Evaluating Commit, Issue and Product Quality in Team Software Development Projects

Christopher Hundhausen
School of EECS
Washington State University
Pullman, WA USA
hundhaus@wsu.edu

Adam Carter
Department of Computer Science
Humboldt State University
Arcata, CA USA
adam.carter@humboldt.edu

Phillip Conrad
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA USA
phtcon@ucsb.edu

Ahsun Tariq
School of EECS, Washington State University
Pullman, WA USA
ahsun.tariq@wsu.edu

Olusola Adesope
Department of Educational Psychology, Washington State University
Pullman, WA USA
olusola.adesope@wsu.edu

ABSTRACT

Providing students with authentic software development experiences is essential to preparing them for careers in industry. To that end, many undergraduate courses include a team-based software development experience in which each team works on a *different* software project. This raises significant challenges for assessing student work and measuring the impact of pedagogical interventions: What do we measure and how, when each team is working on a different project? To address this question, we present a collection of metrics developed using the Goal-Question-Metric framework from the empirical software engineering literature, and an empirical study in which we applied those metrics to assess 23 team software projects involving 94 students at three institutions. Study results suggest that these metrics, which gauge commit, issue, and overall product quality, are sensitive to differences in the quality of teams' processes and products. This work contributes a new metric-based approach to evaluating key aspects of software development processes and products in a wide variety of computing courses.

CCS CONCEPTS

• Applied computing ~ Education ~ Collaborative learning • Software and its engineering ~ Software creation and management ~ Collaboration in software development~ Programming teams • Software and its engineering ~ Software creation and management ~ Collaboration in software development ~ Software verification and validation ~ Empirical software validation

KEYWORDS: Team software development, Empirical software engineering, Soft skills, Collaboration, Software development process quality, Software product quality, Metrics

ACM Reference format:
Christopher Hundhausen, Adam Carter, Phillip Conrad, Ashun Tariq, & Olusola Adesope. 2021. Evaluating Commit, Issue and Product Quality in Team Software Development Projects. In *Proceedings of ACM SIGCSE Symposium (SIGCSE'21), March 13–20, 2021, Virtual Event, USA* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432362

1 Introduction

Given the prominence of team software development in industry, computing educators have long been interested in engaging undergraduate computing students in team software development projects (e.g., [32, 35]). A key pedagogical goal is to provide students with software development experiences that align with those they will encounter in the software industry, thus giving them opportunities to develop the skills and practices that are essential to success [6, 11, 37], particularly so-called "soft skills" [1, 8, 13]. While team software development projects are most frequently assigned in senior capstone courses (e.g., [20]), computing educators have explored their use in a variety of computing courses (e.g., [36]). There has also been great interest in engaging student teams in projects with "real clients" [31] and in free and open-source projects [33].

Among the many questions surrounding how to run team software development projects, one stands out as particularly important: *How do we systematically evaluate the quality of teams' processes and products, given that each student team works on a different software project*? Answers to this question could lead to more effective and ecologically valid pedagogical approaches for team software development projects.

To address this question, we use the Goal-Question-Metric (GQM) framework [2, 38] from the empirical software engineering literature to derive a collection of metrics for assessing the quality of teams' commits, issues, and final software products. To explore the use of these metrics in practice, we present an empirical study of 23 team software development projects involving 96 students in three computing courses at three institutions. The results demonstrate that the metrics are sensitive to differences in the quality of teams' commits, issues, and products. This work contributes a new metric-based approach to evaluating key aspects of software development processes and products in computing courses.

2 Related Work

2.1 Theoretical Foundations

Team software development projects aim to provide students with *authentic* learning experiences [39] that will prepare them for the software industry, where they are likely to work on teams. In addition to providing authenticity, team software development projects are a form of *cooperative learning*, one of the most effective and widespread instructional practices [23]. Cooperative learning aligns with a number of *social learning theories*, which tout the educational benefits of both community participation [27] and positive interdependence with others [23]. Our interest in evaluating commit and issue quality acknowledges that, in team development settings, commits and issues play a central communicative role in building a sense of community and interdependence within teams.

2.2 Measuring Process and Product Quality in the Software Industry

Our goal is to measure the quality of the software products produced by student teams, and the quality of the processes they use. *Empirical software engineering* research has performed such measurements in industrial settings for decades. In fact, metrics for software engineering are so well studied that multiple literature reviews of the research exist [19, 24, 26, 29].

