Relevanz der Codequalität in einem Praktikum mit automatisch getesteten Programmierabgaben

Andre Greubel[1], Tim Hegemann[1], Marianus Ifland[1], Martin Hennecke[1]

Abstract: Systeme zur automatisierten Bewertung von Programmieraufgaben bewerten üblicherweise nicht die Codequalität. In dieser Arbeit wird an einem Praktikum mit 212 Studierenden empirisch untersucht, welche Relevanz diese der Codequalität beimessen und welche Relevanz verschiedene Qualitätsmetriken auf das Bestehen des Praktikums haben. Des Weiteren wird analysiert, ob diese Metriken von externen Faktoren wie dem Studiengang oder der Praktikumswiederholung beeinflusst werden.

Keywords: Codequalität, Softwaremetriken, Java, Grader, Automatische Bewertung, Programmieraufgaben

1 Einleitung

Der Studiengang Informatik erfreut sich aufgrund guter Jobperspektiven immer größerer Beliebtheit, was sich durch immer größere Kurse bemerkbar macht. Systeme zur automatischen Bewertung von Programmieraufgaben sind vielerorts elementar, um den wachsenden Korrekturaufwand zu stemmen.

Ein Nachteil dieser Automatisierung ist, dass das Feedback häufig nicht sehr differenziert ausfällt. Meist wird nur eine Rückmeldung über die semantische Korrektheit des Programms oder seiner Teile gegeben. Insbesondere fehlt in vielen Systemen ein Feedback zur Codequalität, mit dem Studierende die Qualität ihrer Lösungen verbessern könnten. In dieser Arbeit wollen wir analysieren, welche Codequalität typische studentische Bearbeitungen aufweisen.

Ausgangspunkt ist die Erfahrung, dass Studierende der Codequalität bei automatisierter Bewertung zumeist nur wenig Bedeutung beimessen. Das ist kritisch zu betrachten, da die Codequalität in der Praxis aufgrund der hohen Relevanz von Wartungsaufgaben elementar ist und daher auch in der Programmierausbildung einen entsprechenden Stellenwert haben sollte.

[1] Institut für Informatik, Julius-Maximilians-Universität, 97074 Würzburg, {andre.greubel, marianus.ifland, martin.hennecke}@uni-wuerzburg.de, tim.hegemann@stud-mail.uni-wuerzburg.de

doi:10.18420/abp2019-4

In dieser Arbeit wird empirisch überprüft, welche Relevanz Studierende der Codequalität einräumen, welche Aspekte der Codequalität mit dem Bestehen in einem Programmierpraktikum zusammenhängen und ob diese Metriken von externen Faktoren wie dem Studiengang oder der Praktikumswiederholung beeinflusst werden.

2 Organisatorische Rahmenbedingungen

Als Datensatz dienen Abgaben von Studierenden des Java-Programmierpraktikums, das an der Universität Würzburg im Frühjahr 2019 durchgeführt wurde. Die größten Aufgaben in diesem Praktikum haben im Regelfall bei vorgegebener Architektur einen Umfang von etwa 2000 Zeilen Quellcode (Lines of Code).

Das Praktikum wird in organisatorischer Einheit als Programmierpraktikum (PP) für Informatik (INFO) und Einführendes Programmierpraktikum (EPP) für Mensch-Computer-Systeme (MCS) und Wirtschaftsinformatik (WINF) angeboten [1]. Dabei besitzt INFO mit 95 ECTS Informatik im Pflichtbereich den größten Informatikschwerpunkt, gefolgt von MCS mit 50 ECTS und WINF mit 30 ECTS. Die Studierenden haben im Vorfeld des Praktikums üblicherweise 20–25 ECTS in Kursen mit Programmierinhalt.

Der Arbeitsaufwand für die Aufgaben selbst soll in Summe in beiden Modulen jeweils 240 Stunden betragen. Die Abgabe erfolgt über das Würzburger Programmieraufgaben-Bewertungssystem (PABS) [ID17]. Die Einreichungen werden mit serverseitig in PABS ausgeführten, gewichteten JUnit-Tests bewertet und gelten als bestanden, wenn pro Aufgabe mindestens 90 % der Punkte erreicht wurden. Das Bestehen jeder Aufgabe ist notwendig, es dürfen dabei beliebig viele Versuche abgegeben und getestet werden.

