschachtelung geeigneter HTML-Elemente. In Use Case 3 soll mittels *unobtrusive Javascript* (vgl. [Ko14]) bestimmten, mit DOM-Methoden zu selektierenden Elementen einer vorgegebenen HTML-Struktur ein Eventlistener hinzugefügt werden, der eine vorgegebene Eventhandler-Funktion

[7] siehe: `https://github.com/openHPI/dockerfiles`

für eine spezifische Interaktion registriert. Der Funktionstest simuliert die vorgegebene Nutzerinteraktion und prüft das Eintreten der vom vorgegebenen Eventhandler vorgenommenen Inhaltsänderung. Das Feedback adressiert Fehler in der Selektion der korrekten Elemente sowie der korrekten Registrierung des Eventlisteners. In Use Case 4 soll für eine vorgegebene Webseite mit bereits hinzugefügten Eventlistenern der Funktionsrumpf des registrierten Eventhandlers in Javascript ergänzt werden. Der Funktionstest vergleicht die nach erfolgreicher Ausführung des Eventhandlers eingetretene Inhaltsänderung mit der laut Aufgabenstellung erwarteten Änderung. Das Feedback adressiert Fehler im Programmfluss, soweit diese aus dem Vergleich von tatsächlicher mit erwarteter Ereignisantwort abgeleitet werden können.

Probleme bereitete uns zunächst der Einsatz der Ausführungsumgebung als Single-Container-Lösung mit mehreren gleichzeitig aktiven Anwendungen. Das CodeOcean-Entwicklerteam löste dieses Problem jedoch mittels Funktionsupdate[8]. Somit ist die Umsetzung von Aufgaben zur testgetriebenen Web-Entwicklung mittels automatisierter E2E-Tests in CodeOcean möglich. Mit Hilfe von parametrisierten Tests konnten wir das Feedback zu Lösungseinreichungen deutlich differenzierter gestalten.

6 Systemnahe Programmierung in Rust

Bei der systemnahen Softwareentwicklung kommen oft Programmiersprachen zum Einsatz, die Studierende weder aus dem Grundlagenstudium noch aus ihrer persönlichen Erfahrung beherrschen. Eine solche Programmiersprache ist Rust [MK14]. Sie zeichnet sich insbesondere durch eine Forcierung von robustem und effizientem Code aus. Damit empfiehlt sich Rust etwa zur Implementierung von Betriebssystemmechanismen im Kurs „Advanced Operating Systems“, trotz einer für Einsteiger vergleichsweise steilen Lernkurve [Ru18]. Im Kursverlauf bearbeiten die Studierenden in teamorientierten Workshops konzeptionelle und algorithmische Probleme, deren Lösung dann auf individuelle Programmiertätigkeiten ohne unmittelbares Lehrendenfeedback abgebildet wird. Diese Dualität der Lernziele – sowohl im domänenspezifischen Wissen und dessen Anwendung, als auch in der hierfür erforderlichen Programmiersprache – stellt uns Lehrende jedoch vor ein Ressourcenproblem. Existierendes Online-Material,[9] welches auf das individuelle Erlernen von Rust anhand generischer Anwendersoftware ausgelegt ist, kann dieses Problem nicht unter Wahrung des didaktischen Leitgedankens kooperativen, problemorientierten Lernens lösen. Zugleich soll die Entwicklung und Pflege einer spezialisierten Insellösung vermieden werden.

Um Rust-Programmierung motivierend, effizient und stoffgerecht zu lehren, lag ein Ziel dieses Anwendungsfalls daher in der Erweiterung von CodeOcean um diese bislang nicht unterstützte Programmiersprache und wurde von uns, mit Unterstützung durch das HPI, in zwei Schritten umgesetzt: (1) Erstellung eines zugeschnittenen Container-Images als

[8] vgl.: https://github.com/openHPI/poseidon/blob/main/docs/configuration.md#supported-docker-images
[9] Bspw. https://github.com/rust-lang/rustlings, https://doc.rust-lang.org/stable/rust-by-example/.

Grundlage einer neuen Ausführungsumgebung. Dieses stellt alle Sprachtools sowie, für systemnahe Entwicklung besonders relevant, Bibliotheken und Hardwareabstraktionen entsprechend der Aufgabenstellung bereit. (2) Implementierung eines CodeOcean-Adapters für das von Rust genutzte Build-Tool *cargo* in Ruby. Hierfür müssen die Gesamtzahl ausgeführter Tests, die Anzahl fehlgeschlagener Tests sowie etwaige Fehlermeldungen aus dem Container an CodeOcean übergeben werden. Bislang wurde dies mithilfe regulärer Ausdrücke implementiert, welche menschenlesbare Ausgaben eines Build-Tools filtern. Da jedoch *cargo* als Build-Tool JSON zur strukturierten Datenrepräsentation unterstützt, lassen wir stattdessen diese Ausgabe mittels eines Ruby-Standardmoduls im Adapter parsen. Somit können für zukünftige Erweiterungen des CodeOcean-Frontends auch weitere Informationen bereitgestellt werden, beispielsweise Hilfestellungen für typische Programmierfehler, welche durch *cargo* bei nicht kompilierbaren Programmen ausgegeben werden.