In surveying this work, we struggled to find agreement on how to measure process and product quality. However, there was some agreement on how to *derive* measures of process and product quality: the Goal-Quality-Metric (GQM) framework [2, 38], which [29] cites as the most commonly used method. In the GQM framework, metrics are defined in a top-down manner by identifying high-level goals, questions about those goals, and metrics[1] to shed light on those questions. In Section 3, we apply this framework, starting with common learning goals for team software development projects.

2.3 Measuring Process and Product Quality in Student Team Software Projects

Fincher et al. [18] describes the challenges of assessing project work in computing education, in contradistinction to assessment of other computing course work. These include, among others: (1) it is larger scale, (2) both processes and products need to be assessed (3) student teams typically undertake "significantly different projects from one another," and (4) collaboration is a desirable or required objective to be assessed.

Despite these challenges, computing educators have carefully considered ways to evaluate team software projects (see [34] for a review). Clear [9, 10] and Herbert [20] provide general guidance. Other computing educators propose more detailed evaluation models, with a focus on software projects in the context of capstone courses (e.g., [16, 17, 25, 40]).

One issue of concern in the evaluation of team software projects is the fair distribution of credit across individual team members. While many approaches rely on self and peer assessment (see, e.g., [16, 17]), Buffardi [7] gauged individual contributions using process data from GitHub. In contrast, this work leverages GitHub data to evaluate how well teams meet course learning objectives, with an eye towards developing interventions to improve software engineering education.

In work most closely related to that presented here, computing educators have developed metrics for assessing team software projects. For instance, Linhoff and Settle [28] propose metrics firmly rooted in the specific learning goals of a game development course. Dubinsky and Hazzan [14] propose metrics based on the roles that students may play on a software development team. The metrics presented here, in contrast, are based on software processes captured through GitHub log data.

2.4 The Industry-Academia Gap

Studies of software developers in industry suggest a significant gap between students' undergraduate academic preparation and the skills they need to be successful software developers. In their seminal study, Begel and Simon [3, 4] shadowed new hires at Microsoft, finding that they struggled in five broad areas: communication, collaboration, technical skills, cognition, and orientation. They noted that only one of these skills related to the technical skills emphasized in academia. Subsequent studies of new software developers in industry have found similar gaps between the skills of new hires and the skills needed to succeed in the software industry [12, 15, 21].

The studies cited above motivate our interest in evaluating the quality of issues and commit messages. Four of the studies identified version control as a deficient skill [3, 4, 12, 15]; another two identified a deficiency in defining product requirements [12, 15]. The importance of this latter skill is also backed by a large survey of industry professionals [30]. Notice that aspects of both of these skills relate to *written communication*, a skill explicitly identified in [21].

3 Deriving Metrics with the GQM Framework

There are six stages in the Goal-Question Metric (GCM) framework for defining a software engineering metric [2, 38]:

1. Develop a set of goals and associated measurement goals for productivity and quality
2. Generate questions that define those goals as completely as possible in a quantifiable way
3. Specify the measures needed to answer those questions and track process and product conformance to the goals
4. Develop mechanisms for data collection.
5. Collect, validate and analyze the data in real time to provide feedback to projects for corrective action
6. Analyze the data to assess conformance to the goals and to make recommendations for future improvements

[1]While some authors make a technical distinction between "measures" and "metrics," others use the terms interchangeably.

In this section, we present our work toward steps 1–3. In Section 4, we present an empirical study representing steps 4 and 6. Step 5 is left for future work and discussed in Section 7.

3.1 Goals

Our overall goal is to help students learn the professional software development skills they need to succeed. We made this more precise by formulating three specific goals:

(1) Students will write commit messages that are consistent with industry expectations for quality.
(2) Students will specify software requirements (in the form of issues on a Kanban board) in a way that is consistent with industry expectations for quality.
(3) Students will produce software products of high quality.

Our motivation for Goals 1 and 2 is supported by the related work presented in Section 2.4. Goal 3 is fundamental to computing education: we want to know whether our students can produce quality software products.

3.2 Questions

Based on Goal 1, we formulated four questions related to **commit** quality:

(1a) ***atomic***: Do the code changes in the commit deal with one and only one concern?
(1b) ***accurate***: Does the commit message describe all changes, and only those changes, made in the commit?
(1c) ***precise***: Does the commit message unambiguously describe the changes made, situating the commit in the context of the code base or project?
(1d) ***justified***: Does the commit message describe why the change was made from the perspective of the end user?

