An Empirical Study of the Content and Quality of Sprint Retrospectives in Undergraduate Team Software Projects

Christopher Hundhausen
chris.hundhausen@oregonstate.edu
Oregon State University
Corvallis, Oregon, USA

Phillip Conrad
phtcon@ucsb.edu
UC Santa Barbara
Santa Barbara, California, USA

Ahsun Tariq
tariqa@oregonstate.edu
Oregon State University
Corvallis, Oregon, USA

Surya Pugal
spugal@ucsb.edu
UC Santa Barbara
Santa Barbara, California, USA

Bryan Zamora Flores
bzamoraflores@ucsb.edu
UC Santa Barbara
Santa Barbara, California, USA

ABSTRACT

The *retrospective*, or *retro*, is a fundamental component of the Agile process, widely used in both software engineering courses and industry. In a retro, teams come together at the end of a sprint to reflect on their team's performance. We conducted an empirical study to explore three research questions concerning retros in undergraduate team projects: (1) What do students reflect on? (2) What is the quality of their reflections? (3) How do teams' retros vary in terms of content and quality? Our study analyzed a corpus of 963 statements documented in the retros of 32 undergraduate software teams (n = 182 students) enrolled in four software engineering courses at two North American universities. A content analysis revealed that teams reflected most often on their work, communication, and collaboration practices. Nearly a third of teams' reflections focused on their general work practices, while nearly half focused on specific areas of the software development lifecycle—most prominently, pull requests, issues, and coding/testing/debugging. An analysis of the quality of teams' retro reflections showed that only 13% provided justification for a strategy to be stopped, continued, or started. An analysis of team-by-team results indicated significant differences in teams' retro content and quality. We compare these results to previous studies of retros in academia and industry, and consider their implications for software engineering education.

CCS CONCEPTS

• **Human-centered computing** → *Empirical studies in collaborative and social computing*; • **Social and professional topics** → **Software engineering education**; **Software engineering education**; **Team Projects**; • **Software and its engineering** → **Programming teams**.

KEYWORDS

team software projects, software engineering education, Agile development, retrospectives, metacognition, reflection, content analysis

ICSE-SEET '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0498-7/24/04
https://doi.org/10.1145/3639474.3640074

ACM Reference Format:
Christopher Hundhausen, Phillip Conrad, Ahsun Tariq, Surya Pugal, and Bryan Zamora Flores. 2024. An Empirical Study of the Content and Quality of Sprint Retrospectives in Undergraduate Team Software Projects. In *46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3639474.3640074

1 INTRODUCTION

The *Agile* development approach [30]—widely adopted in both software engineering education and industry—is emphatic about the need for software teams to reflect on their processes via a *retrospective* (or *retro*) [12], in which a team comes together at the end of each work period ("sprint") to reflect on:

(1) What went well (and should be continued)?
(2) What did not go well (and should be stopped or changed)?
(3) What new things should be tried in the next sprint?

Retros engage students in reflecting on, assessing, and improving their processes; thus, studying retros can provide educators with insight into what teams are struggling with, where they are succeeding, and how carefully they are reasoning about their development process. This paper presents an empirical study that explores three research questions related to retros:

RQ 1 : *What do teams focus on in retros?*

RQ 2 : *What is the quality of teams' retro reflections?*

RQ 3 : *How do teams' retros vary in terms of content and quality?*

Our study considers the retros of undergraduate software development teams engaged in required projects in four offerings of two undergraduate courses held in person at two large North American research universities (n = 32 teams, 182 students) between 2021 and 2022. To the best of our knowledge, this work contributes the largest-scale study of the content of retros to date, together with the first-ever analysis of the *quality* of retro reflections.

2 RELATED WORK

2.1 Reflection in Software Engineering and Software Engineering Education

A form of project-based [36] and collaborative [7] learning, team software development projects are seen as instrumental in helping students to develop the soft and technical skills needed to succeed in the software profession [18, 19]. In team software projects, *reflection*—assessing the team's processes and progress to determine how the team can improve—is regarded as crucial to success [23]. Professional software engineers have long been interested in honing effective reflective practices [3, 14], which are widely seen as a hallmark of professional expertise [42].

Lamororeux [23] highlights several benefits of reflective practice in software development teams: It helps developers improve practices, build trust, communicate openly, and improve synchronization and productivity. Schön [42] distinguishes between two key forms of reflection: *reflection-in-action*, in which one assesses one's practices while in the midst of performing them; and *reflection-on-action*, in which one assesses one's practices after the fact. This research considers *retros*, which are a form of the latter.

Reflection is a key component of the more general practice of *metacognition*, which encompasses planning, monitoring, and reflecting on thinking and behavior [22, 32, 34]. Metacognition has been widely studied in computing education [29, 38], and has been shown to be a crucial soft skill in professional software development [1, 37, 43]. Moreover, it has been identified as being particularly deficient in studies of new software developers in industry [4, 5]. By studying student teams' use of reflection in retros, our work aims to help address this deficiency by laying a foundation for improved pedagogical and technological interventions.

2.2 Studying and Promoting Reflection in Software Engineering Education

There has been great interest in developing approaches to promote student reflection in the software development process [10, 16, 31, 37, 39]. While much of this research focuses on pedagogical approaches, there have been a few efforts to collect and analyze empirical data on students' reflections in the context of software engineering projects (e.g., [9, 17, 31, 39]). In one study, students were found to exhibit low participation in reflection-based exercises and would often participate only when they were frequently reminded [39]. Likewise, in another study, reflection was found to be rare in undergraduate team software development projects, with team communications overwhelmingly centered around planning and technical contributions [17]. In addition to being of limited quantity, reflections have been found to be qualitatively shallow in team projects [31]. A common theme across these studies is that students are reluctant to engage in reflection, and that when they do engage in reflection, their reflections tend to be at a superficial level. Our study reinforces this theme, while focusing explicitly on the *retro* as opposed to more general reflection activities during software development.

2.3 Studying Agile Retros

The agile *retrospective* (or *retro*) is a form of group reflection [46]. The Agile community of practice has written widely about the retro (e.g., [11, 12, 20, 25, 28]. In addition, several academic papers have proposed approaches for making retros more effective (e.g., [6, 35, 40, 41, 45]. While these papers outline best practices for engaging teams in retros, they do not present empirical studies of retro content and quality.

