Individuelle Benotung von Teamprojekten: Lessons Learnt

Karsten Weicker[1]

Abstract: Teamprojekte werden individuell über ein Bewertungsschema benotet, in dem aus unterschiedlichen Quellen Punkte in den Bereichen Programmierung, Dokumente, Engagement und Präsentationen gesammelt werden. Zu den Quellen zählen neben der Dozentenbeobachtung auch die Analyse der Git-Repositories und Peer-Bewertungen. Auswertungen von 50 Projekten mit 405 Studierenden zeigen, dass sich die einzelnen Metriken gut ergänzen und so den gesamten Entwicklunsgprozess berücksichtigen können.

Keywords: Software-Entwicklungsprojekte; Benotung; VCS-Analyse; Peer-Bewertung

1 Einleitung

Software-Entwicklungsprojekte mit häufig auch großen Teams, d.h. > 6 Teilnehmer:innen, gehören heute zum Standardrepertoire des Informatik-Studiums, da so Anforderungen und Komplexität von Projekten aus dem Berufsalltag der Informatiker:in abbildbar sind. Dies geht zumeist mit organisatorischen und prüfungsrechtlichen Schwierigkeiten bei der Benotung der Projekte einher: die Teams sind zu groß bzw. die individuelle Notenfindung gestaltet sich schwierig (vgl. den beispielhaften Auszug aus einer Prüfungsordnung: "Dabei muss der als Prüfungsleistung zu bewertende Beitrag des Einzelnen als individuelle Prüfungsleistung deutlich abgrenzbar und zu bewerten sein. Die Gruppe sollte in der Regel nicht mehr als drei Studierende umfassen."[2]).

Da die praktizierten (und fachdidaktisch auch sinnvollen) Teamgrößen meist über der Empfehlung der Gruppengröße liegen, muss die Bewertung der individuellen Anteile mit besonderer Sorgfalt durchgeführt werden. Typische Strategien hierfür sind:

1. Verzicht auf eine Note zugunsten eines Bestanden/Nicht-Bestanden auf Basis von Minimalkriterien,
2. Vergabe einer Gruppennote mit Abweichungen nach oben und unten auf Basis einer Peer-Bewertung oder Verhandlung der Peers,
3. Vergabe einer Gruppennote mit Abweichungen nach oben und unten aufgrund von Einschätzungen durch die Dozent:in und

[1] HTWK Leipzig, Fakultät Informatik und Medien, Gustav-Freytag-Str. 42A, 04277 Leipzig, karsten.weicker@htwk-leipzig.de

[2] Prüfungsordnung der Fakultät für Mathematik und Informatik für den Studiengang Informatik, Friedrich-Schiller-Universität Jena, https://www4.uni-jena.de/unijenamedia/-p-43470.pdf

doi:10.18420/SEUH2022_08

4. Ermittlung der Note als Akkumulation mehrerer (evtl. gewichteter) Noten für einzelne Artefakte.

Jede Strategie birgt Herausforderungen und Schwierigkeiten: (1) würdigt nicht den hohen Aufwand der Studierenden und wirkt tendenziell demotivierend, (2) kann prüfungsrechtlich kritisch sein, da die Prüferrolle eindeutig zugeordnet ist, (3) erschwert ggf. der Prüfer:in die Argumentation, dass die Leistung einzelner trotz eines guten Projekts nicht bestanden ist, und (4) kann leicht als eine Sequenz von Hausaufgaben verstanden werden, was den eigentlichen zu benotenden Projekt- und Teamkompetenzen nicht gerecht wird.

In den an der HTWK Leipzig in den Informatik- und Medieninformatikstudiengängen durchgeführten Software-Entwicklungsprojekten, die zwei Semester (3. und 4. Fachsemester) dauern, wird seit 2014 ein Bewertungsschema angewandt, das über ein Punktesystem die Beobachtungen der Dozent:innen und die Peer-Bewertung mit Metriken aus dem Software-Entwicklungsprozess verknüpft [We16]. Dabei werden die Noten ausschließlich individuell für jede Student:in ermittelt – es geht im Gegensatz zu den oben angeführten Strategien (2) und (3) keine Note für das gesamte Projekt in die individuelle Benotung ein.