Goal 2 led to five questions related to **issue** quality:

(2a) ***atomic***: Does the issue deal with one and only one concern?
(2b) ***descriptive title***: Is the issue title short and descriptive?
(2c) ***identifies impact***: Does the issue identify who is impacted by the change?
(2d) ***clearly described***: Does the issue clearly describe the changes to be made?
(2e) ***justified***: Does the issue describe the reason for the change from the perspective of the end user?

Finally, Goal 3 prompted four questions related to **software product** quality:

(3a) ***complexity***: To what degree does the software product demonstrate mastery of the technologies, knowledge and skills covered in the course?
(3b) ***reliability***: To what degree is the software free of bugs?
(3c) ***usefulness***: To what degree does the software product meet its target users' needs?
(3d) ***overall quality***: If you were the course instructor, what grade would you give the software product if you knew how long the team had to work on it?

In formulating the questions in this subsection, we had hoped to find guidance from the literature on empirical software engineering, and from the CS education literature on evaluating software engineering products. However, we failed to identify evaluation criteria and techniques in that literature that could readily address our specific goals.

Instead, for commits and issues, since our main concern was preparing students for the expectations they would encounter in industry, we were guided by industry discussions of good practices for commits and issues (e.g., [22, 41]). For product quality, we worked toward a set of criteria that (a) was general enough to apply to a variety of software products (our courses spanned mobile and web apps in various domains), and (b) could be applied within a reasonable time frame (within 20-30 minutes). Given that team projects provide opportunities for students to apply what they have learned so far, we adopted *complexity* to capture the extent to which a team project made use of the tools and technologies learned so far. Because team software products are presumably intended to be used by real-world users, we identified *reliability* and *usefulness* as two key concerns of real users. Finally, we added an *overall quality* category to acknowledge that instructors must ultimately assign a final grade to an academic project. We wanted to include a metric that gave multiple instructors the opportunity to discuss and converge on an overall quality rating, despite differences in the grading criteria used in their own courses.

3.3 Metrics

For questions 1a–1d and 2a–2e, we defined corresponding metrics in terms of the percentage of commits and issues for which we could answer "yes" to the question. In contrast, for questions 3a-3d, we defined the corresponding metrics in terms of a four-point quality scale where 1 = "Poor," 2 = "Deficient," 3 = "Acceptable" and 4 = "Excellent." We reserved a rating of 0 ("Failure") for software that could not be launched.

4 Empirical Study

To explore the value of the metrics in evaluating student teams' processes and software products, we now present a multi-institutional empirical study that addresses steps 4 and 6 of the GQM framework.

4.1 Courses and Participants

The study focused on team projects in courses at three universities (see Table 1): Humboldt State University (HSU), University of California, Santa Barbara (UCSB), and Washington State University (WSU).

Table 1. Key Attributes of the Courses Studied

Course Attribute	HSU	UCSB	WSU
Course Level	Upper Div.	Lower Div.	Upper Div.
Course Topic	Mobile apps	Soft. Eng.	Web Dev.
Course enrollment	15	80	65
# Participants	15	39	42
# Participant Teams	5	9	9

Project teams used GitHub for version control and project collaboration. Participants at HSU and WSU signed an informed consent form to release their GitHub data to the study. In contrast, participants at UCSB were part of teams that agreed to do their project work in public GitHub repositories, thus providing this study with access to their data.

4.2 Materials and Procedure

Table 2 presents key attributes of the team software development projects implemented in each course. The projects ranged in duration from 4 to 9 weeks, with sprints ranging from one to three weeks. Projects varied in terms of who defined them, how teams were formed, and how students were graded.

Table 2. Key attributes of the Projects Studied

Project Attribute	**HSU**	**UCSB**	**WSU**
Projects defined by	Students	Students	Instr./Students
Teams chosen by	Instructor	Instructor	Students
Team size	2-3	4-6	1-5
Sprint duration	1 week	3 weeks	1 week
Sprints in project	5	3	4
Grading method	Individual	Team	Team w/ind. multipliers

4.3 Data Collection and Sampling

We developed a web application that mined GitHub for the commits and issues in teams' repositories. Since, in some cases, multiple teams worked on the same repository, we mapped commits and issues to teams based on their authorship.

In addition to the GitHub data, we collected team's final software products. Two of the three courses also required final software demo videos. We collected those as well.