At least four prior empirical studies have performed content analyses of retros in an attempt to understand their focus and how they influence team practices and decision making. Only one of these studies occurred within the context of software engineering education [15]; the other three considered teams in industry [2, 13, 26]. Table 1 juxtaposes this body of work with our own study. As shown, the content coding schemes used in these studies differ widely, making it difficult to compare results across studies. Nonetheless, the table highlights two distinguishing features of our study: (1) Ours is by far the largest, spanning 182 developers, 32 teams, and 72 retros; and (2) ours is the only one to analyze the *quality* of retro reflections.

In the only study of retros in computing education, Gestwicki and McNely [15] performed a content analysis of artifacts from retro meetings, identifying four main themes: *Collaboration* (identifying opportunities for improving cooperation and social cohesion), *Learning* (identifying common areas where the team needed to improve their knowledge and skills), *Support* ("the team's articulation of their external dependencies") and *Community* (the external stakeholders and context in which the application was being developed). Across seven retros, the *Collaboration* theme was dominant, always representing over half of the responses. Our study is similar to this study in its use of content analysis, and in its finding that collaboration is a major theme in retros. However, our study differs from this study in three key respects: (a) it uses a different content analysis framework, (b) it analyzes reflection quality in addition to content, and (c) it considers multiple courses at two different universities instead of a single course.

Three studies have investigated the retros of professional software teams. In perhaps the most extensive study, Lehtinen et al. [26] examined 37 retros from multiple teams at a single industrial software organization over three years. While the authors also performed a content analysis of statements from retros, they categorized statements more broadly than we did, identifying statements as *positive* (29%), *negative* (51%), or *corrective action* (20%). As the primary purpose of a retro is continuous improvement of team performance, identification of corrective actions is vital, so they further analyzed the statements about corrective actions and found teams focused most on areas that were under team control, including task outcome, work practices, task difficulty, instructions, estimations/resources and schedules.

Andriyani et al. [2] performed an investigation of agile retros based on four teams comprising 16 professional software engineers in total. By performing face-to-face interviews with developers, observing team retro meetings and applying thematic analysis, the authors found three stages of reflection: *reporting and responding*, *relating and reasoning*, and *reconstructing*. In the *reporting and*

Table 1: Comparison of Empirical Studies of Retrospectives.

Year	Study	Context	*n*	*t*	*r*	Duration	Content Categories Used	Refl. Quality Studied?
2013	Gestwicki & McNely [15]	Academia	13	1	7	15 wks	*Collaboration*, *Learning*, *Support*, *Community*	No
2017	Lehtinen et al. [26]	Industry	30	9	37	9 wks	10 *Process Area* categories (e.g., *Implementation*, *Testing*) × 4 *Topic Type* categories (*People*, *Tasks*, *Methods*, *Environment*) × 3 *Outcome* categories (*Positive*, *Negative*, *Corrective Action*)	No
2017	Andriyani et al. [2]	Industry	16	4	4	2 wks	*Reporting & Responding*, *Relating & Reasoning*, *Reconstructing*, each with 1-3 lower-level themes	No
2018	Dingsøyer et al. [13]	Industry	18	2	10	5 mos	*People/Relationships*, *Process*, *Tools*, *Project*, *Other Teams*	No
2024	This Study	Academia	182	32	72	3–5 wks	Five "Concern" categories (*Collaboration*, *Communication*, *Work*, *Learning*, *Camaraderie*) × 13 "Focus Area" categories aligned with activities and artifacts in software development lifecycle	Yes

n=number of developers, *t*=number of teams, *r*=number of retros. Duration = length of the sprint/iteration(s) reflected on in the retros under study

responding stage, the authors identified two major themes: identification of barriers to team progress, and individual sentiments about various situations. In the *relating and reasoning* stage, there were three themes; evaluating previous action points, identifying background causes, and identifying future action points. In the *reconstructing* stage, the team would come up with a plan to focus on a given action point. In addition to identifying themes that occur in retros, the study identified different levels of reflection, observing that even professional software developers may fail to engage in higher levels of reflection in team retros. The study also highlighted a key problem with team retros: individual reflections may be subject to *memory bias*; team members may recall events based on personal experiences [6]. Based on these findings, the authors proposed a framework for conducting agile retros.

Finally, Dingsøyr et al. [13] did a content analysis of the minutes from six retros performed by two teams over a five month period (*n*=12 retros). The authors categorized the items in the minutes using the categories *people/relationships*, *process*, *tools*, *project* and *other teams*. They found that of 109 items categorized, the dominant categories was *process* (41 items, 38%) and *people/relationships* (30 items 28%). They also found that across the twelve retros, there were 36 suggestions for improvement. The dominant category for these was *process* (13 suggestions, 36%).

3 METHODS

3.1 Courses and Participants

This study considered four offerings of two software engineering courses that took place between Fall, 2021 and Fall, 2022:

Course A ("Advanced Application Development") was taught by the second author at the Unversity of California (UC) Santa Barbara. This 10 week course is taken primarily by second and third year undergraduate computer science majors. It focuses on the development of full-stack web applications through a series of team assignments, followed by a 2-3 week team project focused on a legacy application. The course emphasizes both Agile practices (standups, sprints, Kanban, user stories, acceptance criteria, retros) and GitHub workflows (pull requests, code reviews).

Course B ("Web Development") was taught by the first author at Washington State University. It is a 15-week advanced undergraduate course in full-stack web development. During the first 10 weeks, students learn web programming through lectures, live coding demos, and a series of individual assignments that build upon each other to produce a full-stack web app. During the final five weeks, students form software teams to enhance, and develop new features for, the web app constructed in the first part of the course. As part of the team project, students learn the same Agile practices and GitHub workflows emphasized in Course A.

While both courses took place in person, they provided students with the option to participate remotely via Zoom. Lectures were recorded and made available to students. Outside of class meetings, students engaged with online learning materials and communicated via online communication tools.

Table 2 presents demographic data on study participants. The study involved three offerings of Course A and one offering of Course B. The study was approved by the Institutional Review Boards of each university; students could opt in to the study through an online informed consent form. For a team's retro data to be included in the study, *all* team members had to consent. As shown in Table 2, in the four courses involved in this study, all members of 32 of the 48 teams (*n*=182 students) consented to participate.