Das Konzept folgt dabei der zentralen Idee, über verschiedene Metriken/Teilaspekte den unterschiedlichen Facetten einer Gruppenarbeit (individuelle Programmierleistung vs. sozialen Kompetenzen) gerecht zu werden und es insbesondere den "Trittbrettfahren" schwer zu machen, mit wenigen optimierten Aktionen oder gutem Selbstdarstellungsvermögen die Note wesentlich zu beeinflussen.

Ähnliche Ansätze bezüglich der (halb-)automatischen Analyse einzelner Metriken bei der Benotung von Projekten können in der Arbeit von Coppit [Co06] bzgl. der Arbeitspakete in einem Ticketsystem und Buffardi [Bu20] bzgl. der Analyse der Versionskontrolle gefunden werden. Eine gewichtete Aggregation mehrerer Aspekte in der Form von individuellen und in der Gruppe erstellten Artefakten wurde von Hummel [Hu13] vorgeschlagen, die allerdings stark an dokumentenzentrierten Vorgehensmodellen ausgerichtet ist.

Nach der Benotung von sechs Jahrgängen an der HTWK liegen Daten aus insgesamt 50 Projekten mit 405 Bachelorstudent:innen vor, die das Projekt bis zum Ende (Abgabe des individuellen Abschlussberichts) durchgeführt haben. Auf Basis dieser Daten wird das Bewertungsschema genauer hinsichtlich seiner Anwendbarkeit bewertet.

2 Konzept

Der Benotung unterliegt ein Punkteschema, in dem 120 Punkte (plus 5 Zusatzpunkte) erreichbar sind. Ab 48 Punkten (40%) ist die Prüfung bestanden und die weiteren Noten werden gemäß einer linearen Skala berechnet.

Die Dozenten nehmen während des Projekts an verschiedenen Meetings der Teams teil (Produktvisions- und Technologieworkshop, Sprintplanning, Sprintreview und Retrospektive) und machen sich Notizen zur Beteiligung der Teammitglieder. Zudem werden pro Team drei Code-Walkthroughs durchgeführt, an denen die Teammitglieder ihre Features in der Software wie auch im Quelltext erläutern. Zu diesen Beobachtungen kommen individuelle Zuarbeiten für die Produktvision und die Technologierecherche, die Dokumentation eines Sprints, ein individueller Abschlussbericht sowie die Daten, die in der Versionsverwaltung (hier: git) sowie dem Issue-Tracking-System (Redmine, Jira bzw. YouTrack) erhoben wurden, und eine abschließende Peer-Bewertung.

Für die verschiedenen Bewertungsaspekte Programmierung, Dokumente, Engagement und Präsentationen werden Punkte aus mehreren Teilaspekten zusammengetragen. Die zuletzt angewandte Version steht in Tabelle 1. Details zu den einzelnen Metriken können [We16] entnommen werden.

Bei der Peer-Bewertung wird das relativ simple Schema aus [Hu13] benutzt, in dem die Teammitglieder eine feste Anzahl an unteilbaren Punkten auf die Teammitglieder verteilen müssen.

Das gesamte Schema ist über die vergangenen Jahre weitestgehend stabil geblieben und nur kleinere Anpassungen wurden vorgenommen.

So wurde bezüglich der Quelltextkommentare in [We16] nur das reine Enthaltensein von Kommentaren bewertet, was der heute flächendeckend üblichen Beschreibung von Schnittstellen beispielsweise in JavaDoc-Kommentaren nicht gerecht wird. Daher wurde seit 2019 in der Bewertung auf die laut [AR09] in Open-Source-Projekten durchschnittlichen 19% Kommentaranteil abgezielt und stärkere Abweichungen nach oben und unten führen zu weniger Punkten.

Im Bereich des Engagement wurde anfangs die Anzahl der Tage mit Git-Aktivität als eine Metrik genutzt, was rückblickend nicht die beste Lösung ist, da diese Metrik auch innerhalb von acht Wochen auf einen hohen Wert “gepuscht” werden kann. Daher wird seit 2018 mit der Anzahl der Wochen mit Git-Aktivität gearbeitet. Dabei wird in Wochen mit Commits/Pushes und Wochen mit geringerer Aktivität (Pull) unterschieden – letztere zählen nur als 0,5.