Given the large number of commits and issues logged by teams in this study, we analyzed, for each team, either (a) a 20% random sample of their commits and issues, or (b) 20 of each—whichever was greater. For teams with fewer than 20 commits or issues, we sampled all available commits or issues.

Table 3 presents, by course, counts of the data considered in this study. For analysis purposes, we *excluded* some of the commits prior to drawing our samples: (a) those that were made by students who did not provide informed consent (unless the commit was to a public repository); (b) those that involved only documentation (.md files), not code, and (c) those that were automatically generated by GitHub (e.g., to merge a pull request). Likewise, we excluded some of the issues: (a) those that were not closed or in the "Done" column of the team's Kanban board; and (b) those authored by students who did not provide informed consent (unless the issue was in a public repository).

4.4 Data Analysis

4.4.1 Commits and Issues. After iteratively developing a detailed evaluation manual, we employed a three-phase process to evaluate the sampled commits against questions 1a–1d, and the sampled issues against questions 2a–2e (see Section 3.2). The percent agreement and Cohen's Kappa (inter-rater reliability) values attained at the end of each phase are shown in Table 4.

Table 3. Counts of Included (Inc.) and Sampled (Sam.) Data Items by Course

	HSU		**UCSB**		**WSU**	
Data Item	Inc.	Sam.	Inc.	Sam.	Inc.	Sam.
Issues	55	44	250	176	155	127
Commits	187	92	1019	228	266	165
Software	5	5	9	9	9	9
Video Demo	5	5	9	9	0	0

Table 4. Percent Agreement (% ag.) and Cohen's Kappa (κ) after Each Phase of Commit and Issue Evaluation

	Phase 1		**Phase 2**		**Phase 3**	
Commit Metric	% ag,	κ	% ag.	κ	% ag.	κ
(1a) *atomic*	80	.35	95	.85	100	1.0
(1b) *accurate*	73	.37	94	.86	100	1.0
(1c) *precise*	73	.30	91	.76	99	.98
(1d) *justified*	79	.32	97	.89	100	1.0
Issue Metric						
(2a) *atomic*	93	.61	98	.91	100	1.0
(2b) *descript. title*	84	.65	97	.94	100	.99
(2c) *identifies impact*	93	.86	97	.94	100	1.0
(2d) *clearly described*	77	.41	87	.70	95	.89
(2e) *justified*	90	.78	98	.95	100	1.0

In the first phase, the first three coauthors independently evaluated the sample of commits and issues. We assigned two evaluators to each item such that no one evaluated the commits and issues of their own students. For each item, evaluators were asked to formulate a brief rationale for their decisions. In the second phase, each evaluator inspected the items where there were disagreements, changing their evaluations in cases where they thought their original evaluation was wrong. In the final phase, all three evaluators discussed the remaining disagreements. In cases where disagreements remained after this discussion, the third evaluator resolved the disagreement.

4.4.2 Software Products. After iteratively developing a detailed evaluation manual, we evaluated teams' software products in three phases. In Phase 1, the co-author who was the course instructor first debriefed the two co-authors who were not the course instructor (the evaluators) on the scope and goals of each team project. Next, the instructor led a live demo of the project's final software product. The two evaluators were invited to ask questions and to request interaction sequences for the instructor to attempt. In cases where the product failed to launch, the team's video demo, if available, was also consulted. This debrief and demo period was capped at 10 minutes for each project. To conclude Phase 1, the two evaluators independently rated the software product along the four quality dimensions, writing a rationale for each rating. The first column of Table 5 presents the percent agreement attained by the two evaluators after Phase 1.[2] Notably, no Phase 1 ratings differed by more than one point.

In Phase 2, the two evaluators revealed their ratings and rationales to each other. In cases of disagreement, the evaluators

[2]We opted not to compute Cohen's Kappa in our product evaluation process because of the low number of ratings involved (one per team per metric), and because the evaluations were scalar and not categorical.

were invited to discuss and change their ratings. Column 2 of Table 5 shows the percent agreement attained after this phase. In cases where disagreements remained after Phase 2, the course instructor resolved them in Phase 3. The total time for Phases 1, 2, and 3 was capped at 30 minutes.