Table 2: Details on Teams and Students in Study

	# Teams		# Students					
				Consenting				
Course	Enr.	Cons.	Enr.	*n*	M	F	U	*M* Age
A1	12	8	72	48	41	5	2	20.5
A2	12	8	71	47	39	8	0	20.3
A3	12	8	71	48	43	4	1	20.2
B	12	8	60	39	34	5	0	24.0
Totals:	**48**	**32**	**263**	**182**	**157**	**22**	**3**	**20.9**

Enr.=Enrolled, Cons.=Consenting, *n*=Total # of students, F=Female, M=Male, U=Unspecified/Other, M Age=Mean Age

3.2 Materials and Procedure

3.2.1 Team Project. Table 3 presents details on the team software development projects in each course. In Course A, teams of up to six

members were formed through a CATME team-building survey [24]. Students in Course B were asked to form their own teams of up to five students; unassigned students were randomly placed on teams with slots available. Teams in Course A were randomly assigned to one of two legacy full stack web development projects from previous course offerings. In Course B, all teams worked on the same code project: a full-stack web app that served as the running example in the first part of the course. Team projects, which were worth 25% of students' course grade in both courses, spanned three to five weeks.

As they engaged in the project, student teams in both courses were expected to employ the agile development practices emphasized in each course, including the use of issues, Kanban (project) boards, feature branches, pull requests, code reviews, automated tests, and retros. Grading of students and teams differed between the two courses. In Course A, teams were awarded a grade based on the number of *story points* they completed during the term, with teaching personnel determining the story point values of each team's completed issues. In Course B, the instructor used a structured rubric to perform a detailed evaluation of teams' GitHub repositories, chat channels, and a required sprint report.

3.2.2 Retros. In their projects, teams engaged in two (Course A) or three (Course B) sprint cycles. At the end of each sprint, teams were required to participate in a retro. In Course A, teams were instructed to choose a retro leader to run the retro session. At the start of the retro, students were given five minutes to independently compile lists of practices to (a) start in the next sprint, (b) stop doing in the next sprint, and (c) continue doing in the next sprint. Team members were asked to read and explain their items aloud; the team leader would then insert the items into the appropriate list in a team document. At the end of the retro, team members voted on which items were most important. The three items with the most votes were flagged for review at the next retro.

In Course B, teams were similarly prompted to participate in a retro at the end of each sprint. In the retro meeting, which teams were required to video record, teams were instructed to compile three separate lists: (a) what went well (i.e., continue) (b) what could be improved (i.e., stop), and (c) what changes to implement in the next sprint (i.e., start). Teams documented their lists in a section of a sprint report they were required to submit for each sprint.

3.3 Data Collection and Analysis

We collected teams' retrospective statements from the documents they were required to submit at the end of each sprint. For analysis purposes, we created a spreadsheet initially containing one row for each retro statement. In each row, we also indicated (a) the team that made the statement, (b) the course in which the team was enrolled, (c) the number of the sprint in which the statement was made, and (d) the prompt ("change/start," "continue," or "improve/stop") to which the statement responded. While most statements expressed a single thought or idea, we found a few statements that addressed multiple thoughts or ideas. We partitioned such statements into multiple segments such that each segment expressed a single thought or idea.

To address **RQ 1**, we conducted a bottom-up thematic analysis [8] of the statements. Through an iterative process that involved experimentally coding groups of 50 statements based on their content [21], three of the coauthors gradually converged on a two-tiered content coding scheme. First, each statement is coded according to the *primary concern* it identifies (see Table 4). Then, each statement is coded into one or more *focus areas* that identify where in the software development cycle the concern lies (see Table 5).

To address **RQ 2**, we began with Leijen et al.'s [27] framework for assessing the quality of reflection. Drawing on Moon's [33] distinction between *superficial* and *deep* reflection, Leijen et al.'s framework distinguishes four different levels of reflection based on the depth of its argumentation. At the most superficial level of reflection are *descriptive* statements that simply describe a strategy that the team took or would like to take. At the second level of the framework are *justified* statements that not only describe a strategy, but also provide a rationale for the strategy grounded in logic, reason, data, or observation. At the third level are *critiqued* statements, which provide a more thorough justification of a strategy by considering at least one pro and one con of the strategy. Finally, at the deepest level of reflection in the framework are *discussed* statements, which differ from *critiqued* statements in that they consider the pros and cons of *two or more* strategies.

After applying this framework to a subset of our corpus, we decided to make two modifications. First, to emphasize that retro statements identify general *strategies* for improving team processes, we changed the word *statement* to *strategy* in the four reflection levels. Second, we discovered that the framework was unable to characterize one class of retro statements observed in our corpus—those that assert a *specific fact or observation* rather than a general strategy that can be stopped, continued, or started. An example of such a statement is *"We have assigned Issue #55 to the next sprint."* Because statements like these are not generalizable beyond the current sprint, they are arguably at at the most superficial level of reflection. To account for this type of statement, we added a fifth level (*Statement*) to the reflection framework, as shown in Table 6.

Using a coding manual with more detailed definitions, examples, and rules than those shown in the tables, two of the coauthors independently coded a 15% sample of our corpus (n = 144 statements), achieving 93% agreement (Cohen's Kappa 0.89) on the *Concern* categories, 92% agreement (Cohen's Kappa 0.89) on the *Focus Area* categories, and 99% agreement (Cohen's Kappa 0.97) on the *Reflection Level* categories. Having established high inter-rater reliability, a single analyst coded the remaining statements.

To address **RQ 3**, we collected the team-by-team counts of retro statements coded into each content and reflection level category. These counts were used as a basis for both statistical analysis and visual exploration of graphs.

Table 3: Key Dimensions of Team Software Projects

Project Dimension	Course A	Course B
Starting Code Base	Legacy full-stack web app projects developed in previous course offerings	Full stack web app project developed in first part of course
Team Formation	Based on CATME survey	Self-selected
Team size	5-6	3-5
Project Length (Weeks)	3-4	5
# Sprints	2	3
Project Weight	25% of course grade	25% of course grade

Table 4: Definitions of "Concern" Categories

Concern	Definition	Observed Example(s)
Collaboration Practice	Focuses on practices, strategies, processes for collaborating on work. Most prominently, this includes strategies for delegating and coordinating work (who does what when).	*"Have a mechanism to know who's reviewing whose code"* *"Schedule daily standups"* *"Helping each other out"* *"Code in pairs"*
Communication Practice	Focuses on strategies, approaches, and processes for communicating. Most prominently, this includes chats, meetings, asking for and receiving help, communicating status, and asking and answering questions.	*"We need to communicate more frequently."* *"We didn't clarify deadlines when we communicated."* *"Do Daily standups"* *"Fast responses on Slack"* *"Checking Slack frequently"*
Work Practice	Focuses on practices, strategies, and processes for getting work done.	*"Procrastinating"* *"Optimizing github workflows so that mutation testing doesn't take extremely long"* *"Reviewing code thoroughly in PRs"* *"Better understand the assignment before creating tickets."*
Learning Practice	Focuses on strategies, practices, and processes for learning or gathering information, including the target content of such efforts.	*"We need to learn MongoDB"* *"Stop going straight to the instructor for help if the answer can be found through other resources"*
Camaraderie Practice	Focuses on strategies,practices, and processes for building team rapport and camaraderie	*"We need kit kats at every meeting."* *'Hold a party at end of each sprint"*
None	Prompt response is missing or it is stated that no response is needed.	*"We don't need to change anything"*

4 RESULTS

4.1 Retro Content (RQ 1)

Figure 1a shows the percentage of retro statements that fell into each *Concern* category. As the figure illustrates, 95% of all retro statements focused on three concerns: *Work Practices* (50%), *Communication Practices* (27%), and *Collaboration Practices* (18%). The remaining 5% of the retro statements focused on *Learning Practices*, *Cameraderie Practices*, and *None*.