In diesem Paper betrachten wir ausschließlich die Benotung der Bachelorstudent:innen. Die Projekte werden jeweils zusätzlich durch 2–3 Masterstudent:innen begleitet, die diese Leistung im Rahmen einer Lehrveranstaltung Projektmanagementpraktikum absolvieren [WW09].

	Metrik	Regel für Punktevergabe	Maximum
Programmierung			**55**
	Git: Anzahl der Commits	1 Punkt pro 5 Commits	5
	Git: $LOC_+ - LOC_-$	1 Punkt für 100 LOC	10
	Git: blamed LOC	1 Punkt für 50 LOC im Endprodukt	10
	Git: Kommentare	14–24%= 5 Punkte, ab 49% 0 Punkte, linear skaliert	5
	verbuchte Arbeitszeit Programmierung	1 Punkt für 7 Stunden	10
	inspizierte Dateien	für drei Dateien berechnen sich anhand dem Anteil der Code-Ownership und dergröße der Datei bis zu 10 Punkte pro Datei	30
Dokumente			**30**
	Zuarbeit Produktvision	0–6 Punkte gemäß Umfang und Qualität	6
	Zuarbeit Technologierecherche	0–6 Punkte gemäß Umfang und Qualität	6
	Dokumentation eines Sprints	0–16 Punkte gemäß Umfang und Qualität	16
	Abschlussbericht	0–5 Punkte gemäß formaler Kriterien	5
Engagement			**30**
	Gesamtarbeitszeit	1 Punkt pro 20 Stunden	8
	Wochen mit Git-Aktivität	0,5 Punkte pro Woche, wobei Wochen mit geringer Aktivität nur halb zählen	8
	Beiträge in Meetings	aus der Anzahl der Meetings mit substantiellen Redebeiträgen und der Gesamtanzahl der Beiträge werden bis zu 16 Punkte vergeben	16
	Peer-Bewertung	durchschnittliche Bewertung: 5 Punkte, ohne Anerkennung der Peers: -5 Punkte, hohe Anerkennung: bis zu 10 Punkten, linear skaliert	10
Präsentationen			**10**
	Code-Walkthrougs und Anfangsworkshops	wird aus Qualität und Länge des Beitrags errechnet	10
			125

Tab. 1: Übersicht über das Bewertungsschema (aktuelle Version).

3 Erfahrungen

Bei jährlich 7–9 Projekten mit insgesamt etwa 70–85 Teilnehmern fällt ein moderater Betreuungsaufwand während der zwei Semester an. Die Meetings und das begleitende Seminar für die Masterstudent:innen kann im Rahmen von insgesamt 8 SWS bewältigt werden. Hierfür wurden im Stundenplan pro Woche 4 SWS reserviert und zwei Lehrkräfte teilen sich die Termine auf. Dabei fällt fast ausschließlich Dokumentationsaufwand an. Kundengespräche wurden nicht in diesen reservierten Zeiten durchgeführt, da sich die Kunden aus allen Teilen der Hochschule wie externen Organisationen und Firmen zusammengesetzt haben.

Die eigentliche Benotung findet im Anschluss an die Projekte nach der Abgabe der

Abschlussberichte statt. Durch die Möglichkeit der automatisierbaren Auswertung von Projektdaten kann für die eigentliche Bewertung ein Aufwand von etwa 25–30 Minuten pro Student:in gerechnet werden. So ergibt sich bei 70–85 Prüflingen ein Gesamtzeitaufwand zwischen 30 und 43 Stunden.

Insbesondere bei der Analyse der Versionsverwaltung git können viele Aufgaben automatisiert durchgeführt werden. So werden die Metriken beispielsweise auch durch das Werkzeug gitinspector berechnet. Alternativ könnte auch, wie in [Bu20] geschildert, das Log von git mit eigenen Skripten bearbeitet werden, wobei für die Ermittelung der Urheberschaft von Codezeilen auf diesem Weg mehr Aufwand einzuplanen ist. Wesentlich tiefgreifendere Analysen des Git-Repositories sind denkbar, wie sie beispielsweise mit der Graphdatenbank Neo4J und der Software jQAssistant in [Ma17] gezeigt werden – die Techniken stammen dabei aus dem Buch [To15].

