Informatikstudierende als Teamplayer. Wie die Integration von Teamarbeit in die Lehre gelingen kann

Anja Hawlitschek [1], Galina Rudolf[2] und Sebastian Zug[2]

Zusammenfassung: Die Integration von Teamarbeit in die Lehre ist aus vielerlei Gründen didaktisch wünschenswert. In der Praxis zeigt sich jedoch oftmals, dass effektive Teamarbeit ein sorgfältiges didaktisches Design voraussetzt. Nachdem wir bei der DELFI 2021 das (partielle) Scheitern der Integration von Teamarbeit in eine Online-Lehrveranstaltung zur Softwareentwicklung vorgestellt und Gründe dafür analysiert haben, wollen wir in diesem Beitrag die didaktische Weiterentwicklung der Lehrveranstaltung in den Fokus nehmen. Wir beschreiben die durchgeführten didaktischen Analysen und das resultierende didaktische Design in Bezug auf Gruppenbildung, Lernaktivitäten und Strukturierung der Teamarbeit sowie die Umsetzung. Ergebnisse einer Logfileanalyse zeigen, dass die Studierenden in der weiterentwickelten Lehrveranstaltung signifikant mehr Teamarbeit beim Programmieren zeigen, GitHub-Funktionen signifikant öfter nutzen und signifikant mehr bearbeitete Aufgaben einreichen. Auf der Basis unserer Ergebnisse ziehen wir Rückschlüsse zur Integration von Teamarbeit in Lehrveranstaltungen der Informatik.

Keywords: kooperatives Lernen, Collaboration Scripts, didaktisches Design

1 Einleitung

Die Integration von Teamarbeit in Lehrveranstaltungen der Informatik ist nicht nur aufgrund der Vorteile für Lernprozesse sinnvoll, sondern auch aufgrund der Relevanz von Teamfähigkeiten in IT-Unternehmen [Ga19]. Die Integration von Teamarbeit kann jedoch scheitern – dann ist die Arbeitslast ungleich verteilt oder Teams brechen auseinander. Ein solches Scheitern beschrieben wir in unserem Beitrag zur DELFI 2021 [HRZ21]. Unsere damaligen Analysen ergaben, dass unsere Studierenden wenig Erfahrungen in der Teamarbeit hatten, sich eine didaktische Unterstützung der Teamarbeit wünschten, zugleich Vorbehalte gegenüber Teamarbeit hatten und den Mehrwert nicht sahen. Im Anschluss an diese Auswertung entwickelten wir das didaktische Design des Kurses systematisch weiter und führten die angepasste Lehrveranstaltung 2021 erneut durch. Zielstellung war es, die Anzahl der gelösten Aufgaben in GitHub und insbesondere der gelösten Aufgaben im Team aber auch die Nutzung von GitHub-Funktionen (z.B. Branches, Issues, Pull Requests) zu erhöhen. Die systematische Weiterentwicklung des Kurses stellen wir in diesem Artikel vor. Mittels ANOVAs vergleichen wir die Teamarbeit in beiden Kursen

[1] Otto-von-Guericke-Universität, Zschokkestr. 32, 39104 Magdeburg, anja.hawlitschek@ovgu.de, https://orcid.org/0000-0001-8727-2364
[2] TU Bergakademie Freiberg, Institut für Informatik, Bernhard-von-Cotta-Straße 2, 09596 Freiberg, Galina.Rudolf@informatik.tu-freiberg.de & Sebastian.Zug@informatik.tu-freiberg.de

doi: 10.18420/delfi2022-019

(2020 und 2021), um Rückschlüsse auf den Erfolg unserer Intervention ziehen zu können. Wir fokussieren dabei auf Teamarbeit im Sinne des kooperativen Lernens [RT95].

2 Didaktisches Design

Bei der Weiterentwicklung des didaktischen Designs des Kurses folgten wir dem Vorgehensmodell von [HBS21]. Ausgehend von didaktischen Analysen zu Lehr-Lernzielen, Lernendencharakteristika und Rahmenbedingungen leiteten wir unter Rückbezug auf diesbezügliche empirische Studien didaktische Designentscheidungen zur Gruppenbildung, zu Lernaktivitäten und zur Strukturierung der Zusammenarbeit ab.