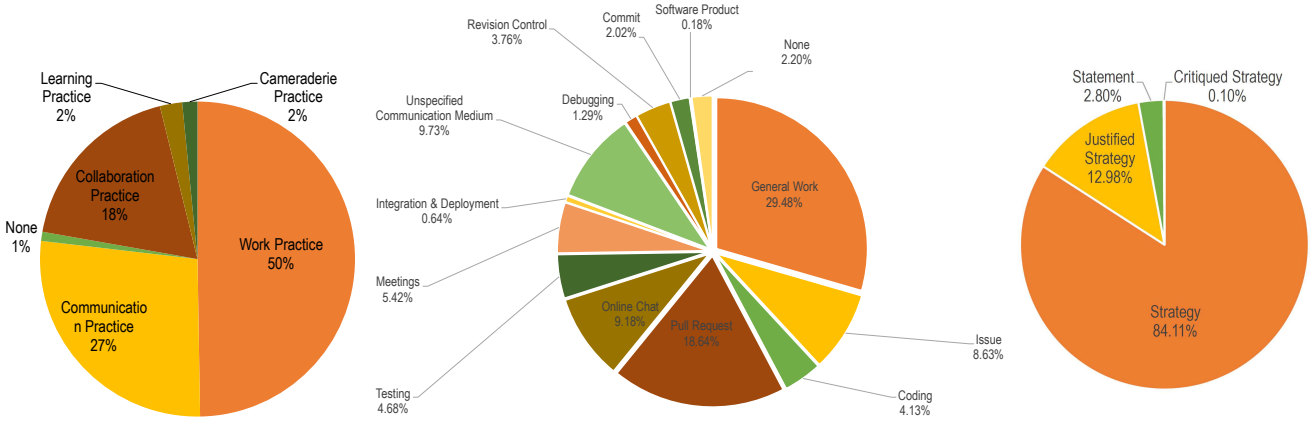
Figure 1b classifies retro statements by software development lifecycle areas. Nearly 30% of the statements focused on "work" in general, without specifying a specific activity or software engineering artifact. The next most common focus area was communication (24%), including online (9%), face-to-face (5%), and of an unspecified nature (10%). Some 19% of statements focused on pull requests, which encompassed code review, code merging, and resolving merge conflicts. Activities involving issues and the Kanban board were the focus of 9% of the statements. None of the remaining six focus areas accounted for more than 5% of the statements.

4.2 Reflection Quality (RQ 2)

Figure 1c classifies retro statements based on their reflection quality. The dominant level of reflection exhibited in retro statements was *Strategy* (84.1%), which simply stated a practice to start, stop, or continue, without providing any justification. Just 13% of retro statements included some form of justification (*Justified Strategy*). Only one statement (0.1%) exhibited a higher form of reflection (*Critiqued Strategy*) by identifying at least one pro and one con. *Critiqued Strategy* was the highest form of reflection found in our sample; no retro statements considered the pros and cons of alternative strategies (*Discussed Strategy*). The remaining 2.8% of retro statements simply stated facts or observations and were classified as the lowest form of reflection (*Statement*).

Table 5: Definition of "Focus Area" Categories

Focus Area	Definition	Observed Example(s)
General Work	Concern focuses on general work, which can be referred to in a variety of ways, including "tasks," "assignments," "blocks," "things," or "issues"	*"Start tasks early"* *"Staying updated on everyone else's progress"* *"Staying updated on everyone else's progress"* *"Helping each other with their difficulties"*
Meetings	Concern focuses on meetings (face to face or online)	"Being efficient during class time" *"Speaking up in stand-ups"*
Online Chat	Concern focuses on online chat	*"Checking Slack more often"* *"Giving status updates in Discord"*
Unspecified Communication Medium	Concern focuses on an unspecified communication meeting (applies only to "Communication Practice")	*"Communicating more often"* *"Speaking up when troubles arise"*
Coding	Concern focuses on coding/computer programming	*"Writing better comments in code"* *"Using appropriate variable names"*
Debugging	Concern focuses on debugging	*"Finding bugs prior to the PR"*
Testing	Concern focuses on testing, continuous integration testing, or quality assurance	*"Ensuring broad test coverage"*
Integration & Deployment	Concern focuses on integration and deployment, including GitHub work flows	*"Making sure UI uses API calls"*
Software Product	Concern focuses on a software product produced by the team	*"Pay more attention to user interface usability"*
Pull Request	Concern focuses on pull requests, including code review, merging code and handling merge conflicts	*"Reviewing thoroughly prior to merge"*
Issue	Concern focuses on issues, including functional or non-functional requirements, features, user stories, and the Kanban board.	*"Fully implemented the graphic feature in round options"* *"Having so many issues in progress at once, at most 1 per person"*
Commit	Concern focuses on code commits	*"Commit more often."* *"Descriptive commit messages"*
Revision Control	Concern focuses on revision control, including pushing, pulling, branch management and rebasing, but not merging or resolving merge conflicts	*"Pull code after every PR."*
None	Team chooses not to respond to a retro prompt or says no reponse is needed.	