Die Verwendung von Werkzeugen zur Analyse des Git-Repositories birgt einige Schwierigkeiten, die hier kurz adressiert werden sollen:

- In zwei von 50 Projekten brach der gitinspector mit Fehlermeldungen ab. Vermutlich wurde in diesen Projekten Cherry-Picking bei der Übernahme von Commits genutzt, was zu den Problemen bei der Analyse führte. In diesen Fällen wurden die entsprechenden Daten manuell mit noch moderatem Mehraufwand erhoben.
- Bei der Verwendung von Frameworks müssen oft initiale, nicht selbst programmierte sehr große Dateien eingebunden werden. Diese sind jedoch leicht zu erkennen und werden dann im Weiteren nicht berücksichtigt.
- Am Ende zählen nur Programmierleistungen, die es in das Endprodukt geschafft haben. Diese müssen im entsprechend main- oder dev-Branch vorliegen. In Einzelfällen wurden große Programmierleistungen nicht berücksichtigt, weil sie in einem Feature-Branch verblieben sind.
- Für den vorherigen Anstrich wie auch für programmierte Dateien, die im Laufe der Entwicklung als Legacy-Code wieder entfernt wurden, wird den Projekten empfohlen, diese Dateien in einem zusätzlichen Ordner *removedcode* im Repository zu sichern, so dass dieser gegebenenfalls mit geringerer Wichtung berücksichtigt werden kann.
- Die Projekte zeichnen sich oft durch eine große Vielfalt an Programmiersprachen aus. Auch dies muss gegebenenfalls bei der Konfiguration der automatisierten Analyse berücksichtigt werden. Für neuere Sprachen wie Kotlin oder Rust werden die Dateiendungen ggf. noch nicht erkannt und die Identifikation von Kommentaren im Quelltext wird z.T. nicht automatisiert unterstützt.

Abbildung 1 zeigt, wie sich die Notenverteilung über die Jahre hinweg verändert hat. Dabei werden zwei Trends deutlich: Der Anteil an Noten “ungenügend” ging zurück und in den Anfangsjahren haben die sehr guten und guten Noten massiv zugenommen. Letztere haben sich dann weitestgehend stabil eingepegelt.

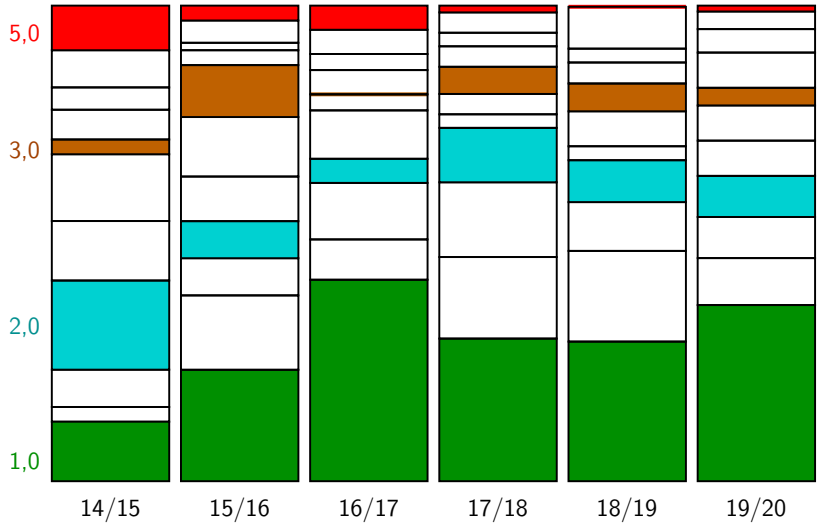

Abb. 1: Verteilung der Noten über die Jahrgänge.

