Guiding Principles for Assessing Software Engineering Teams

Bowen Hui
Computer Science
University of British Columbia
Kelowna, Canada
bowen.hui@ubc.ca

***Abstract*—This research-to-practice full paper presents guiding principles for developing assessments based on our experience in teaching the software engineering capstone course over the last twelve years. Software engineering courses are central to computer science and engineering programs. To provide students with an authentic learning experience, teams of students work on realistic projects that help them apply theoretical concepts to develop practical skills. Challenges arise with increasing class sizes and limited teaching resources. Despite these constraints, educators must intentionally design assessments that align with learning theories that promote deeper learning opportunities and support lifelong learning. In this work, we report on our experience designing and adapting the assessments used in our software engineering capstone course over the past twelve years. We reflect on our approach by aligning the evaluation strategies to learning theories such as behaviorism, constructivism, and social constructivism. This research-to-practice paper discusses the practical implications behind different assessment strategies in the face of large class sizes and presents guiding principles for developing assessments for software engineering team projects.**

***Index Terms*—Capstone project, authentic assessments, behaviorism, constructivism, social constructivism, large classes**

I. Introduction

Software engineering courses are central to computer science and computer engineering programs. To provide students with an authentic experience, students are often placed into teams to work on real-world projects that help them connect theoretical concepts to practical skills. Some programs also run a software engineering capstone course where students pool the knowledge and skills they acquired throughout their degree and apply them to complete a graduating project. Capstone courses enable students to gain proficiency in writing efficient, maintainable, and scalable code in a large project that runs for the full academic year. In some cases, capstone projects also connect students with industry clients to solve real-world problems. Such capstone projects serve as authentic assessments at the end of the student's degree and prepare them before they enter the industry workforce.

As student enrolments grow, it becomes increasingly difficult to manage a large number of teams while keeping the teaching resources unchanged. Two main issues arise in such cases – grading student assessments and monitoring the teams for ongoing project progress and behavioral interactions that indicate rewarding and risky actions. Certain assessments afford deeper and more authentic learning experiences for students, which are typically harder to administer in large classes. The complexity of these issues increases if the projects involve varying clients and requirements, which is common in capstone courses. Due to the limited visibility into team processes, educators are faced with the challenge of assessing individual contributions fairly while also considering the effort exhibited by the team weighted against the complexity of the projects. Different situations make grading individual work non-trivial such as highly skilled students who do not cooperate or coordinate well with their teams, average students who demonstrate great leadership in the team, and low-performing students who are not given any opportunities to do meaningful work. Learning about the underlying dynamics in each team can be very time-consuming for the teaching staff, causing many obstacles in monitoring team progress in real time. Lastly, since individual behaviors influence overall team dynamics, assessment becomes more complex due to the subjectivity and ethical considerations involved. Different evaluators may have varying perspectives on the quality of work, team dynamics, and individual contributions. This subjectivity can make it challenging to provide fair and consistent evaluations. This paper explores the following research questions:

RQ1: What are the necessary capstone evaluation components?
RQ2: How do various assessment activities align with learning theories that promote deep learning experiences?
RQ3: What are the assessment challenges encountered due to an increase in enrollment?

Over the past twelve years, our experience from teaching the software engineering capstone course has led us to explore different learning theories such as behaviorism, constructivism, and social constructivism. Although behaviorism is often aligned with traditional teaching techniques and passive learning while constructivism and social constructivism are associated with student-centered teaching and active learning strategies [1], [2], [18], many courses still use a mix of these theoretical perspectives. We review these theories and their applications in designing capstone courses in Section II. Section III elaborates on the course context at our university and discusses the assessment changes made over the years. We will further identify the challenges encountered due to the growth in the student population, discuss the necessary adaptations

made to these assessments, and reflect on the lessons learned based on our experience. Section IV aligns specific aspects of the course design and assessment strategies to the learning theories. Our discussion reveals that many assessments have a constructivist or social constructivist approach, but the underlying systemic nature of grade assignment follows a behaviorist perspective. Section V identifies guiding principles that emphasize a social constructivist approach to developing assessments for software engineering team projects. Our hope is to help other educators be intentional about their assessment approach in designing authentic and fair assessments for grading student team projects.

II. Background on Learning Theories

Behaviorism is a learning theory based on the principle of stimulus and response [14], [19], [22]. Behaviorists believe that learning is a function of the conditions in the environment and they focus on how people form habits and exhibit observable behaviors based on these conditions. Positive reinforcements are given to desirable behaviors while negative reinforcements are given to undesirable behaviors. In a behaviorist classroom, "the teacher is in control of what needs to be taught, how it will be taught and what evidence of behavioral change needs to be produced" (p.27) [2]. Teachers design learning activities with the appropriate stimulus to elicit student responses that demonstrate the desired behavior. Behaviorists view students as passive learners who soak up information given to them without concerns about what might be happening in the minds of the learners.