(a) by Concern (b) by Software Dev. Lifecycle Focus Area (c) by Reflection Level

Figure 1: Retro Statements Classified Three Ways

Table 6: Definition of "Reflection Level" Categories, from Shallowest to Deepest

Reflection Level	Definition	Observed Examples
Statement	Retro statement is not an actionable strategy; it cannot be started, stopped, continued, or improved	*"Issue 55 has been set in TMP 2"*
Strategy	Retro statement includes an actionable strategy (something that can be started, stopped, continued, or improved)	*"Start work earlier"* *"Add comments to code"*
Justified Strategy	Retro statement includes an actionable strategy, together with justification for why the team wants to stop, continue, improve, or start the strategy. Justification can be logic/reason, evidence for why the statement is true (data, observation), or elaboration of the effect of the team's action or proposed action.	*"Having strict early deadlines for each pull request, which prevents last minute fiascos"* *"Communicating when we get stuck. We often had bugs that other team members had encountered already so this helped in reducing debugging time"*
Critiqued Strategy	Retro statement performs a principled critique of an actionable strategy. This includes a statement of at least one pro (strength/advantage) and one con (weakness/disadvantage)	*"Several team members were assigned to multiple tasks, rather than each member assigned a single task. We were able to complete tasks faster this way, while maintaining a level of subject knowledge that was consistent across the team"*
Discussed Strategy	Retro statement considers/discusses alternative/multiple strategies, including pros/cons and rationale for changes	None observed

4.3 Retro Content and Quality by Team (RQ 3)

Figure 2a presents a stacked bar chart depicting the number of retro statements coded into each *Concern* category. Likewise, Figures 2b and 2c show stacked bar charts for the *Focus Area* categories and *Reflection Level* categories, respectively. In these charts, each team is represented by a bar whose height corresponds to the number of retro statements documented by the team. Bar shading is used to indicate the number of statements coded into each category. Categories are displayed in the same order (from most common to least common) in all bars to facilitate direct comparisons of bars.

Inspection of these figures yields four notable observations. First, while on average, teams documented 30.1 retro statements, they varied widely with respect to the number of retro statements they documented (SD = 12.7). Team B7 documented the fewest statements (13), while Team A2-8 documented the most (66). Second, while visual inspection of the charts suggests similar categorical patterns across the teams, chi-square tests of homogeneity indicate that teams' categorical distributions are significantly different with respect to *Concern* (df = 155, $\chi^2 = 655.0, p < 0.001$), *Focus Area* ($df$ = 403, $\chi^2 = 780.7, p < 0.001$), and *Reflection Level* (df = 93, $\chi^2 = 712.5, p < 0.001$). Third, with respect to *Reflection Level* (see Figure 2c), Teams B7 and B8 stand out for their extensive use of *Statement*—the lowest level of reflection quality used by only five of the 32 teams in the study. In fact, Team B8 used the *Statement* reflection level almost exclusively (14 of 15 statements), indicating that their retros were nearly devoid of strategies to help them improve their team process. Fourth, and in stark contrast, Teams A1-2 and A1-5 stand out for their extensive use of the *Justified Strategy* reflection level. Just under half of Team A1-2's statements (20 of 43) were at this level, while 40% of Team A1-5's statements (17 of 42) were at this level. Interestingly, Team A1-2 was the only team to use the *Critiqued Strategy* reflection level.

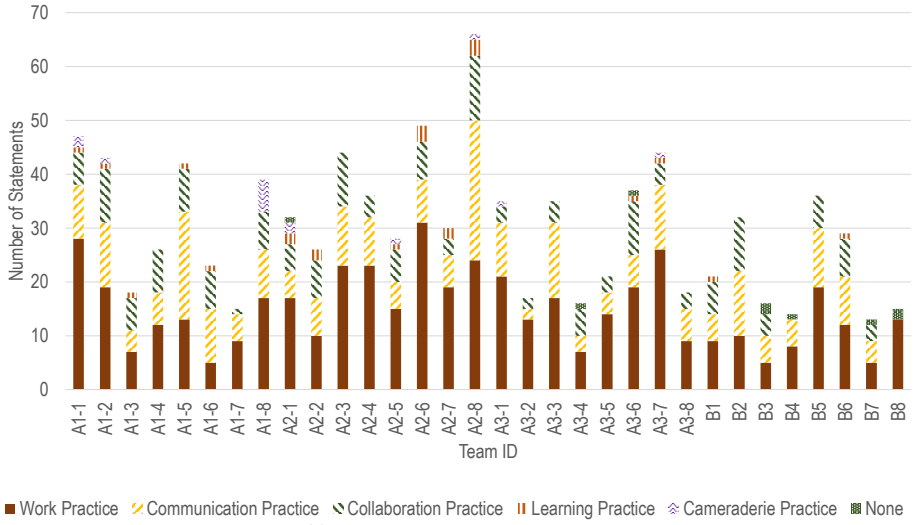
5 DISCUSSION

Our study provides insight into what student software teams focus on in their retrospectives, how conscientiously they reflect within those retrospectives; and how retrospectives differ across teams.

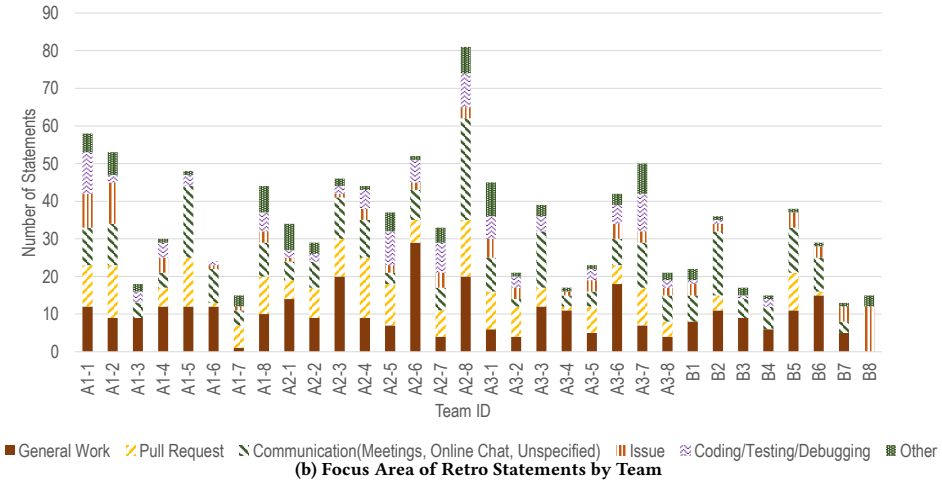
5.1 Reflection Content

With respect to retro content, our key finding is that retros focus overwhelmingly on three areas of concern: *work practices*, *communication practices*, and *collaboration practices*. Together, these three concerns make up 95% of teams' retro statements. That teams' retrospectives tend to focus on their work, collaboration, and communication practices is unsurprising; all of these are central to the effective functioning of a team, and potential targets for improvement through reflection. In contrast, *team camaraderie* was relatively rare in retrospectives. While it plays an important role in the functioning of a team [44], we speculate that teams may not see a need to reflect on it because it is established through the day-to-day interactions of the team.