Das Praktikum selbst ist in drei Phasen gliedert: (1) Die Einführungsaufgabe dient als Überprüfung, ob die für das Praktikum notwendigen Kenntnisse vorhanden sind. Die Bearbeitungszeit beträgt ca. sechs Tage. (2) Drei weitere (im EPP zwei weitere) Aufgaben müssen innerhalb von sechs Wochen nach Praktikumsstart bearbeitet und eingereicht werden. Die Aufgaben der Phase 2 haben gegenüber der Einführungsaufgabe größeren Umfang und höhere Komplexität. (3) Zur Überprüfung, ob die Aufgaben eigenständig bearbeitet wurden, muss zusätzlich eine etwa 90-minütige Abschlussklausur am PC bestanden werden.

Im analysierten Durchlauf haben nach insgesamt 60 930 Abgabeversuchen $n = 212$ Studierende (95 INFO, 46 MCS, 44 WINF) ausreichende Lösungen zur Einführungsaufgabe eingereicht. Davon haben 111 Studierende (52.4 %) auch alle Aufgaben der Phase 2 bestanden. Die Bestehensquote lag dabei in MCS deutlich höher (73.9 %) als in INFO (54.7 %) und WINF (47.7 %). Das Praktikum wird in jedem Semester angeboten und kann (außer in MCS) bei Misserfolg beliebig oft wiederholt werden. Eine numerische Note wird nicht vergeben.

[1] Das Praktikum kann noch in in neun weiteren Studiengängen belegt werden. Die Teilnehmerzahlen sind mit insgesamt 27 Absolventen allerdings jeweils geringfügig.

3 Studentische Einschätzung zur Codequalität

Im ersten Schritt erfolgt eine Bestandsaufnahme, welche Relevanz Studierende der Codequalität einräumen. Hierfür wurde nach Abschluss des Praktikums im Rahmen einer am Institut üblichen, freiwilligen anonymen Veranstaltungsevaluation eine Umfrage mit elf Fragen bezüglich der Einstellung zur Codequalität durchgeführt. Es liegen 88 ausgefüllte Evaluationsbögen vor. Nach eigener Aussage der Befragten wurden in 59 Fällen alle Aufgaben der Phase 2 erfolgreich bearbeitet. Der Anteil der Antworten nach Studiengang entspricht weitgehend dem Anteil des Studiengangs an der Grundgesamtheit [2]. Sofern nicht anders angegeben wird eine Likert-Skala (1: „Trifft voll und ganz zu“ ⇔ 4: „Trifft gar nicht zu“) verwendet.

Alle Antworten sind in Tabelle 1 dargstellt. Die erste Fragengruppe betrifft die Einstellung zur Codequalität, welche im Praktikum (Fragen 1-4) insgesamt als eher relevant und im Allgemeinen (Frage 5) sogar als sehr relevant eingestuft wird. Arbeitsweisen, die die Codequalität unterstützen, werden allerdings nur gelegentlich genutzt (Fragen 6 und 7, Skala hier abweichend: 1: „nie“ ⇔ 4: „oft“). Auch im Praktikum vorhandene Unterstützungsangebote, wie Angaben zum typischen Codeumfang in Klassendiagrammen der Angabe, werden als eher nicht hilfreich eingestuft (Fragen 8 und 9). Konsequenzen für das Praktikum werden kontrovers beurteilt (Fragen 10 und 11). Mehr Feedback zur Codequalität wird eher befürwortet, eine Aufnahme als Bewertungskriterium jedoch sehr kontrovers bewertet. Alle Antworten weisen allerdings eine hohe Varianz auf.

Ein Erfahrungswert bei der Betreuung des Praktikums ist, dass Rückmeldung zur Codequalität häufig interessierter angenommen wird, wenn die betroffene Person einen Studiengang mit hohem Informatikanteil studiert. Um zu testen, ob sich das in den gegebenen Antworten bemerkbar macht, wurden alle Antworten für jeden Studiengang in zwei Gruppen aufgeteilt (Teil des Studienganges vs. nicht Teil des Studienganges). Anschließend wurde mittels eines Welch-t-Tests untersucht, ob sich die Mittelwerte der Gruppen unterscheiden. Deutlich erkennbare Abweichungen ($p < 20\,\%$) sind ebenfalls in der Ergebnistabelle angegeben.

Derartige Abweichungen haben sich jedoch nur bei der Relevanz der Codequalität im Praktikum (Fragen 1-4) und der Einbindung der Codequalität in die Praktikumsbewertung (Frage 11) gezeigt. Die Richtung der Effekte entsprach den Erwartungen: Befragte aus der Gruppe INFO (WINF) messen der Codequalität im Praktikum tendenziell eine höhere (niedrigere) Relevanz bei als solche aus den übrigen Gruppen. MCS Studierende wichen hier nicht deutlich erkennbar von anderen Studiengängen ab. Eine Aufnahme der Codequalität als Bewertungskriterium wurde von Teilnehmenenden des PP stark befürwortet, von denen des EPP eher abgelehnt.