In response to behaviorism, a number of cognitive theories emerged. During this time, "research suggests that learners – from a very young age – make sense of the world, actively creating meaning while reading texts, interacting with the environment, or talking with others" (p.3) [24]. Under the theory of *constructivism* [15], this interaction facilitates knowledge construction, where learners build meaning based on prior experiences and refine ideas through interacting with the environment. In this view, students bring their own perspectives to interpret and shape their understanding of the concepts taught. The philosophy of treating learning as a process in the mind is referred to as *cognitivism*, and it has many overlaps with constructivism where knowledge construction extends the concepts in the mind.

Students under the constructivist viewpoint create a personalized experience where knowledge is negotiated and learners do not automatically absorb the information presented to them. In contrast to a behaviorist classroom, a constructivist classroom shifts the focus of learning from the teacher to the students. Constructivism promotes activities that enable students to actively engage in the learning process, such as interactive exercises and student-led discussions. In contrast to cognitivism, pure cognitivist classrooms focus on concept development achievable through teacher-led instruction, while constructivist classrooms depend on the learners to interpret the content they are given to deepen their grasp of the knowledge. Since constructivists believe that knowledge is built upon one's prior experience, constructivist classrooms emphasize what students already know and help them build deeper learning experiences.

Social constructivism focuses on the collaborative aspect of constructivism where learning is situated in society and culture [6], [21]. Learning is viewed as a social process, where knowledge is constructed based on social interactions and the relationships people form in their society. Students rely on others to help create their building blocks, and learning from others helps them construct their own knowledge and reality. Social constructivist classrooms provide the context for students to engage in authentic learning experiences measured by the norms of their communities. Activities are often situated in collaborative and interactive settings where students practice the concepts they need to learn.

Several researchers reported their ideas and experiences from aligning learning theories to teaching and learning in practice. We found two cases where the authors suggested using constructivist learning activities to promote better learning experiences in online learning environments [23] and in designing ePortfolio capstone projects [17]. One closely related work is a survey of 200 papers on computer science capstone courses conducted in 2011 [11]. The review found some instructors revised their courses to use more constructivist techniques, such as providing an authentic project, creating ill-structured and complex problems to promote problem-based learning, deliberately inserting conflicting requirements, requiring students to perform software lifecycle and process tasks, employing studio-based learning to provide formative feedback, requiring student reflections, presenting new material through a just-in-time learning approach, and focusing on the process rather than the product.

More broadly, a general assessment framework was proposed that maps learning theories to learning levels in Bloom's taxonomy [4] and example assessment activities [12]. In this framework, behaviorism was mapped to lower-order thinking skills in the taxonomy (such as remembering and understanding), whereas cognitivism, constructivism, and theories on metacognition were mapped to mid- and higher-order thinking skills (such as applying, analyzing, evaluating, and creating). This alignment suggests while behaviorist thinking does not promote or exercise higher-order thinking skills, it is necessary for building foundational, lower-order thinking skills.

In a general learning context, some researchers propose shifting away from traditional assessments completely because they typically follow a behaviorist perspective to alternative assessments that better align with constructivist thought [1]. The authors propose several alternative assessments that offer deeper learning opportunities, but it is unclear how well they scale to large classes. These authors raise very interesting points about the role of assessments serving as instruments for improving education and questioning how one can ensure students learn for the sake of learning rather than for the sake of passing an exam or a course. From this perspective, one may argue that the impact of an assessment is only as effective as allowed by the learner's mindset regardless of the

intended design. We return to this point when we reflect on our assessment choices in Section IV.

III. Course Context and Design Changes

We offer a four-year undergraduate degree in computer science. As part of the degree, one of the required courses is a capstone software engineering project course that is expected to be taken in the student's final year of study. The purpose of this course is to serve as a holistic assessment of the student's cumulative knowledge of computer science where they apply their collective knowledge to solve a real-world problem. The official calendar description of our course is as follows: "A capstone project requiring team software development for an actual client. Students must produce comprehensive reports and deliver presentations."

In this course, students work in small teams to elicit technical requirements from an industry client, design a software solution, develop a working prototype with extensive features, and present the accomplishment to the client. Akin to a doctoral candidacy exam, our capstone course was designed to help students make connections across the different subject matter from their degree, demonstrate their ability to learn and troubleshoot technical problems, as well as learn new industry processes and standards. This project course runs for two 13-week semesters in the academic year, spanning approximately 8 months of the calendar year. The capstone project serves as an authentic assessment at the end of their degree and prepares the students before they enter the workforce.