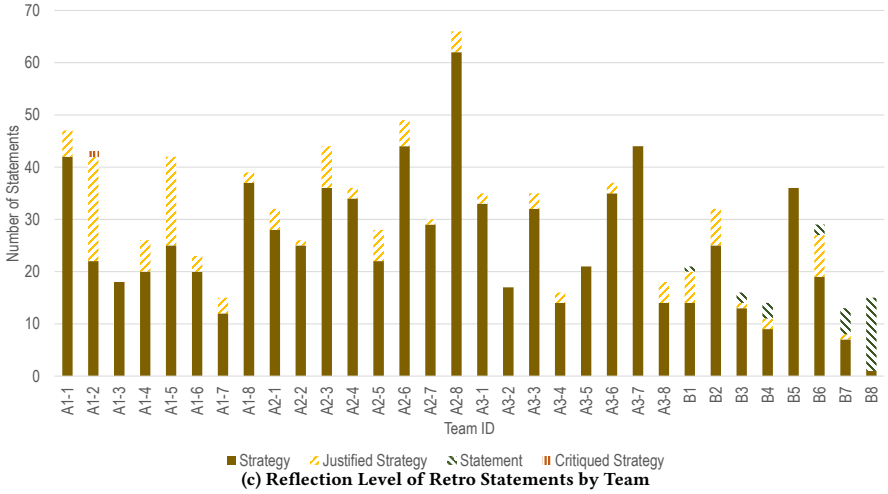
Our *Focus Area* coding scheme further illuminates the focus of teams' retros by linking concerns to specific activities and artifacts within the software development lifecycle. In this analysis, we found that the greatest proportion of teams' retrospective statements (29%) focused on general work practices (e.g., "Start tasks early," "Update team on your progress"). However, substantial portions of teams' concerns were also concentrated on activities related to communication (24%), pull requests (19%), coding/testing/debugging (10%),



(a) Concern of Retro Statements by Team



(b) Focus Area of Retro Statements by Team



(c) Reflection Level of Retro Statements by Team

Figure 2: Retro Content and Quality by Team

and issues (9%). These suggest key parts of the software development lifecycle where students may benefit from additional training or coaching.

These results align to various degrees with the quantitative results of three related studies reviewed in Section 2.2. In Gestwicki and McNely's [15] analysis, between 60 and 80% of retro statements focused on *collaboration*. This is markedly higher than the share of *Collaboration Practice* found in our study's retros; however, Gestwicki and McNely's definition of "collaboration" appears also to encompass our *Work Practice* and *Communication Practice* categories, suggesting that their results are indeed consistent with ours. Likewise, Lehtinen et al. [26] found that 49% of retro statements focused on *Implementation Work*, closely aligning with our finding that 50% of retro statements focused on *Work Practices*. In the study of Dingsøyer et al. [13], statements classified as *Process* were most common, constituting 38% of all statements.

Interestingly, two of the related studies [13, 26] further classified retro statements based on whether they were *positive* (i.e., identified things that were working) or *negative/change required* (i.e., identified things that weren't working or needed to be changed). Our data also allows us to perform such a breakdown, since each retro statement in our corpus responded to one of three prompts ("continue," "change/start," and "improve/stop"). In the two related studies and ours, a clear majority of retro statements—between 60% and 71%–were classified as "negative/change," suggesting that, in retros, teams tend to focus more on things that are not going well.

5.2 Reflection Quality

As mentioned in the Related Work section, prior studies of student reflections have found they are often superficial (e.g., [9, 17, 31, 39]). The results of this study bear this out: just 13% of teams' retro statements included an explicit rationale for why a practice should be started, stopped, or continued. This suggests that it would be helpful for software engineering instructors to experiment with pedagogical strategies (e.g., prompts) for improving the depth of students' reflections. Follow-up studies similar to this one could then be conducted to determine the extent to which those strategies are effective. It may also be helpful to find effective ways both to provide timely formative assessments related to the depth of students' reflections, and to incentivize deeper reflection.

5.3 Team Variance

Our team-by-team analysis of retro content and quality revealed not only wide variance in the number of retro statements produced by each team, but also statistically significant differences in teams' categorical distributions of content and quality. On one extreme, two teams' retrospectives were nearly devoid of the type of reflection that could lead to actionable changes in the subsequent sprint. On the other extreme, over 40 percent of two of the teams' retros consisted of strategies that were grounded in a rationale.

We can identify two factors that might account for these stark differences in teams' reflection. First, *individual differences* in teams, and the members that compose those teams, may have led them to engage in more, less, shallower, or deeper reflection, and to focus on different things in their retros. For example, some teams may have had team members with prior experience performing retros, or team members who were naturally self-reflective. Likewise, differences in the software development tasks that teams focused on may have led them to focus on different things in their retros. Second, we observe that while students received instructions on how to perform retros, they did not receive explicit instruction on how to engage in higher levels of reflection, or on what to focus on in retros. Without such instruction, perhaps we might have anticipated wide variance in the topics on which teams focused, and in their levels of reflection. We suspect that, with additional guidance, modeling, and coaching (see, e.g., the framework of [2]), student teams could learn to engage in deeper reflection in their retros, leading, perhaps, to more effective team processes and better team outcomes.

6 THREATS TO VALIDITY

6.1 Internal Threats to Validity

One internal threat to the validity of this research arises from studying team retrospectives based solely on teams' written summaries of their retros. It could be the case that teams' written summaries do not capture the depth and breadth of the reflections in teams' face-to-face retro meetings. To explore this possibility, we reviewed the video recordings of a 19% convenience sample of teams' retrospective sessions—a total of 14 recordings submitted by six teams in Course B. Based on this review, we determined that teams' face-to-face retro discussions closely aligned with the written summaries they handed in. In other words, we failed to find evidence that statements made in teams' retro sessions were at a deeper level of reflection than the statements included in their written summaries.

A second threat to internal validity stems from subtle differences in the retrospective prompts used in the two courses involved in the study. In Course A, the three prompts were *start, stop* and *continue*. In course B, the prompts were *what went well, what we would like to improve*, and *changes we plan to implement in the next milestone period*. Although the two sets of prompts are similar, differences in the prompts could have led to differences in student teams' retro statements. A third threat to internal validity relates to potential differences in students' prior experience with retrospectives, which may have led to differences in the content and quality of teams' retros. To mitigate this threat, we ideally would have have collected data on students' prior experience with retrospectives, and used that data to help interpret our results.

A fourth threat to internal validity relates to differences in the student populations enrolled in Courses A and B. As Table 2 indicates, the average age of students in Class A was roughly four years younger than in Class B. In addition, there may have been additional differences in the populations—for example, differences in prior industry experience—that we failed to document. Given the observed and potential differences between the student populations studied, caution should be exercised in interpreting the results.