Insgesamt werten wir die Ergebnisse als Indikator, dass die Codequalität allgemein auch unter Studierenden als relevant erkannt wird. Im Kontext des Praktikums ist das vor allem im Bachelorstudiengang Informatik der Fall.

[2] Relative Abweichung der Zusammensetzung: INFO: −7.3 %, MCS: −0.2 %, WINFO: +5.4 %, Sonstige: +2.1 %

Frage	Ergebnis avg	stdev	Abweichung Studiengang: avg (p-Wert)
1: Die Hauptsache bei der Bearbeitung ist es, die PABS-Test grün zu bekommen.	1.80	0.90	WINF: 1.57 (14.3%)
2: Mir ist es wichtig, dass mein Code über das Bestehen der PABS-Test hinaus inhaltlich richtig ist.	1.94	1.02	WINF: 2.34 (5.1%) INFO: 1.73 (10.6%)
3: Mir ist es wichtig, dass ich über das Bestehen des Praktikums hinaus lerne gut zu programmieren.	1.62	0.96	WINF: 2.04 (3.3%) INFO: 1.36 (4.4%)
4: Mir ist es wichtig, dass mein Code nicht nur inhaltlich richtig, sondern auch gut lesbar ist.	1.94	0.96	WINF: 2.22 (14.0%) INFO: 1.73 (9.8%)
5: Programmieren ist einfacher, wenn man konsequent darauf achtet, guten Code zu schreiben	1.47	0.87	
6: Ich habe meinen eigenen Code refactored (= bereits inhaltlich richtigen Code überarbeitet, um ihn lesbarer zu machen oder besser zu strukturieren)	2.66	1.09	
7: Ich habe mir Codemetriken ausrechnen lassen, damit ich weiß, wie gut mein Code ist	1.26	0.86	MCS: 1.00 (1.6%)
8: Die LoC-Angaben in den Klassendiagrammen der Graphen-Aufgabe haben mir bei der Bearbeitung geholfen	2.85	1.27	WINF: 2.48 (15.8%)
9: In freieren Aufgaben meine eigene Struktur definieren zu können hilft mir, besseren Code zu schreiben	2.44	1.20	INFO: 2.16 (9.8%)
10: Ich würde mir mehr Rückmeldung zu meiner Codequalität wünschen	2.02	1.23	
11: Solange das Praktikum dadurch nicht schwieriger wird sollte auch die Qualität des Codes in die Bewertung mit einbezogen werden (statt nur inhaltliche Korrektheit zu bewerten).	2.54	1.26	WINF: 2.96 (3.7%) INFO: 1.21 (0.1%) MCS: 2.95 (12.7%)

Tab. 1: Ergebnisse der Umfrage zur Codequalität.

4 Codequalität der Einführungsaufgabe

Im nächsten Schritt folgt eine Analyse, ob besserer Code das Bestehen des Praktikums begünstigt. Hierfür wurden für alle ausreichenden Lösungen der Einführungsaufgabe insgesamt neun Qualitätsmetriken erhoben, die sich in drei Kategorien einteilen lassen.

Folgende drei Metriken wurden gewählt, da sie unserer Meinung nach übliche Metriken für die Qualität der Implementierung darstellen: (1) *Gesamtkomplexität (WMC)* definiert als Weighted Methods per Class nach [CK94]. Wir definieren eine Abgabe als besser, wenn sie die gleiche Funktionalität mit einer niedrigeren Gesamtkomplexität erbringt. (2) *Maximale Schachtelungstiefe (Nesting Depth)* von Blöcken unter allen Methoden der Abgabe. Eine Methode, die außer ihrem Rumpf keinen Block enthält, hat hier einen Wert von 0. Da geschachtelte Anweisungen schwieriger zu verstehen sind, ist ein niedrigerer Wert besser. (3) *Maximale Komplexität einer Methode* berechnet als das Maximum der zyklomatischen Komplexität nach McCabe [Mc76] aller Methoden.

Daneben wurden folgende Metriken erhoben, die als Indikatoren für die Strukturiertheit gelten können: (4) *Lines of Code (LoC)* der Abgabe, ohne Kommentare und Leerzeilen.