Despite having different instructors teach the course over the last twelve years, the general course design and evaluation approach have remained rather stable. This stability is not a positive indicator as the number of students has grown from less than 10 students in the class (grouped into two or three teams) to over 100 students (split into 20+ teams). The course has been taught in a relatively similar way despite the increase in the instructor-to-student ratio. The main difference is the added support for undergraduate and graduate teaching assistants (TAs) to help with assessments. However, there is also no guarantee about the quality of the TAs assigned to the course so TA training has become an added concern. Thus, we need to rethink project assessments to accommodate this under-resourced situation.

Traditionally, every student team is assigned a unique external client and project. Clients typically come from a non-technical background, although we occasionally have technical clients as well. In most years, clients are entrepreneurs, small business owners, or managers from large companies. On rare occasions, university faculty members also acted as clients. Teams between 3 and 5 students were formed at the beginning of the project by balancing skill sets and their elicited preferences. Due to the university's relationship with external clients, there is a strong emphasis on project success and the quality of the project outputs. At the same time, we highlight the importance of the teamwork process and professionalism when working with the instructor and the client. Students are advised to act as consultants for clients by giving them technical advice, anticipating potential obstacles, providing technical options, and building prototypes that realize the intended project goals. Overall, we wish the students to be accountable to their team and their clients while ensuring their grades fairly reflect the work they produce individually.

The general design of this course follows a social constructivist paradigm where students work in teams and learn from each other, collaborating and learning by doing together to solve a real-world problem. What they learn is immediately experienced by applying to their projects and adapting the techniques to tackle the problems at hand. Students are taught industry-standard tools and processes and are asked to use them throughout the capstone project. When they make mistakes or use the tools in suboptimal ways, they gain feedback from the teaching staff and, in some cases, also from the clients. By setting clear expectations and giving feedback on student deliverables, the client establishes industry norms within the classroom setting.

The rest of this section reviews changes in our capstone software engineering project course by sampling the course design details from 2012, 2017, and 2023. We chose these years as they were the first, middle, and last course offerings of the twelve years of teaching experience reported here. Each year, the capstone course instructors meet to discuss what went well and brainstorm ideas to tackle issues that arose in the class. The discussion in this section describes changes in the course design over the years and the accompanying rationale for those changes.

A. Course Learning Outcomes

We designed course learning outcomes (CLOs) aimed at integrating the soft and hard skills needed to transition smoothly into a software development role in the industry. In 2012, the CLOs were not explicitly stated in the syllabus. As seen from the list provided below, the CLOs from 2017 focused on the software development process:

1) Apply software engineering principles in a real-world project
2) Research the needs and interests of a particular target group
3) Determine key elements in complex issues, problems, and questions
4) Collect, synthesize, and evaluate reliable information or data from relevant sources
5) Manage, mitigate, and resolve conflicts
6) Anticipate likely problems, consider unanticipated outcomes, propose the means by which resolutions may be attained
7) Gain a deeper understanding of key design and implementation issues
8) Acquire experience working with clients and professionals in the industry

Due to a lack of student interest, CLO 2 is typically not done well. Rather than focusing on developing this outcome further, we opted to remove it because it was covered in a prerequisite course on human-computer interaction. Another removal was CLO 5 because most teams did not encounter conflicts. In

the rare cases when conflicts arose, the instructor was heavily involved in helping the teams and individual members work through their challenges. Lastly, CLO 8 was removed due to large class sizes that required us to match one client to multiple student teams. As a result, clients no longer worked closely with students, although they were available to give input, evaluate student deliverables, and provide a source of external validation of the student's work. The remaining CLOs were subsequently synthesized into the following in 2023:

1) Apply software engineering principles to work in a team on a non-trivial project
2) Gain hands-on experience with design and implementation issues
3) Adopt industry standards and tools in a programming project
4) Collect, synthesize, and evaluate information or data from reliable sources
5) Troubleshoot technical problems and propose alternative feasible solutions

According to Bloom's taxonomy, these CLOs focus on mid- to higher-order thinking skills where students synthesize, apply, and create new knowledge. From a course design perspective, it is difficult to tease apart which aspect of the capstone project or which assessment maps to each of these CLOs because, arguably, the entire process of working toward building a successful capstone project addresses these CLOs collectively. Another challenge is, given a particular assessment, to what extent does each team member achieve the associated CLO? The nature of the capstone team projects makes this mapping extremely difficult to accomplish accurately and reliably. Although we do not have an ironed-out solution to this problem, we are working towards developing an assessment approach for it. Below, we discuss the assessments used over the years that were designed to help students meet the CLOs.