6.2 External Threats to Validity

An external threat to the validity of this study stems from its limited scope: We considered four offerings of two courses at two North American universities. The courses in the study could not capture the range of individual, team, and software project differences that are present in the general population, including team size, cultural and linguistic characteristics of teams, and differences in retro

formats. While we have made an effort to compare our results to the existing literature, care must be exercised in any attempt to generalize our findings beyond the population we studied.

7 SUMMARY AND FUTURE WORK

This paper has presented an analysis of the content and quality of 72 retros performed by 32 teams in four offerings of undergraduate software engineering courses. We have found that teams reflected most often on their work, communication, and collaboration practices. An analysis of the quality of teams' retros reflections showed that only 13% provided justification for a strategy to be stopped, continued, or started. In addition, the content and quality or retro statements varied significantly by team.

Our results, which align with the general themes of previous studies of retros, suggest at least two directions for future research. First, while our study shed light on the content and quality of retrospective reflections, future work should identify links between retrospective reflections and concrete changes in teams' processes and products. Second, given the wide variance observed in teams' retrospectives, future work should explore the impact of pedagogical interventions aimed at getting students to reflect deeply on their processes, implement meaningful changes, and document the impacts of those changes on team processes and products.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grant nos. 1915196 and 1915198.

REFERENCES

[1] R. R. Adisurya, H. B. Santoso, S. Fadhilah, and O. Lawanto. 2020. Information visualization of metacognitive skills during the software development process based on an adapted engineering design metacognitive questionnaire. *Journal of Physics: Conference Series* 1566, 1 (June 2020), 012078. https://doi.org/10.1088/1742-6596/1566/1/012078 Publisher: IOP Publishing.

[2] Yanti Andriyani, Rashina Hoda, and Robert Amor. 2017. Reflection in Agile Retrospectives. In *Agile Processes in Software Engineering and Extreme Programming (Lecture Notes in Business Information Processing)*, Hubert Baumeister, Horst Lichter, and Matthias Riebisch (Eds.). Springer International Publishing, Cham, 3–19. https://doi.org/10.1007/978-3-319-57633-6_1

[3] Jeffry Babb, Rashina Hoda, and Jacob Nørbjerg. 2014. Embedding Reflection and Learning into Agile Software Development. *IEEE Software* 31, 4 (2014), 51–57. https://doi.org/10.1109/MS.2014.54

[4] Andrew Begel and Beth Simon. 2008. Novice software developers, all over again. In *Proceedings of the Fourth international Workshop on Computing Education Research (ICER '08)*. ACM, New York, NY, USA, 3–14. https://doi.org/10.1145/1404520.1404522

[5] Andrew Begel and Beth Simon. 2008. Struggles of new college graduates in their first software development job. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08)*. ACM, New York, NY, USA, 226–230. https://doi.org/10.1145/1352135.1352218

[6] Elizabeth Bjarnason and Björn Regnell. 2012. Evidence-Based Timelines for Agile Project Retrospectives – A Method Proposal. In *Agile Processes in Software Engineering and Extreme Programming (Lecture Notes in Business Information Processing)*, Claes Wohlin (Ed.). Springer, Berlin, Heidelberg, 177–184. https://doi.org/10.1007/978-3-642-30350-0_13

[7] Matt Bower and Debbie Richards. 2006. Collaborative learning: Some possibilities and limitations for students and teachers. In *23rd Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education: Whos Learning*. Sydney University Press, Sydney, Australia, 79–89.

[8] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa Place: United Kingdom Publisher: Hodder Arnold.

[9] Christopher N Bull and Jon Whittle. 2014. Supporting reflective practice in software engineering education through a studio-based approach. *IEEE software* 31, 4 (2014), 44–50.

[10] Håkan Burden and Jan-Philipp Steghöfer. 2019. Teaching and Fostering Reflection in Software Engineering Project Courses. In *Agile and Lean Concepts for Teaching and Learning*. Springer Nature Singapore Pte Ltd., Singapore, 231–262.

[11] Aino Corry. 2020. *Retrospectives Antipatterns*. Addison-Wesley, Boston.

[12] Derby, E. and Larsen, D. 2006. *Agile Retrospectives: Making Good Teams Great*. Pragmatic Bookshelf, Dallas.

[13] Torgeir Dingsøyr, Marius Mikalsen, Anniken Solem, and Kathrine Vestues. 2018. Learning in the Large - An Exploratory Study of Retrospectives in Large-Scale Agile Development. In *Agile Processes in Software Engineering and Extreme Programming (Lecture Notes in Business Information Processing)*, Juan Garbajosa, Xiaofeng Wang, and Ademar Aguiar (Eds.). Springer International Publishing, Cham, 191–198. https://doi.org/10.1007/978-3-319-91602-6_13

[14] Tore Dybå, Neil Maiden, and Robert Glass. 2014. The Reflective Software Engineer: Reflective Practice. *IEEE Software* 31, 4 (2014), 32–36. https://doi.org/10.1109/MS.2014.97

[15] Paul V. Gestwicki and Brian J. McNely. 2013. Empirical Evaluation of Periodic Retrospective Assessment. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 699–704. https://doi.org/10.1145/2445196.2445399 event-place: Denver, Colorado, USA.

[16] Orit Hazzan. 2002. The reflective practitioner perspective in software engineering education. *Journal of Systems and Software* 63, 3 (2002), 161–171.

[17] Christopher Hundhausen, Phill Conrad, Olusola Adesope, Ahsun Tariq, Samir Sbai, and Andrew Lu. 2023. Investigating Reflection in Undergraduate Software Development Teams: An Analysis of Online Chat Transcripts. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 743–749. https://doi.org/10.1145/3545945.3569790

[18] Robert F. Dugan Jr. 2011. A survey of computer science capstone course literature. *Computer Science Education* 21, 3 (2011), 201–267. https://doi.org/10.1080/08993408.2011.606118 arXiv:https://doi.org/10.1080/08993408.2011.606118

[19] An Ju, Adnan Hemani, Yannis Dimitriadis, and Armando Fox. 2020. What Agile Processes Should We Use in Software Engineering Course Projects?. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 643–649. https://doi.org/10.1145/3328778.3366864

[20] Norman L. Kerth. 2001. *Project Retrospectives: A Handbook for Team Reviews*. Dorset House, New York.

[21] K. Krippendorff. 1980. *Content analysis: an introduction to its methodology*. Sage Publications, Beverly Hills.