Table 5. Percent Agreement at the End of the First Two Phases of Software Product Evaluation

Product Metric	Phase 1	Phase 2
(3a) *complexity*	58	96
(3b) *reliability*	71	100
(3c) *usefulness*	88	100
(3d) *overall quality*	75	96

4.5 Results

Table 6 presents metric values corresponding to commit and issue quality—that is, the percentages of sampled commits and issues that satisfied questions 1a–1d and 2a–2e (Section 3.2). Additionally, the table presents the percentages of *perfect* (i.e., atomic, accurate, precise and justified) commits and *perfect* (i.e., (atomic, has descriptive title, identifies impact, is clearly described and is justified) issues. Table 7 presents metric values corresponding to product quality. In Tables 6 and 7, values with superscript (a) are significantly different (Pearson's chi-squared, p < 0.05) from values in the same row with superscript (b).

Table 6. Commit and Issue Metric Values by Course

Commit Metric	HSU	UCSB	WSU
(1a) *atomic*	80%	80%	72%
(1b) *accurate*	64%	65%	50%
(1c) *precise*	^{a}45%	^{b}14%	^{b}25%
(1d) *justified*	^{a}38%	^{b}20%	^{b}15%
perfect commit	^{a}22%	^{b}5%	^{b}3%
Issue Metric			
(2a) *atomic*	^{a}94%	^{a}95%	^{b}75%
(2b) *descriptive title*	70%	^{a}80%	^{b}36%
(2c) *identifies impact*	^{b}27%	^{a}88%	^{b}28%
(2d) *clearly described*	55%	86%	60%
(2e) *justified*	18%	^{a}50%	^{b}8%
perfect issue	^{b}2%	^{a}21%	^{b}1%

Table 7. Product Quality Metric Values by Course

Product Metric	HSU	UCSB	WSU*
(3a) *complexity*	2.4	3.2	2.0
(3b) *reliability*	2.4	^{a}3.3	^{b}1.4
(3c) *usefulness*	1.8	3.1	2.0
(3d) *overall quality*	2.2	^{a}3.2	^{b}2.0

*Averages for WSU do not include evaluations from two teams whose projects would not run and therefore could not be evaluated.

Table 6 indicates there are differences in team proficiency based on the metrics. An analysis of variance detected significant differences between both the commit quality metrics ($F(3,92)=59.90$, $p < 0.001$, $\eta^2 = 0.66$) and the issue quality metrics ($F(4,115)=10.36$, $p < 0.001$, $\eta^2 = 0.26$). A post-hoc Bonferroni test on the commit quality metrics identified statistically significant differences between *atomic* and all other commit quality metrics ($p < 0.05$). In addition, a significant difference was detected between *accurate* and both *precise* and *justified* ($p < 0.05$). A post-hoc Bonferroni test on issue quality detected statistically significant differences between *atomic* and *justified* ($p < 0.01$), and between *descriptive title* and *justified* ($p < 0.01$).

We also tested for differences in metric values among teams in the same course. One HSU team exhibited significantly higher commit quality than their peers ($\chi^2 = 23.50$, $df = 4$, $p < 0.001$, V=0.26). This team scored 4's in all product metrics. Conversely, one WSU team had significantly lower commit quality than their peers ($\chi^2 = 26.23$, $df = 9$, $p = 0.002$, V=0.21). Notably, this team scored 1's in all product metrics.

In addition, we considered whether a statistical relationship existed between the process and product metrics. To reduce the likelihood of detecting *false* significance, we established that any *true* relationship between process and product metrics needed to be significant across all three courses. Using this standard, we did not detect statistically significant correlations between any process and product metrics. That is, no relationship was found between adherence to good process and final product quality.

5 Discussion

Inspection of Tables 6 and 7 suggests that the greatest strength of the teams in this study was their creation of *atomic* issues, and, to a lesser degree, their creation of *atomic* commits. In all three courses, teams scored significantly higher on the *atomic* metric than on most or all other quality metrics. Thus, creating issues and commits that focus on a single concern may come relatively easily to students. Instructors may be able to teach this behavior with minimal effort.

In contrast, our data show that students have much room for improvement when it comes to (a) clearly, accurately, and precisely describing commits and issues, and (b) justifying commits and issues. We suspect that teaching these best practices will require instructors to make the case that, even if they seem like a waste of time in smaller software projects, these practices are important and valuable in larger software projects. Frequent formative assessments, especially if they can be automated or streamlined, could also help in this regard.