Benotungsbereich	Mittelwert±sdv
Programmierung	43.55±13.65
Dokumente	23.44±5.83
Engagement	21.69±8.22
Präsentationen	6.95±2.86

Tab. 2: Durchschnittliche Punktzahl und Standardabweichung in den Bewertungskategorien (über alle 405 Studierende).

Bezüglich der zurückgehenden Durchfaller muss an dieser Stelle angemerkt werden, dass hier nur diejenigen Kandidat:innen erfasst sind, die bis zum Ende des Projekts aktiv waren und einen Abschlussbericht abgegeben haben. Während im Jahrgang 14/15 vermutlich noch viele ausprobieren wollten, ob es trotz geringer eigener Aktivität nicht doch zum Bestehen reicht, ist in den vergangenen Jahren zunehmend einen gezielten Abbruch des Projekts zu beobachten – oft verknüpft mit einem engagierten neuen Versuch im Folgejahr.

Die große Anzahl der sehr guten und guten Noten spiegelt unter anderem die zunehmende Akzeptanz des agilen Vorgehensmodells bei den Studierenden wider. Die projektleitenden Masterstudent:innen kennen dies zumeist aus eigener Werktätigkeit oder Berufspraktika und können so die Bachelorstudent:innen gut auf den Prozess einstellen.

Der Beitrag der einzelnen Bereiche im Benotungsschema zur Gesamtpunktzahl ist in Tabelle 2 aufgeschlüsselt. Bei der Programmierung und den Dokumenten werden durchschnittlich etwas mehr als drei Viertel der möglichen Punkte erreicht. Das Engagement und die Präsentationen scheinen mit etwa 70% der Punkte geringfügig höhere Hürden darzustellen.

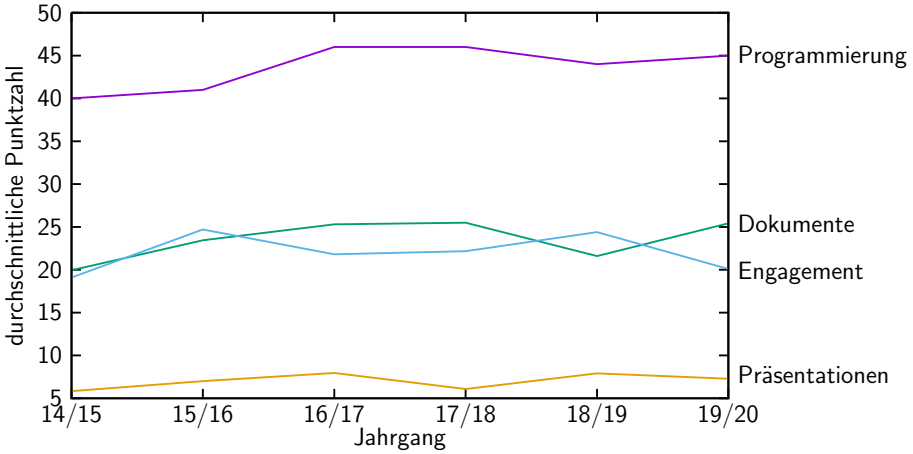

Abb. 2: Entwicklung der durchschnittlich erzielten Punkte in den einzelnen Bereichen.

Wenn man zu den einzelnen Bewertungsaspekten die Entwicklung in Abbildung 2 betrachtet, kann als Trend ebenfalls die Verbesserung der Programmierleistung beobachtet werden. Im Engagement und bei den Dokumenten zeigen die verschiedenen Jahrgänge unterschiedliche Stärken und Schwächen.

4 Reicht weniger Aufwand bei der Benotung?

Die Bewertung mit ihren vielen unterschiedlichen Kriterien ist ein aufwändiger Prozess für den Prüfer. Daher soll hier abschließend untersucht werden, ob man nicht mit weniger Kriterien und Metriken zu ähnlichen Entscheidungen bei der Notenfindung gelangen kann.