2.1 Didaktische Analysen

Die Lehr-Lernziele des Kurses unterscheiden sich zwischen Vorlesung und Übung. Während in der Vorlesung das Verstehen des Domänenwissens zur Programmierung in C# im Vordergrund steht, geht es in den Übungen um die Anwendung des Domänenwissens und der Programmierfertigkeiten. Die Studierenden sollen zudem Kenntnisse und Fertigkeiten erwerben, die für das Mikromanagement von Projekten in der Softwareentwicklung wichtig sind. Dabei steht die Verwendung von Versionsverwaltungswerkzeugen, um einen effizienten Programmierprozess in einem Team zu gewährleisten, im Vordergrund, aber auch übergeordnete Teamfähigkeiten wie die Koordination von Teamarbeit und die Kommunikation im Team. Auf diese Weise wollen wir die Studierenden auf Programmierprojekte in IT-Unternehmen vorbereiten.

Hinsichtlich der Lernendencharakteristika haben wir die Anzahl der Semester im Studiengang sowie die Vorerfahrungen der Studierenden berücksichtigt. Wie wir aus der Vorbefragung wissen (vgl. Kapitel 3), waren die Studierenden in unserem Kurs überwiegend im zweiten Semester (B.A.), mittelmäßig erfahren im Programmieren und unerfahren in der Arbeit mit GitHub.

Die Rahmenbedingungen stellten sich wie folgt dar: Der Kurs fand online statt, Moodle und GitHub waren verfügbar, kollaborative Prüfungen waren laut Prüfungsordnung möglich, wurden 2020 aber nicht angeboten. Die Rahmenbedingungen wurden durch die Covid19-Pandemie beeinflusst. Die Studierenden beider Kohorten mussten online interagieren, persönliche Treffen an der Universität waren nicht vorgesehen. Da kooperatives Online-Lernen einen höheren Koordinationsaufwand für die Teamarbeit mit sich bringt [Wi21], musste dies bei der Kursgestaltung berücksichtigt werden.

2.2 Entscheidungen zum didaktischen Design

Hinsichtlich der Gruppenbildung entschieden wir uns, die Gruppengröße von zwei Personen aus 2020 beizubehalten [HRZ21], um den Aufwand für die Gruppen-koordination nicht zu erhöhen. Die Ergebnisse der empirischen Forschung zur optimalen

Gruppierung von Studierenden in Programmierkursen sind uneinheitlich [HBS22]: Es lässt sich z.B. nicht eindeutig ableiten, ob es besser ist, Studierende mit heterogenen oder homogenen Programmierkenntnissen in Gruppen zusammenzufassen. Studien zur Wahl von Teampartner*innen weisen darauf hin, dass Studierende die Zusammenarbeit mit Freund*innen bevorzugen [HBS22]. Da unsere Studierenden im zweiten Semester sind, haben sie ggf. bereits Freund*innen unter ihren Kommiliton*innen gefunden. Aufgrund der Einschränkungen wegen der Covid19-Pandemie ist jedoch zu vermuten, dass dies nicht auf alle Studierende zutrifft. Deshalb haben wir entschieden, (1) die Studierenden selbst entscheiden zu lassen, mit wem sie zusammenarbeiten aber (2) die Gruppenarbeit erst nach einer Eingewöhnungsphase beginnen zu lassen. Auf diese Weise wird sichergestellt, dass die Studierenden bei Bedarf in den Online-Übungen noch Kontakte zu möglichen Teampartner*innen knüpfen können und ihnen zudem die Zeit gegeben, ohne den zusätzlichen Aufwand der Gruppenkoordination in den Kurs einzusteigen. Eine Selbstselektion von Teampartner*innen kann jedoch auch mit Problemen einhergehen [Fr17], zur Problemlösung standen die Lehrenden per Mail zur Verfügung.