With respect to the quality of commits and issues between courses, we found that HSU teams had higher-quality commits, and UCSB teams had higher-quality issues. While there are several possible explanations for this difference, the most obvious relates to the pedagogical choices made by the course instructors. The HSU instructor made it clear to students that their grades would be based in part on the quality of their commit messages, although the instructor's definition of commit quality differed from the definition of commit quality presented in Section 3.3. The UCSB instructor provided an issue template for students to use. This template aligned with the issue quality metrics defined in Section 3.2. However, even with these course incentives, the percentage of "perfect" commits and issues across all courses remained below

22%. These observations suggest that there is much room for improvement when it comes to these practices, and that carefully designed pedagogical practices have the potential to positively influence students' behaviors around issues and commits.

We failed to detect a statistically significant relationship between the process and product metrics. This was not a surprise. We suspect that the benefits of superior processes might require longer project durations to materialize than were present in our study. Yet, we did identify a single HSU team that was statistically *more* likely to produce high quality commits, and that also excelled in our product metrics. Likewise, we identified a single UCSB team that was statistically *less* likely to produce good quality commits, and that also scored poorly in our product metrics. We suspect that a follow-up qualitative investigation of these teams' processes could provide insight into the possible relationships between process and product metrics.

6 Threats to Validity

The threats to the validity of this work include threats to *internal*, *external*, and *construct* validity.

Internal validity reflects the degree to which the data collected in the study robustly applied our metrics to gauge student teams' commit, issue and product quality. One threat to internal validity is that we may not have collected a representative sample of these items. We have attempted to mitigate this threat by using random sampling, but there is no guarantee that our samples were truly representative. A second threat is that we may not have robustly applied our metrics. We mitigated this threat in three ways. First, by iteratively developing a detailed evaluation manual to guide the application of the metrics, we increased the chances that evaluators uniformly applied the metrics. Second, by ensuring that course instructors did not evaluate the work of their own students, we mitigated potential instructor bias—what Buffardi [7] identifies as the *Halo Effect.* Third, by breaking the process into three stages, we encouraged evaluators to be deliberative in their evaluations, reducing the chances of capricious decisions.

External validity reflects the degree to which our metrics are relevant to real-world software development. Given that version control and issue tracking are crucial to modern collaborative software development, our measurements of commit and issue quality are relevant to real-world contexts. However, software developers have mixed opinions about what makes for good commits and issues. Likewise, end users have mixed opinions about what makes for good software products. Thus, the external validity of our metrics is threatened by the reality that there is no clear consensus on these matters. We have tried to mitigate this threat by deriving the metrics from published sources.

Finally, *construct validity* has to do with the extent to which our metrics gauge the intended construct. A clear threat to the construct validity of our metrics is that they require human judgment of a complex entity (e.g., a commit spanning many lines) within a limited time frame, making the judgment prone to error. We have attempted to mitigate this threat by having multiple evaluators perform each judgment and by having them resolve disagreements through deliberative discussion.

7 Conclusions and Future Work

Using the GQM framework from the empirical software engineering literature, we have developed a collection of metrics for evaluating two aspects of process (commits and issues) and overall product quality in a wide variety of team software projects. Through an empirical study, we have demonstrated not only that these metrics are sensitive to differences in the quality of teams' processes and products, but also that teams performed better on some quality metrics than they did on others.

This work contributes a new metric-based approach to evaluating key aspects of software development processes and products in a wide variety of computing courses. Future work could build on this contribution by implementing Step 5 in the GQM process—that is, by adopting the metrics for *formative* assessment. Since it may not be feasible to assess every commit or issue, instructors could assess a randomly chosen sample at various points in the course and offer feedback on how to improve. Effectiveness could be measured by examining whether the process and product metrics improve over time.

In the current study, we did not establish a uniform set of criteria for process and product quality across all three courses, nor did we share our process and product quality with students. In future work, we could study the impact of sharing the metrics up front—a practice Biggs calls "constructive alignment" [5].

Like issues and commits, pull requests and code reviews are important avenues of communication within a software team. In future work, we would like to apply the GQM process to develop metrics for these, and to perform empirical studies that use them for formative and summative assessment. Similarly, the use of an online communication tool (e.g., Slack) is increasingly essential for collaboration in team software development projects. In future work, we would like to apply the GQM to derive metrics for measuring the extent to which students develop professional communications skills aligned with the learning goals of courses with team software development projects.

How to evaluate student and team success in software projects remains an important open question. We believe that leveraging data from online software development tools such as GitHub provides a promising way forward. We are optimistic that future research can leverage these data in increasingly sophisticated ways both to gain new insights into the relationships between teams' development processes and products, and to advance pedagogy through improved formative and summative assessment.