Hierfür wird in einer ersten Betrachtung die vergebene Note als “Wahrheit” aufgefasst und es wird errechnet, wie stark die Bereiche und die Metriken mit dieser Wahrheit korrelieren. Tabelle 3 enthält die Ergebnisse. Bei den einzelnen Bereichen zeigt sich deutlich, dass die höchste Korrelation zu den im Bereich der Programmierung erzielten Punkte besteht. Dies ist einerseits dadurch erklärbar, dass es sich um 55 von 125 möglichen Punkten handelt, und andererseits ist der Quelltext das Hauptartefakt der Projekte. Theoretisch könnten daher ähnliche Ergebnisse erwartet werden, wenn nur die Programmierleistung herangezogen wird. Allerdings liegt hier die Vermutung nahe, dass ohne die regulierenden Aspekte des Engagements und der Präsentationen viele Studierenden den Fokus von der Teamleistung zum reinen Produzieren von viel Code verschieben. Grundsätzlich wäre eine abgespeckte Variante der Bewertung denkbar, die lediglich die Programmierung und das Engagement berücksichtigt.

Die schwachen Korrelationen bei den Präsentationen würde ich mit der evtl. zu geringen Berücksichtigung (10 von 125 Punkten) der Codewalkthroughs bei der Notenfindung

erklären. Die Dokumente sind objektiv betrachtet für die schwächeren Studierenden eine Möglichkeit, Defizite in der Programmierkompetenz zu einem geringen Grad auszugleichen.

Die vier ausgewählten Einzelmetriken in Tabelle 3 zeigen eine moderate Korrelation mit der Note, was letztendlich allerdings auch unterstreicht, dass hier tatsächlich unterschiedliche Aspekte ermittelt werden. Dies wird belegt durch die in Tabelle 4 aufgezeigten Korrelationen zwischen den vier betrachteten Metriken. Die Werte sind insgesamt als geringe Korrelationen einzuordnen. Lediglich die Peer-Bewertung und die Anzahl der Wochen mit Git-Aktivität zeigen eine moderate Korrelation.

Zum Abschluss soll noch ein Vergleich zur Strategie (2) aus der Einleitung gezogen werden – der Vergabe einer Gesamtnote für das Projekt, welche dann für die einzelnen Teammitglieder entweder durch die Dozent:in auf Basis einer Peer-Bewertung angepasst wird oder direkt durch die Peers ausgehandelt wird (unter der Prämisse, dass die Durchschnittsnote gleich bleibt).

Wenn die Peer-Bewertung innerhalb jedes Teams die anderen Aspekte der Bewertung widerspiegeln würde, dann wäre über die Teammitglieder eines Teams eine hohe Korrelation zwischen der individuellen Note und der Einordnung in der Peer-Bewertung zu beobachten. Die Korrelation wurde für alle 50 Teams bestimmt und in Abbildung 3 zur Durchschnittsnote in Bezug gesetzt.

Mehr als 80% der Projekte zeigen dabei eine moderate bis hohe Korrelation zwischen der Note und der Peer-Bewertung, was anzeigt, dass dieses Bewertungsschema grundsätzlich zu ähnlichen Ergebnissen führen kann.

Ferner fällt auf, dass die kleineren Korrelationswerte nahezu ausschließlich bei Projekten mit sehr guten Durchschnittsnoten auftreten – mit einem prominenten Ausreißer bei Note 2,92. Zur Analyse der Gründe bei den Projekten im besseren Notenbereich, wurden für drei Projekte exemplarisch die Einzelbewertungen untersucht und zwei Auffälligkeiten als Gründe identifiziert:

Bereich bzw. Metrik	Korrelation
Punktzahl Programmierung	-0.9056
Punktzahl Dokumente	-0.5815
Punktzahl Engagement	-0.8198
Punktzahl Präsentationen	-0.5519
Punkte für blamed LOC	-0.7398
Punkte der Dateiinspektion	-0.7291
Wochen mit Git-Aktivität	-0.6932
Peer-Bewertung	-0.7312

Tab. 3: Korrelation der Note mit den Bewertungsbereichen und einigen ausgewählten Einzelmetriken (über alle 405 Studierende bzw. 219 Studierende bei der Git-Aktivität)

	blamed LOC	Dateiinspektion	Git-Wochen
Dateiinspektion	0.5994		
Git-Wochen	0.4011	0.5408	
Peer-Bewertung	0.4380	0.5207	0.6992

Tab. 4: Korrelation der einzelnen Metriken über der Datenbasis aller Studierender.