(5) *Maximaler Umfang einer Methode* unter allen Methoden in Codezeilen (vgl. LoC). (6) *Anzahl Methoden* in der Abgabe. Wir erwarten, dass sinnvoll strukturierte Abgaben viele, kurze Methoden benutzen und Code wiederverwenden, was zu weniger LoC führt.

Zusätzlich wurden folgende Metriken erhoben, die als Good bzw. Bad Smells im Code angesehen werden können [FB18]: (7) *Anzahl Literale* in der Abgabe. Viele Literale können auf fehlerträchtige Indextransformationen oder Magic Numbers im Code hinweisen. Wir gehen daher davon aus, dass bessere Abgaben weniger Literale beinhalten. (8) *Anzahl Typecasts* in der Abgabe. Die Aufgaben im Praktikum sind so gestellt, dass explizite Typecasts ausschließlich bei equals-Methoden notwendig sind. Viele Typecasts werten wir als Indikator, dass hier Lücken im Umgang mit generischer Programmierung und Vererbung vorhanden sind. (9) *Anzahl Lambda-Ausdrücke* in der Abgabe. Da funktionale Programmierung kein fester Teil des Curriculums ist, gehen wir davon aus, dass wer sich diese Programmiertechniken im Selbststudium beigebracht hat, besser abschneidet.

Metrik		Referenz-	Stud+		Stud−		Signifikanz
Nr.	Kürzel	lösung	avg	stdev	avg	stdev	(p-Wert)
1	WMC	66	127.41	12.73	133.15	17.79	0.86%
2	Nesting Depth	2	2.87	0.80	3.11	0.98	5.31%
3	Most Complex M.	5	13.83	3.53	14.50	3.94	9.06%
4	LoC	353	496.64	61.80	505.56	67.68	10.84%
5	Longest Method	12	38.98	15.05	37.99	9.64	43.41%
6	Methods	52	54.22	2.94	54.83	4.83	35.14%
7	Literals	41	87.80	25.20	93.37	22.58	0.72%
8	Typecasts	1	1.86	2.20	1.97	2.44	44.40%
9	Lamdas	14	1.37	3.30	1.30	3.05	48.67%

Tab. 2: Absolute Werte der Metriken und Einfluss auf den Praktikumserfolg.

Für die Erhebung wurden die Abgaben in zwei Gruppen aufgeteilt: Die Gruppe (Stud−) aller Studierenden, die zwar die Einführungsaufgabe bestanden haben, jedoch nicht alle folgenden Aufgaben ($n = 102$) sowie die Gruppe (Stud+) derjenigen, die alle Aufgaben bestanden haben ($n = 111$). Zusätzlich wurde eine bereits vorhandene Referenzlösung untersucht, die unserer Meinung nach die Anforderungen an eine qualitativ hochwertige Lösung erfüllt.

Die Ergebnisse sind in Tabelle 2 dargestellt. Erkennbar (und erwartbar) ist, dass die Referenzlösung in jeder Kategorie deutlich besser abschneidet als der Durchschnitt der Studierenden. Der Abstand fällt mit teilweise mehreren Standardabweichungen dennoch größer aus als von uns erwartet. Insbesondere bestehen Studierendenlösungen, unabhängig vom Praktikumserfolg, aus etwa einem Drittel mehr Codezeilen als die Referenzlösung und haben beinahe die doppelte Komplexität.

Nach Erhebung der Metriken wurde anschließend für jede Metrik analysiert, ob sie das Bestehen im Praktikum beeinflusst. Da Codemetriken üblicherweise nicht normalverteilt sind [Co07] und ein Shapiro-Wilk-Test für alle erhobenen Metriken eine Normalverteilung

Metrik		INFO		MCS		WINF	
Nr.	Kürzel	avg	p-Wert	avg	p-Wert	avg	p-Wert
1	WMC	127.62	1.12%	132.20	15.76%	133.48	4.82%
2	Nesting Depth	2.88	4.28%	2.89	34.44%	3.27	0.76%
3	Most Complex M.	13.56	0.10%	14.57	5.25%	15.57	2.26%
4	LoC	494.96	19.61%	495.22	19.92%	516.93	3.11%
5	Longest Method	37.49	8.29%	38.98	28.37%	41.86	14.75%
6	Methods	54.52	16.56%	53.80	44.00%	53.89	16.16%
7	Literals	87.82	20.02%	90.35	44.35%	95.45	12.83%
8	Typecasts	1.64	7.99%	2.07	34.44%	2.27	2.20%
9	Lamdas	2.03	0.10%	0.48	4.14%	0.48	3.14%