ACKNOWLEDGMENTS

This work is supported by a grant from the National Science Foundation (DUE-1915196).

REFERENCES

[1] Abernethy, K. and Treu, K. 2009. Teaching Computing Soft Skills: An Experiential Approach. *J. Comput. Sci. Coll.* 25, 2 (Dec. 2009), 178–186.
[2] Basili, V.R. 1992. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. *University of Maryland at College Park Computer Science Technical Report UMIACS-TR-92-96.* (1992), 1–24.

[3] Begel, A. and Simon, B. 2008. Novice software developers, all over again. *Proceedings of the Fourth International Workshop on Computing Education Research*. ACM. 3–14.

[4] Begel, A. and Simon, B. 2008. Struggles of new college graduates in their first software development hob. *SIGCSE Bull.* 40, 1 (Mar. 2008), 226–230. DOI:https://doi.org/10.1145/1352322.1352218.

[5] Biggs, J.B. 2011. *Teaching for Quality Learning at University: What the Student Does*. McGraw-Hill Education (UK).

[6] Bridging the Academia-Industry Gap in Software Engineering: A Client-Oriented Open Source Software Projects Course: 1AD. *https://www.igi-global.com/gateway/chapter/121869*. Accessed: 2020-08-20.

[7] Buffardi, K. 2020. Assessing Individual Contributions to Software Engineering Projects with Git Logs and User Stories. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, Feb. 2020), 650–656.

[8] Carter, L. 2011. Ideas for Adding Soft Skills Education to Service Learning and Capstone Courses for Computer Science Students. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2011), 517–522.

[9] Clear, T. 2010. Managing mid-project progress reviews: a model for formative group assessment in capstone projects. *ACM Inroads*. 1, 1 (Mar. 2010), 14–15. DOI:https://doi.org/10.1145/1721933.1721938.

[10] Clear, T. 2009. Thinking Issues: the three p's of capstone project performance. *ACM SIGCSE Bulletin*. 41, 2 (Jun. 2009), 69–70. DOI:https://doi.org/10.1145/1595453.1595468.

[11] Coppit, D. and Haddox-Schatz, J.M. 2005. Large Team Projects in Software Engineering Courses. *SIGCSE Bull.* 37, 1 (Feb. 2005), 137–141. DOI:https://doi.org/10.1145/1047124.1047400.

[12] Craig, M., Conrad, P., Lynch, D., Lee, N. and Anthony, L. 2018. Listening to early career software developers. *J. Comput. Sci. Coll.* 33, 4 (2018), 138–149.

[13] Damian, D. and Borici, A. 2012. Teamwork, coordination and customer relationship management skills: As important as technical skills in preparing our SE graduates. *2012 First International Workshop on Software Engineering Education Based on Real-World Experiences (EduRex)* (2012), 37–40.

[14] Dubinsky, Y. and Hazzan, O. 2006. Using a role scheme to derive software project metrics. *Journal of Systems Architecture*. 52, 11 (Nov. 2006), 693–699. DOI:https://doi.org/10.1016/j.sysarc.2006.06.013.

[15] Exter, M. 2014. Comparing educational experiences and on-the-job needs of educational software designers. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2014), 355–360.

[16] Farrell, V., Farrell, G., Kindler, P., Ravalli, G. and Hall, D. 2013. Capstone project online assessment tool without the paper work. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (New York, NY, USA, Jul. 2013), 201–206.

[17] Farrell, V., Ravalli, G., Farrell, G., Kindler, P. and Hall, D. 2012. Capstone project: fair, just and accountable assessment. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education* (New York, NY, USA, Jul. 2012), 168–173.

[18] Fincher, S., Petre, M. and Clark, M. 2001. *Computer Science Project Work: Principles and Pragmatics*. Springer Science & Business Media.

[19] Gómez, O., Oktaba, H., Piattini, M. and García, F. 2008. A Systematic Review Measurement in Software Engineering: State-of-the-Art in Measures. *Software and Data Technologies* (Berlin, Heidelberg, 2008), 165–176.

[20] Herbert, N. 2018. Reflections on 17 Years of ICT Capstone Project Coordination: Effective Strategies for Managing Clients, Teams and Assessment. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2018), 215–220.

[21] Hewner, M. and Guzdial, M. 2010. What game developers look for in a new graduate: Interviews and surveys at one game company. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2010), 275–279.

[22] Hutterer, P. 2009. Who-T: On commit messages. *Who-T*.