Hinsichtlich der Lernaktivitäten fokussierten wir auf die Gestaltung der Aufgaben, auf den Mehrwert von Teamarbeit und auf die Gestaltung der Prüfungen. Aus der empirischen Forschung ist bekannt, dass Programmieraufgaben nicht zu leicht oder zu schnell zu lösen sein sollten, um die Teamarbeit nicht ineffizient zu machen bzw. erscheinen zu lassen [HBS22]. Die Studierenden 2020 begannen mit der Teamarbeit bereits in den ersten Aufgaben. Diese waren von den Lehrenden als einfache Einstiegsaufgaben gestaltet worden, um Grundlagen der Programmiersprache zu vermitteln. Diese Aufgaben wurden für 2021 als individuelle Lernaktivitäten konzipiert. Die Teamarbeit startet nun erst ab der dritten Aufgabe. Um den Mehrwert der Teamarbeit für die Studierenden zu verdeutlichen, wurden die damit verbundenen Lernziele sowie diesbezügliche didaktische Entscheidungen in der Einführungsvorlesung besprochen. Zudem wurden Schlüsselfaktoren für die erfolgreiche Softwareentwicklung sowie GitHub vertiefend vorgestellt[3]. Darüber hinaus war es unser Ziel, Prüfungen, Lernaktivitäten und Lernziele besser aufeinander abzustimmen (Constructive Alignment). Da die Studierenden lernen sollen, Programmierprojekte im Team durchzuführen, wurde neben der 2020 angebotenen individuellen, mündlichen Prüfung ein kollaboratives Programmierprojekt als Prüfungsleistung eingeführt.

Hinsichtlich der Strukturierung der Teamarbeit wird in der empirischen Forschung deutlich, dass Studierende mit wenig Vorerfahrung in der Teamarbeit von der Nutzung von Kollaborationsskripten, d.h. Anleitungen, wie die Teammitglieder miteinander interagieren und zusammenarbeiten sollen, profitieren. Ein Kollaborationsskript kann mit Fokus auf den Ablauf der Zusammenarbeit entworfen werden, aber Lernende auch dazu anzuleiten ihre Zusammenarbeit selbst zu organisieren und zu reflektieren [No13]. Im Vergleich zu nicht angeleiteter Teamarbeit können Skripte die domänenspezifischen Lernergebnisse, das Wissen über Prinzipien für erfolgreiches kollaboratives Lernen und die diesbezüglichen Kompetenzen verbessern [RS05]. Mit zunehmender Kompetenz der Lernenden wird empfohlen, deren Autonomie in der Zusammenarbeit zu erhöhen. Dies

[3] https://bit.ly/3ybdFLY , https://bit.ly/3N8EXqx

kann durch eine schrittweise Reduktion der Anleitung erreicht werden [WF11]. Für die Teamarbeit in unserer Lehrveranstaltung entwarfen wir ein Kollaborationsskript mit einem Fokus auf den Ablauf der Zusammenarbeit. Die Studierenden schlüpfen in die Rolle des Maintainers bzw. des Entwicklers. Anschließend werden sie strukturiert durch die Phasen der Zusammenarbeit geleitet, von der Initialisierung des Projekts in GitHub bis hin zum Deploy. Im Kollaborationsskript werden für Aufgabe 3 für jeden Schritt des Prozesses die Interaktionen zwischen den beiden Gruppenmitgliedern und die GitHub-Funktionen, die die Studierenden verwenden sollen, festgelegt. In den Aufgaben 4-7 war der Ablauf nur noch teilweise vorgegeben, in den Aufgaben 8-9 gar nicht mehr[4].

3 Studie

3.1 Vorgehen

In den Lehrveranstaltungen konnten von 29 Studierenden (2020) bzw. 34 Studierenden (2021) Logfiles und von 18 Studierenden (2020) bzw. 26 Studierenden (2021) Fragebogen erhoben werden. Zunächst wurde jeweils zu Beginn des Semesters eine Online-Vorbefragung durchgeführt. Dabei wurde neben den Basisdaten, wie Alter, Studiengang und Anzahl der Fachsemester, auch Vorkenntnisse und Jahre der Programmiererfahrung erhoben. Die Studierenden bewerteten ihr Vorwissen und ihre praktischen Fertigkeiten im Programmieren sowie ihre Erfahrungen mit Git auf einer Likert-Skala von 1 (sehr gering) bis 5 (sehr hoch). Mit sechs inhaltlichen Fragen wurde das domänenspezifische Vorwissen erfasst. Zur Analyse der Zusammenarbeit wurden Logfiles verwendet. Dafür wurden die Git-Repositories der Teams automatisch erfasst, anonymisiert und gefiltert. Die Umsetzung erfolgte auf der Grundlage des Python-Pakets github2pandas[5]. Aus den Logdateien wurde ermittelt, wie viele Aufgaben die Studierenden in GitHub insgesamt gelöst haben und wie viele Aufgaben in Teamarbeit gelöst wurden. Zu letzteren zählten wir alle Aufgaben, bei denen jede/r Partner*in mindestens eine Codeänderung beigetragen hat. Außerdem wurde erhoben, wie oft GitHub-Funktionen, wie Commits to Branches, Issues und Pull Requests, genutzt wurden und welche Abschlussnoten die Studierenden erreichten.