Tab. 3: Durchschnittliche Werte nach Studiengang und Signifikanz der Abweichungen.

widerlegte (jeweils $p < 1\,\%$) wurde der Wilcoxen-Rangsummentest (zum Niveau $p = 10\,\%$) verwendet. Die zugrundeliegende Nullhypothese besagt, dass die Werte der entsprechenden Metrik in den Gruppen gleich verteilt sind. Die Ergebnisse mit p-Werten sind ebenfalls in Tabelle 2 aufgeführt. Die drei klassischen Komplexitätsmetriken haben sich dabei als Indikator für den Praktikumserfolg bestätigt. Insbesondere die Hypothese, dass die Gesamtkomplexität der Abgabe unabhängig vom Praktikumserfolg ist, konnte verworfen werden ($p = 0.86\,\%$). Bei den anderen Metriken war das, mit Ausnahme der Literalanzahl, nicht der Fall.

Anschließend wurde analysiert, ob die Codequalität abhängig vom Studiengang ist. Hierfür wurde erneut mittels Wilcoxen-Test untersucht, ob sich die Verteilung der Codemetriken eines Studiengangs von denen anderer Studierender unterscheidet. Die Ergebnisse sind in Tabelle 3 dargestellt. Dabei erreichten INFO Studierende in allen Metriken bessere Ergebnisse. Diese Abweichung war in sechs Fällen signifikant. WINF Studierende hingegen erreichten in allen Metriken schlechtere Ergebnisse. Diese Abweichung war ebenfalls in sechs Fällen signifikant, insbesondere auch in den drei klassischen Komplexitätsmetriken. MCS Studierende zeigten bei mittelmäßigen Ergebnisen nur zwei signifikante Abweichungen.

Interessanterweise war die individuelle Bearbeitungszeit zwischen erstem Test und Lösungsabgabe ebenfalls stark vom Studiengang abhängig. So benötigen WINF Studierende mit etwa 85.25 Stunden durchschnittlich fast doppelt so viel Zeit für ihre Abgabe wie INFO (45.12) und MCS (46.94) Studierende.

Als weiterer Einflussfaktor wurde die Praktikumswiederholung analysiert. Dabei wurden 80 Personen mit durchschnittlich 2.75 (stdev 1.04) Praktikumsdurchläufen identifiziert, die mit exakt 50% auch eine leicht niedrigere Bestehensquote hatten als Studierende im Erstversuch. Erneut wurde getestet, ob sich die Verteilungen der Codemetriken tendenziell unterscheiden. Dies war jedoch nur bei zwei Metriken der Fall: Die längste Methode war im Durchschnitt um $11.23\,\%$ kürzer ($p = 0.37\,\%$) und es wurden durchschnittlich 0.54 Lambdas weniger pro Abgabe ($p = 1.58\,\%$) genutzt als in der Referenzgruppe. Die

durchschnittliche Bearbeitungszeit lag mit 60.52 Stunden jedoch signifikant ($p = 3.09\%$) höher, als die der Teilnehmenden im Erstversuch (48.43).

Weiterhin wurde analysiert, ob sich die Codequalität der Gruppe mit wiederholter Teilnahme im Vergleich zum letzten Versuch tendenziell verbessert hat. Da die absoluten Werte von der Aufgabe abhängen und nicht direkt vergleichbar sind, wurden für diese Analyse stattdessen die Verteilung der relativen Position ausgewertet. Mit Ausnahme der Literal- und Lambdaanzahl konnte eine leichte Verbesserung in allen Metriken festgestellt werden. Signifikant ist diese Änderung bei der Schachtelungstiefe ($p = 3.51\%$), der Komplexität der komplexesten Funktion ($p = 5.80\%$), Anzahl der Methoden (3.34%) und Typecasts ($p = 2.23\%$); allerdings auch bei der Anzahl der Lambdas ($p = 0.13\%$).

Zusammenfassend lässt sich sagen, dass gutes Abschneiden hinsichtlich der Metriken 1–3 tendenziell das Bestehen im Praktikum begünstigt. Die Referenzlösung war wesentlich besser als die der Studierenden. In Studiengängen mit höherer Affinität zur Informatik und umfangreicherer vorangegangener Programmierausbildung war die Codequalität tendenziell besser. Bei Praktikumswiederholung wird der Code im darauf folgenden Durchgang etwas besser und ist dann absolut kaum vom Durchschnitt zu unterscheiden. Die Bestehensquote liegt hier dennoch leicht unter dem Durchschnitt.