[23] Johnson, D.W. and Johnson, R.T. 2009. An Educational Psychology Success Story: Social Interdependence Theory and Cooperative Learning. *Educational Researcher*. 38, 5 (2009), 365–379. DOI:https://doi.org/10.3102/0013189X09339057.

[24] Kitchenham, B. 2010. What's up with software metrics? – A preliminary mapping study. *Journal of Systems and Software*. 83, 1 (Jan. 2010), 37–51. DOI:https://doi.org/10.1016/j.jss.2009.06.041.

[25] von Konsky, B.R. and Ivins, J. 2008. Assessing the capability and maturity of capstone software engineering projects. *Proceedings of the tenth conference on Australasian computing education - Volume 78* (AUS, Jan. 2008), 171–180.

[26] Kupiainen, E., Mäntylä, M.V. and Itkonen, J. 2015. Using metrics in Agile and Lean Software Development – A systematic literature review of industrial studies. *Information and Software Technology*. 62, (Jun. 2015), 143–163. DOI:https://doi.org/10.1016/j.infsof.2015.02.005.

[27] Lave, J. and Wenger, E. 1991. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press.

[28] Linhoff, J. and Settle, A. 2009. Motivating and evaluating game development capstone projects. *Proceedings of the 4th International Conference on Foundations of Digital Games* (New York, NY, USA, Apr. 2009), 121–128.

[29] Meidan, A., García-García, J.A., Ramos, I. and Escalona, M.J. 2018. Measuring Software Process: A Systematic Mapping Study. *ACM Computing Surveys*. 51, 3 (Jun. 2018), 58:1–58:32. DOI:https://doi.org/10.1145/3186888.

[30] Misic, M.M. and Russo, N.L. 1999. An assessment of systems analysis and design courses. *Journal of Systems and Software*. 45, 3 (Mar. 1999), 197–202. DOI:https://doi.org/10.1016/S0164-1212(98)10078-X.

[31] Murphy, C., Sheth, S. and Morton, S. 2017. A Two-Course Sequence of Real Projects for Real Customers. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2017), 417–422.

[32] Perkins, T.E. and Beck, L.L. 1980. A Project-Oriented Undergraduate Course Sequence in Software Engineering. *SIGCSE Bull.* 12, 1 (1980), 32–39. DOI:https://doi.org/10.1145/953032.804607.

[33] Postner, L., Ellis, H.J.C. and Hislop, G.W. 2018. A Survey of Instructors' Experiences Supporting StudentLearning using HFOSS Projects. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, Feb. 2018), 203–208.

[34] Richards, D. 2009. Designing Project-Based Courses with a Focus on Group Formation and Assessment. *ACM Transactions on Computing Education*. 9, 1 (Mar. 2009), 2:1–2:40. DOI:https://doi.org/10.1145/1513593.1513595.

[35] Robillard, P.N. 1996. Teaching Software Engineering through a Project-Oriented Course. *Proceedings of the 9th Conference on Software Engineering Education* (USA, 1996), 85.

[36] Saltz, J.S. and Heckman, R.R. 2018. A Scalable Methodology to Guide Student Teams Executing Computing Projects. *ACM Trans. Comput. Educ.* 18, 2 (Jul. 2018). DOI:https://doi.org/10.1145/3145477.

[37] Sherriff, M. and Heckman, S. 2018. Capstones and Large Projects in Computing Education. *ACM Trans. Comput. Educ.* 18, 2 (Jul. 2018). DOI:https://doi.org/10.1145/3229882.

[38] Solingen, R. van, Basili, V., Caldiera, G. and Rombach, H.D. 2002. Goal Question Metric (GQM) Approach. *Encyclopedia of Software Engineering*. John Wiley & Sons.

[39] Stein, S.J., Isaacs, G. and Andrews, T. 2004. Incorporating authentic learning experiences within a university course. *Studies in Higher Education*. 29, 2 (Apr. 2004), 239–258. DOI:https://doi.org/10.1080/0307507042000190813.

[40] Vasilevskaya, M., Broman, D. and Sandahl, K. 2015. Assessing Large-Project Courses: Model, Activities, and Lessons Learned. *ACM Transactions on Computing Education*. 15, 4 (Dec. 2015), 20:1–20:30. DOI:https://doi.org/10.1145/2732156.

[41] Writing a proper GitHub issue: 2018. *https://medium.com/nyc-planning-digital/writing-a-proper-github-issue-97427d62a20f*. Accessed: 2020-08-27.