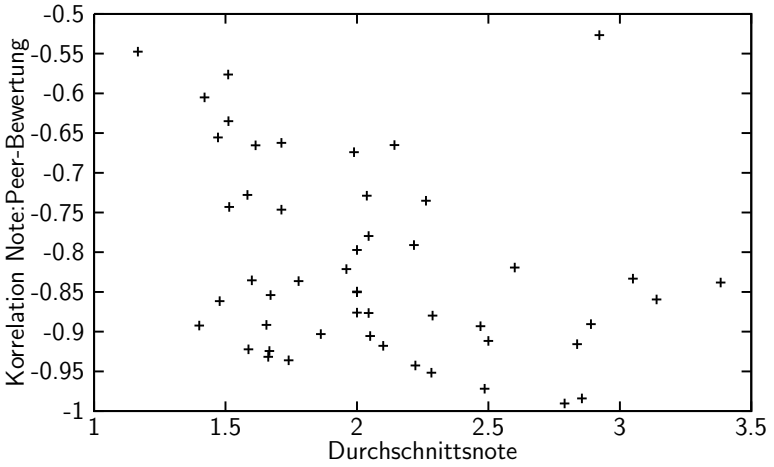

Abb. 3: Vergleich der Durchschnittsnote aller 50 Projekte mit der Korrelation der Peerbewertung im Projekt.

- Eine Einzelgänger:in trägt zwar umfassend zum Projekt bei (volle Punktzahl in der Programmierung), arbeitet aber an einem eher isolierten Teil der Software, was sich in nur wenigen Wochen mit Git-Aktivität und geringer Peer-Bewertung zeigt. Der geringeren Punktzahl im Bereich Engagement stehen allerdings volle Punktzahlen in allen Bereichen gegenüber, was in der Summe in einer sehr guten Note resultiert. (Beobachtet in zwei der Projekte).
- Die teamförderlichen Kompetenzen einer Student:in werden zwar von den Peers anerkannt, aber nicht in den Aspekten Programmierung, Dokumente und Präsentationen reflektiert. Dadurch wird nicht die Bestnote erreicht.

Bei den Projekten mit einer sehr guten Durchschnittsnote liegen häufig die Einzelnoten sehr dicht beieinander, was bei abweichender Peer-Bewertung auch zu schlechteren Korrelationswerten führt.

Der Ausreißer bei Note 2,92 entspricht einem Projekt mit vielen Schwierigkeiten. Die Mehrzahl der Student:innen hat nur schwer in die eigentliche Programmierung hineingefunden und eine Stundent:in hat sich als Hauptprogrammierer:in etabliert. Zudem ist die Peer-Bewertung stark durch Grüppchenbildung beeinflusst.

	1.0	1.3	1.7	2.0	2.3	2.7	3.0	3.3	3.7	4.0	5.0
beste Note	44	1	3	1	1						
schlechteste Note				3	4	2	4	9	7	10	11

Tab. 5: Verteilung der Projekte bezüglich der besten und der schlechtesten Note für die Teammitglieder.

Diese Beispiele in der Projektanalyse zeigen, dass die Peer-Bewertung gewisse Aspekte nicht abbilden kann, die letztendlich nur durch Dozentenbeobachtungen oder die Analyse des Git-Repositories ergänzt werden können.

In Bewertungsschemata mit einer gemeinsamen Projektnote und der Anpassung der Einzelnoten aufgrund einer Peer-Beurteilung neigen Projektteams manchmal zu Quick-and-Dirty-Hacks, da der Fokus auf dem Produkt liegt. Abweichungen der Einzelnoten bewegen sich häufig im Rahmen ±1 Note. Demgegenüber werden die individuellen Noten mit dem hier vorgestellten Schema breiter gestreut, wie die besten und schlechtesten Noten der Projekte in Tabelle 5 zeigt. während in 90% der Projekte mindestens eine sehr gute Individualnote vergeben wurde, gab es in 11 Projekten durchgefallene Student:innen und in 17 Projekten als schlechteste Note eine nur ausreichende Bewertung.