3.2 Ergebnisse

Die Studierenden im Kurs 2021 (Alter M: 20,5; SD: 2,6) befanden sich überwiegend im zweiten Semester eines IT-Bachelor-Studiengangs einer technischen Universität. Sie gaben im Durchschnitt an, über 3,2 Jahre (SD: 2,4) Programmiererfahrung zu verfügen. Sie schätzten ihre theoretischen Vorkenntnisse (M: 2,5, SD: 1,1) und ihre praktischen Fertigkeiten im Programmieren (M: 2,6; SD: 1,1) als eher mittelmäßig und ihre Vorerfahrungen mit Git (M: 1,7; SD: 0,8) als relativ gering ein. Die Selbsteinschätzung

[4] https://github.com/SebastianZug/Delfi_2022_Aufgabenblaetter
[5] https://pypi.org/project/github2pandas/

spiegelt sich in den Ergebnissen des Vorwissenstests. Von sechs möglichen Punkten erreichten die Studierenden im Durchschnitt 3,1 (SD: 1,6). Signifikante Mittelwertsunterschiede zwischen den Studierendengruppen 2020 und 2021 ergaben ANOVAs für keine der Variablen.

Mit ANOVAs berechneten wir Mittelwertunterschiede zwischen den Kohorten bzgl. der Anzahl der gelösten Aufgaben in GitHub, der Anzahl der Aufgaben in Teamarbeit, der Abschlussnoten und der Nutzung von GitHub-Funktionen. Bezüglich letzterer schlossen wir Aufgabe 3, bei der die Nutzung der GitHub-Funktionen im Kollaborationsskript vorgegeben war, aus der Analyse aus. Signifikante Unterschiede ergaben sich hinsichtlich der Anzahl der gelösten Aufgaben ($F(1,61) = 7{,}1$, $p = {,}01$, $Eta^2 = {,}11$), der Anzahl der Aufgaben in Teamarbeit ($F(1,61) = 16{,}7$, $p < {,}001$, $Eta^2 = {,}22$) und der Anzahl der Commits to Branches ($F(1,61) = 12{,}4$, $p < {,}001$, $Eta^2 = {,}17$) sowie der Pull Requests ($F(1,61) = 4{,}6$, $p = {,}03$, $Eta^2 = {,}07$). Die Studierenden im Kurs 2021 lösten mehr Aufgaben in GitHub (M_{2020}: 3,0, SD_{2020}: 2,2; M_{2021}: 4,5, SD_{2021}: 2,1) und im Team (M_{2020}: 1,5, SD_{2020}: 1,5; M_{2021}: 3,5, SD_{2021}: 2,2), nutzen mehr Commits to Branches (M_{2020}: 3,3, SD_{2020}: 3,0; M_{2021}: 9,5, SD_{2021}: 8,9) und Pull Requests (M_{2020}: 2,1, SD_{2020}: 4,0; M_{2021}: 4,6, SD_{2021}: 4,8). Auch die Abschlussnoten der Studierenden 2021 war im Durchschnitt besser (M_{2020}: 2,4, SD_{2020}: 1,1; M_{2021}: 2,0, SD_{2021}: 1,0), jedoch nicht signifikant ($F(1,41) = 1{,}4$, $p = {,}24$, $Eta^2 = {,}03$).