Allerdings mussten Automatisierungsfunktionen der Rust-Tools stellenweise limitiert werden: Da *cargo* darauf ausgelegt ist, Codeabhängigkeiten durch direkten Download der benötigten externen Quellen (Rust „Crates“) aufzulösen, muss sowohl die Ausführungsumgebung für Netzwerkzugriff konfiguriert sein, als auch der Host entsprechende Berechtigungen gewähren. Ersteres erwies sich in Machbarkeitstests mit verschiedenen Container-Images als problematisch, während Letzteres aus Sicherheitsgründen fragwürdig ist. Es hat sich daher für unseren Kurs als sinnvoll erwiesen, mehrere Ausführungsumgebungen mit verschiedenen Container-Images zu verwenden: je nachdem, ob bspw. Interprozesskommunikation, Dateisystemzugriffe oder (simulierte) Netzwerkkommunikation im Fokus der jeweiligen Aufgabe stehen, setzen wir hierfür angepasste und mit Offline-Versionen der jeweils benötigten Crates bestückte Images ein. Dies bietet den didaktischen Mehrwert, die Studierenden durch die Beschränkung auf konkret vorgegebene Drittsoftware an realistische Arbeitsbedingungen in der Praxis zu gewöhnen. Darüber hinaus können wir das Spektrum möglicher Lösungsvarianten durch verbindliche Vorgabe bestimmter APIs sinnvoll eingrenzen, was wiederum die Entwicklung passgenauer Tests begünstigt.

Im Rahmen des nächsten Lehrzyklus ist geplant, eine Rust-Übungsreihe in CodeOcean mit ca. 20 Teilnehmern prototypisch umzusetzen. Wir erwarten hieraus weitergehendes Feedback aus Teilnehmersicht, welches bspw. Impulse zur ergonomischen Neugestaltung des Frontends geben kann, um Rust-spezifische Werkzeugunterstützung in den Ausgaben umfassender zu berücksichtigen.

7 Zusammenfassung und Ausblick

Die Erprobung von CodeOcean in mehreren Kursen mit jeweils spezifischen Anforderungen unter realen Bedingungen hat gezeigt, dass bis dahin separat entwickelte Ansätze in einem einheitlichen System abgebildet werden können. Dies ermöglicht eine homogene, hochschulweite Lösung durch die viele einzelne Insellösungen vermieden werden können. Infolgedessen kann administrativer Aufwand zentralisiert und Einarbeitungsaufwand von Studierenden reduziert werden. Die Machbarkeit konnten wir in vier Anwendungsszenarien

auf dem Technology Readiness Level TRL7[10] prüfen und qualitativ bestätigen. Es zeigte sich, dass neue, vom Entwicklungsteam des HPI nicht vorhergesehene Anwendungsfälle mit vertretbarem Aufwand in CodeOcean abgebildet werden können. Entwicklungsbedarf besteht mit Blick auf die Verbesserung der Nutzungserfahrung für die Lernenden. Der Einsatz gängiger Test-Werkzeuge kommt bei der Erzeugung hilfreichen Feedbacks zum richtigen Zeitpunkt (vgl. [Je22]) für getestete Lösungseinreichungen an seine Grenzen. Unsere weiterführenden Untersuchungen befassen sich daher aufbauend auf Ergebnissen der quantitativen Beurteilung im Rahmen der kontinuierlichen Lehrevaluation u. a. mit der Weiterentwicklung der Nutzer-Schnittstelle von CodeOcean sowie der Generierung differenzierten Feedbacks.

Literaturverzeichnis

[Au21] Aubele, L.; Martin, L.; Hirmer, T.; Henrich, A.: An Architecture for the Automated Assessment of Web Programming Tasks. In. 5. ABP-Workshop. 2021.

[Dö23] Döring, U.: JUnit free checks of Java solutions in CodeOcean, https://github.com/udoering/JavaSolCheckCOJU, 2023.

[Je22] Jeuring, J. et al.: Towards Giving Timely Formative Feedback and Hints to Novice Programmers. In: ITiCSE-WGR '22. S. 95–115, 2022.

[Ko14] Koch, P.-P.: The principles of unobtrusive JavaScript, W3C Wiki, 9. Okt. 2014, URL: https://t.ly/Ym0lh.

[Le16] Leotta, M. et al.: Approaches and Tools for Automated End-to-End Web Testing. In: Advances in Computers. Bd. 101, Elsevier, S. 193–237, 1. Jan. 2016.

[MK14] Matsakis, N. D.; Klock, F. S.: The Rust Language. In: Proc. 2014 ACM SIGAda Ann. Conf. on High Integrity Language Technology. HILT '14, S. 103–104, 2014.

[Pa22] Pajankar, A.: Python Unit Test Automation: Automate, Organize, and Execute Unit Tests in Python. Apress, Berkeley, CA, 2022.

[RRH12] Rodríguez-del-Pino, J. C.; Rubio Royo, E.; Hernández Figueroa, Z.: A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. In: Proc. EEE 2012. 2012.

[Ru18] Rust Foundation: Rust survey 2018 results, Nov. 2018, URL: https://blog.rust-lang.org/2018/11/27/Rust-survey-2018.html, Stand: 12. 06. 2023.

[Se21] Serth, S. et al.: Improving the Scalability and Security of Execution Environments for Auto-Graders in the Context of MOOCs. In. 5. ABP-Workshop. 2021.

[SS22] Strickroth, S.; Striewe, M.: Building a Corpus of Task-Based Grading and Feedback Systems for Learning and Teaching Programming. iJEP 12/5, S. 26–41, 2022.

[10] vgl. https://op.europa.eu/s/yNrL, Kap. 3