5 Fazit und Ausblick

Die große Datenbasis aus sechs Jahrgängen zweisemestriger Software-Entwicklungsprojekte erlaubt tiefe Einblicke in die Gestaltung und Auswirkung des gewählten Bewertungsschemas. Aufwand und Ergebnisse haben einen stabilen Zustand erreicht, der demonstriert, dass diese Art der Bewertung routiniert durchgeführt werden kann. Dennoch besitzt es eine Modularität, die leicht auf andere Begebenheiten angepasst werden kann, wie auch das vorgestellte Schema immer wieder in Einzelmetriken leicht modifiziert wurde.

Ist ein Bewertungsschema den Prüflingen transparent, lenkt es deren Fokus – wie in der Physik beeinflusst auch hier der Beobachtungsprozess das Experiment. So wird beim vorgestellten Bewertungsschema den Student:innen klar vermittelt, dass der Prozess der Software-Entwicklung im Team benotet wird. Konsequenterweise können einzelne Beobachtungen in Bewertungen mit einem anderen Fokus gegebenenfalls nicht reproduziert werden.

Die Betrachtung der Einzelmetriken zeigt, dass die in den Projekten durchgeführte reine Peer-Bewertung das Gesamtpaket der Metriken nicht ersetzen kann. Es werden jeweils unterschiedliche, eher nicht korrelierte Aspekte gemessen. Während in diesem Paper dies im Rahmen eines komplexen Bewertungsschema gezeigt wurde, kam Buffardi [Bu20] mit einem orthogonalen Ansatz zum selben Resultat. Er hat innerhalb einer Peer-Bewertung mit der CATME-Technik [Oh12] festgestellt, dass Analysen des Git-Repositories die Peer-Bewertung anreichern können.

Literaturverzeichnis

[AR09] Arafat, Oliver; Riehle, Dirk: The comment density of open source software code. In: 31st International Conference on Software Engineering, ICSE 2009, Companion Volume. IEEE, S. 195–198, 2009.

[Bu20] Buffardi, Kevin: Assessing Individual Contributions to Software Engineering Projects with Git Logs and User Stories. In: The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20). ACM, New York, S. 650–656, 2020.

[Co06] Coppit, David: Implementing Large Projects in Software Engineering Courses. Computer Science Education, 16(1):53–73, 2006.

[Hu13] Hummel, Oliver: Transparente Bewertung von Softwaretechnik-Projekten in der Hochschullehre. In (Spiller, A.; Lichter, H., Hrsg.): Tagungsband 13. Workshop SEUH Software Engineering im Unterricht der Hochschulen. S. 103–114, 2013. http://ceur-ws.org/Vol-956/.

[Ma17] Mahler, Dirk: Shadows Of The Past: Analysis Of Git Repositories. https://jqassistant.org/shadows-of-the-past-analysis-of-git-repositories/, 2017.

[Oh12] Ohland, Matthew W.; Loughry, Misty L.; Carter, Rufus L.; Bullard, Lisa G.; Felder, Richard; Finelli, Cynthia J.; Layton, Richard; Schmucker, Douglas G.: The Comprehensive Assessment of Team Member Effectiveness: Development of a Behaviorally Anchored Rating Scale for Self- and Peer Evaluation. Academy of Management Learning & Education, 11(4):609–630, 2012.

[To15] Tornhill, Adam: Your Code As a Crime Scene. O'Reilly, Sebastopol, CA, 2015.

[We16] Weicker, Karsten: A Metric-Based Point System for Grading Individual Performance in Software Engineering Projects. In (Hagel, Georg; Mottok, Jürgen, Hrsg.): European Conference on Software Engineering Education ECSEE. Shaker Verlag, S. 231–244, 2016.

[WW09] Weicker, Karsten; Weicker, Nicole: Von Häuptlingen und Indianern – Bachelor/Master als Chance. In (Jaeger, Ulrike; Schneider, Kurt, Hrsg.): Software Engineering im Unterricht der Hochschulen SEUH 11. dpunkt.verlag, Heidelberg, S. 61–743, 2009.