5 Verwandte Arbeiten

Einen aktuellen Überblick über den Stand automatischer Bewertungssysteme haben u.a. Ihantola et al. 2010 [Ih10] zusammengetragen. Dabei wurden in mehreren Arbeiten und mit unterschiedlichen Zielsetzungen auch Codemetriken berücksichtigt. Cardell-Oliver beschrieb 2011 [Ca11], wie sich Codemetriken nutzen lassen um das Lernverhalten von Studierenden zu analysieren und zu verbessern. Auch Pettit et al. 2015 [Pe15] betrachten dies mit Fokus auf die Veränderung in den Codemetriken zwischen verschiedenen Abgabeversuchen. Der Einfluss verschiedener Abgabemodelle wurde bereits 2005 von Malmi et al. [Ma05] untersucht.

6 Zusammenfassung und Ausblick

Die Auswertung des aktuellen Semesters hat gezeigt, dass auch Studierende eine hohe Codequalität als erstrebenswert ansehen, im Vergleich dennoch signifikant schlechtere Ergebnisse zeigen. Außerdem legen unsere Ergebnisse nahe, dass die Codequalität Einfluss auf den Praktikumserfolg hat. Ein erster Vergleich mit vorangegangenen Praktika hat zwar bereits eine Verallgemeinerbarkeit, aber auch den Einfluss von Organisationsstruktur und Aufgabe auf die Relevanz unterschiedlicher Metriken angedeutet. In unserer weiteren Arbeit wollen wir genauer analysieren, wie sich allgemeingültige Bestehensindikatoren ableiten lassen und welche Ziele diesbezüglich in der Programmierausbildung angesetzt

werden können. Auch soll untersucht werden, inwiefern unsere Politik der unbegrenzten Abgabeversuche Einfluss auf die Codequalität hat, weil sie evtl. Eigeninitiative hemmt, aber auch mehr Raum für Refactorings lässt.

Des Weiteren wollen wir mögliche Maßnahmen zur Verbesserung der studentischen Codequalität diskutieren. Dies könnte sowohl über zusätzliche Hilfsmittel wie Codereviews während der Bearbeitung oder einer Einbindung der Codequalität in die Bewertungskriterien erreicht werden. Auch dabei soll analysiert werden, welche Ansätze zur Umsetzung den gewünschten Effekt am besten erreichen.

Literatur

[Ca11] Cardell-Oliver, R.: How Can Software Metrics Help Novice Programmers? In: Proc. of the 13th Australasian Computing Education Conference. Bd. 114, Perth, Australia, S. 55–62, 2011.

[CK94] Chidamber, S. R.; Kemerer, C. F.: A metrics suite for object oriented design. IEEE Trans. on Software Engineering 20/6, S. 476–493, 1994.

[Co07] Concas, G.; Marchesi, M.; Pinna, S.; Serra, N.: Power-Laws in a Large Object-Oriented Software System. IEEE Trans. on Software Engineering 33/10, S. 687–708, Okt. 2007.

[FB18] Fowler, M.; Beck, K.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 2018, ISBN: 978-0-13-475759-9.

[ID17] Iffländer, L.; Dallmann, A.: Der Grader PABS. In: Automatisierte Bewertung in der Programmierausbildung. Bd. 6, Digitale Medien in der Hochschullehre, Waxmann Verlag, Kap. 15, S. 241–254, 2017.

[Ih10] Ihantola, P.; Ahoniemi, T.; Karavirta, V.; Seppälä, O.: Review of Recent Systems for Automatic Assessment of Programming Assignments. In: Proc. of the 10th Koli Calling Int. Conference on Computing Education Research. Koli, Finland, S. 86–93, 2010.

[Ma05] Malmi, L.; Karavirta, V.; Korhonen, A.; Nikander, J.: Experiences on Automatically Assessed Algorithm Simulation Exercises with Different Resubmission Policies. ACM Journal on Educational Resources in Computing 5/7, Sep. 2005.

[Mc76] McCabe, T. J.: A Complexity Measure. IEEE Trans. on Software Engineering SE-2/4, S. 308–320, 1976.

[Pe15] Pettit, R.; Homer, J.; Gee, R.; Mengel, S.; Starbuck, A.: An Empirical Study of Iterative Improvement in Programming Assignments. In: Proc. of the 46th ACM Technical Symposium on Computer Science Education. Kansas City, Missouri, USA, S. 410–415, 2015.