4 Fazit

Die Weiterentwicklung des didaktischen Designs der Lehrveranstaltung kann vor dem Hintergrund der beschriebenen Zielstellungen als Erfolg angesehen werden. Auch wenn von den erhobenen Parametern nicht direkt auf die tatsächliche Qualität der Teamarbeit geschlossen werden kann, ist die gestiegene Nutzung der GitHub-Funktionen und die gestiegene Zusammenarbeit in GitHub ein wichtiger Schritt in Richtung einer erfolgreichen Integration von Teamarbeit in die Lehrveranstaltung. Da wir mehrere Änderungen am didaktischen Design der Lehrveranstaltung vorgenommen haben, lässt sich keine Aussage dazu treffen, ob eine bestimmte Veränderung besonders wirksam war oder ob die Summe der Änderungen zum gewünschten Effekt führte. Die in Abschnitt 2 beschriebene Vorgehensweise bei der Integration von Teamarbeit eignet sich aus unserer Sicht als Handlungsempfehlung für Lehrende. Sie ermöglichte uns eine intensive Auseinandersetzung mit Teamarbeit in unserer Lehrveranstaltung sowie ein an empirischen Evidenzen ausgerichtetes didaktisches Design. Auf diese Weise konnten wir zugleich den Studierenden die Hintergründe von Entscheidungen zum didaktischen Design sehr transparent kommunizieren.

Eine Limitation der Studie ist das methodische Design. Zwar wurden die beiden Kohorten auf systematische Unterschiede bezüglich Variablen wie Vorwissen geprüft und die grundsätzlichen Rahmenbedingungen (Lehrende, Inhalte) nicht verändert, es lässt sich jedoch nicht ausschließen, dass es dennoch systematische Unterschiede zwischen den Gruppen im Vorfeld der Intervention gegeben haben könnte.

Literaturverzeichnis

[Ga19] Garousi, V.; Giray, G.; Tüzün, E.; Catal, C.; Felderer, M.: Aligning software engineering education with industrial needs: A meta-analysis. Journal of Systems and Software, 156, S. 65-83, 2019.

[Fr17] Freeman, S.; Theobald, R.; Crowe, A. J.; Wenderoth, M. P.: Likes attract: Students self-sort in a classroom by gender, demography, and academic characteristics. Active Learning in Higher Education, 18/2, S. 115-126, 2017.

[HBS21] Hawlitschek, A.; Berndt, S.; Schulz, S.: Towards a Framework of Planning Collaborative Learning Scenarios in Computer Science. In (Seppäla, O.; Petersen, A. Hrsg.): Proc. 21st Koli Calling International Conference on Computing Education Research, ACM, New York, S. 1-5, 2021.

[HBS22] Hawlitschek, A.; Berndt, S.; Schulz, S.: Empirical research on pair programming in higher education: a literature review, Computer Science Education, 2022.

[HRZ21] Hawlitschek, A.; Rudolf, G.; Zug, S.: Herausforderungen bei der Integration von Teamarbeit in die Lehre am Beispiel einer Lehrveranstaltung aus der Informatik. In (Kienle, A.; Harrer, A.; Haake, J.M.; Lingnau, A. Hrsg.): Die 19. Fachtagung Bildungstechnologien (DELFI). Lecture Notes in Informatics, Gesellschaft für Informatik, Bonn, S. 247-252, 2021.

[No13] Noroozi, O.; Teasley, S.; Biemans, H.J.A.; Weinberger, A.; Mulder, M.: Facilitating learning in multidisciplinary groups with transactive CSCL scripts. International Journal of Computer-Supported Collaborative Learning 8/2, S. 189–223, 2013.

[RS05] Rummel, N.; Spada, H.: Learning to collaborate: An instructional approach to promoting collaborative problem solving in computer-mediated settings. The Journal of the Learning Sciences 14/02, S. 201–241, 2005.

[RT95] Roschelle, J.; Teasley, S. D.: The construction of shared knowledge in collaborative problem solving. Proceedings of the NATO Advanced Research Workshop on Computer Supported Collaborative Learning, Springer, Berlin, Heidelberg, S. 69-97, 1995.

[WF11] Wecker, C.; Fischer, F.: From guided to self-regulated performance of domain-general skills: The role of peer monitoring during the fading of instructional scripts. Learning & Instruction, 21/6, S. 746–756, 2011.

[Wi21] Wildman, J. L.; Nguyen, D. M.; Duong, N. S.; Warren, C.: Student Teamwork During COVID-19: Challenges, Changes, and Consequences. Small Group Research 52/2, S. 119–134, 2021.