[22] Deanna Kuhn and David Dean, Jr. 2004. Metacognition: A bridge between cognitive psychology and educational practice. *Theory into practice* 43, 4 (2004), 268–273.

[23] Marilyn Lamoreux. 2005. Improving agile team learning by improving team reflections [agile software development]. In *Agile Development Conference (ADC'05)*. IEEE, IEEE, Denver, CO, 139–144.

[24] Richard Layton, Matthew Ohland, and Hal R Pomeranz. 2007. Software for Student Team Formation and Peer Evaluation: CATME Incorporates TeamMaker. In *2007 Annual Conference & Exposition*. ASEE, Honolulu, Hawaii, 12–1286. https://peer.asee.org/software-for-student-team-formation-and-peer-evaluation-catme-incorporates-team-maker

[25] Madhavi Ledalla. 2020. *Retrospectives for everyone: Powerful metaphors for effective retrospectives*. Notion Press, Chennai, India.

[26] Timo O. A. Lehtinen, Juha Itkonen, and Casper Lassenius. 2017. Recurring opinions or productive improvements—what agile teams actually discuss in retrospectives. *Empirical Software Engineering* 22, 5 (Oct. 2017), 2409–2452. https://doi.org/10.1007/s10664-016-9464-2

[27] Äli Leijen, Kai Valtna, Djuddah A.J. Leijen, and Margus Pedaste. 2012. How to determine the quality of students' reflections? *Studies in Higher Education* 37, 2 (March 2012), 203–217. https://doi.org/10.1080/03075079.2010.504814 Publisher: Routledge.

[28] Marc Loeffler. 2018. *Improving Agile Retrospectives: Helping Teams Become More Efficient*. Addison-Wesley, Boston.

[29] Dastyni Loksa, Lauren Margulieux, Brett A. Becker, Michelle Craig, Paul Denny, Raymond Pettit, and James Prather. 2021. Metacognition and Self-Regulation in Programming Education: Theories and Exemplars of Use. *ACM Trans. Comput. Educ.* TBD, TBD (dec 2021), 1–30. https://doi.org/10.1145/3487050 Just Accepted.

[30] Robert Cecil Martin. 2003. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, USA.

[31] Roger McDermott, Mats Daniels, Åsa Cajander, Mats Cullhed, Tony Clear, and Cary Laxer. 2012. Student reflections on Collaborative Technology in a globally distributed student project. In *2012 Frontiers in Education Conference Proceedings*. IEEE, Seattle, WA, 1–6. https://doi.org/10.1109/FIE.2012.6462410

[32] Janet Metcalfe and Arthur P. Shimamura (Eds.). 1994. *Metacognition: Knowing about knowing*. The MIT Press, Cambridge, MA, US. https://doi.org/10.7551/mitpress/4561.001.0001 Pages: xiii, 334.

[33] Moon, J.A. 2004. *Reflection in Learning and Professional Development*. Routledge-Falmer, New York.

[34] Thomas O Nelson and Louis Narens. 1994. Why investigate metacognition. *Metacognition: Knowing about knowing* 13 (1994), 1–25.

[35] Yen Ying Ng, Jędrzej Skrodzki, and Maciej Wawryk. 2020. Playing the sprint retrospective: a replication study. In *Advances in Agile and User-Centred Software Engineering: Third International Conference on Lean and Agile Software Development, LASD 2019, and 7th Conference on Multimedia, Interaction, Design and Innovation, MIDI 2019, Leipzig, Germany, September 1–4, 2019, Revised Selected Papers 3*. Springer, Leipzig, Germany, 133–141.

[36] M.J. O'Grady. 2012. Practical Problem-Based Learning in Computing Education. *Trans. Comput. Educ.* 12, 3 (July 2012), 10:1–10:16. https://doi.org/10.1145/2275597.2275599

[37] Daniela Pedrosa, Mario Madureira Fontes, Tânia Araújo, Ceres Morais, Teresa Bettencourt, Pedro Duarte Pestana, Leonel Morgado, and José Cravino. 2021. Metacognitive challenges to support self-reflection of students in online Software Engineering Education. In *2021 4th International Conference of the Portuguese Society for Engineering Education (CISPEE)*. CISPEE, Lisbon, Portugal, 1–10. https://doi.org/10.1109/CISPEE47794.2021.9507230

[38] James Prather, Brett A. Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What Do We Think We Think We Are Doing? Metacognition and Self-Regulation in Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. ACM, New York, NY, USA, 2–13. https://doi.org/10.1145/3372782.3406263

[39] Julia Prior, Samuel Ferguson, and John Leaney. 2016. Reflection is Hard: Teaching and Learning Reflective Practice in a Software Studio. In *Proceedings of the Australasian Computer Science Week Multiconference* (Canberra, Australia) *(ACSW '16)*. Association for Computing Machinery, New York, NY, USA, Article 7, 8 pages. https://doi.org/10.1145/2843043.2843346

[40] Adam Przybyłek, Marta Albecka, Olga Springer, and Wojciech Kowalski. 2021. Game-based Sprint retrospectives: multiple action research. *Empirical Software Engineering* 27, 1 (Oct. 2021), 1. https://doi.org/10.1007/s10664-021-10043-z

[41] Adam Przybyłek and Dagmara Kotecka. 2017. Making agile retrospectives more awesome. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, Prague, Czech Republic, 1211–1216. https://doi.org/10.15439/2017F423

[42] D. Schön. 1983. *The reflective practitioner: How professionals think in action*. Basic Books, New York.

[43] J.E. Sims-Knight and R.L. Upchurch. 1998. The acquisition of expertise in software engineering education. In *FIE '98. 28th Annual Frontiers in Education Conference. Moving from 'Teacher-Centered' to 'Learner-Centered' Education. Conference Proceedings (Cat. No.98CH36214)*, Vol. 3. IEEE, Tempe, AZ, USA, 1302–1307 vol.3. https://doi.org/10.1109/FIE.1998.738679

[44] Tuckman, B.W. 1965. Developmental Sequence in Small groups. *Psychological Bulletin* 63, 6 (1965), 384–399. https://doi.org/10.1037/h0022100

[45] Maciej Wawryk and Yen Ying Ng. 2019. Playing the sprint retrospective. In *2019 federated conference on computer science and information systems (fedcsis)*. IEEE, Leipzig, Germany, 871–874.

[46] Tingting Yang and Ikseon Choi. 2023. Reflection as a social phenomenon: a conceptual framework toward group reflection research. *Educational technology research and development* 71, 2 (2023), 237–265. https://doi.org/10.1007/s11423-022-10164-2 Place: New York Publisher: Springer US.