B. Major Milestones and Deliverables

A total of 4 team deliverable dates were set approximately once every two months throughout the eight-month course. In 2012, these milestones included a project plan, a design document and test plan, a prototype presentation, and a final project presentation and document. These team deliverables were designed to resemble real-world business reports. During this time, class size was small and teams had the luxury of submitting a draft of their reports to obtain formative feedback from the instructor and subsequently improve their work before the official reports were graded. Modeled after core ideas of mastery learning [5], this process gave teams personalized feedback with multiple chances to succeed.

Soon after, we felt the students did not take the feedback seriously from the teaching staff during the milestones where they presented their prototypes. Many students felt their opinions and approach were better, while others felt the suggested changes required too much work. For these reasons, we replaced the prototype presentation in the second semester with two peer testing sessions evenly spaced out in the second semester. Dedicated class time was allocated to run these sessions. In a peer testing session with a team of 4 students, two members would run a usability study with their classmates as participants and the other two would participate in the usability study for other teams. After two rounds, students in the team would switch roles from running the study to participating in one, and vice versa, and repeat the study for two more rounds. At the end of the session, each team had to write up the usability results of their study and produce a list of prioritized features and bugs to fix as a result of the study. Students have generally been extremely well-prepared for these peer testing sessions and have found them to be very engaging and helpful in getting critical feedback from their peers. In summary, the major milestones were changed to a project plan, a design document, two peer testing sessions, and a final deliverable in 2017.

In hindsight, using written reports as an assessment combines behaviorist and constructivist approaches. On the one hand, the reports facilitate team management so the teaching staff can better assess project progress. On the other hand, the writing includes an explanation of the system design, which is an articulation of how well the students understand what they have learned and what they are supposed to be doing. In cases where students take the graded feedback to correct possible misunderstandings either in future versions of the report or in implementing the software that adheres to those changes, we may conclude that these students have successfully applied what we intended for them to learn. However, if students are only writing the reports for the sake of completing a course milestone and they never refer to the feedback or the contents of the reports, then the students themselves have a behaviorist mindset and the class has failed to teach them the importance of those concepts. Evidence of students writing separate sections and stitching their work together without proofreading each other's work also points to this pattern.

These five milestones have persisted through to 2023, although the size of the documents required has significantly decreased. Operationally, while creating a project plan was a helpful tool to scope out the project requirements and have students consider how well their skills meet those requirements, the inevitable changes of these requirements quickly outdated the project plan making it not very useful in the second semester. Similarly, the design document in the early years resembled the deliverable from a waterfall development process where a full system design was built (e.g., use case diagram, entity-relationship diagram, user interface mockups). Over the years, we shifted to a more agile approach and encouraged incremental design and builds. The peer testing sessions no longer required reports, but the deliverables were replaced with logging issues for project tracking in the code repositories. Finally, the final project deliverable requirements have decreased via an imposed maximum page length. To better assess individual contributions, the final report also became an individual report in 2023.

A major change in the presentation format occurred when class sizes increased. Previously, each team would present in front of the whole class. In general, this format encouraged

students to be more involved in each other's work and improved team communication and morale. Unfortunately, class time was insufficient to accommodate all the presentations for large classes. Thus, the presentation format changed to video presentations only. While some teams still delivered high-quality, cohesive presentations, many teams opted for each member to record their own clips and stitch them together as the final submission. We believe using a video format for presentations reverts our students to a behaviorist mindset.

C. Team Roles

Team roles are commonly used in the workplace to increase individual accountability and team success [3]. Initially, team roles were designed to facilitate effective teamwork in the course. In 2012, we asked teams to identify an individual who would take on each of the following roles: Team Lead, Lead Assistant, Client Liaison, and Secretary. Our view was that different students could contribute to the project in technical and non-technical ways because their technical competencies have been assessed in previous courses in the program. However, asking students to pick a role and keeping them "in their place" seems counterproductive to student learning; students often choose tasks that they are good at and avoid doing work that requires extra effort. Furthermore, we saw power dynamics between a high-performing student who assumed the Team Lead role and others who were expected to follow the lead's directions and felt they could not speak up in the team meetings. Having dedicated team roles may also create inequitable situations as the literature reports that minorities are often pressured to take on less technical tasks in project work [7], [9], which we also encountered in our experience.

To create a less hierarchical structure, we introduced multiple leading roles. In 2017, the roles evolved to include Project Manager, Technical Lead, Test Lead, and Client Liaison (which assumes secretarial tasks). Due to changes in technology and industry trends, many projects had more demands on system administration and operations. For this reason, we replaced the Test Lead with an Integration Lead, and everyone was responsible for high-quality testing. Unfortunately, the challenges mentioned about having team roles persisted. For example, roles such as the Project Manager and the Client Liaison were seen as less technical, and students in those roles often took on more of the "chores" in the project. Note that we did not collect data directly related to the student roles and their participation level, and therefore, we cannot make conclusions that certain roles fostered higher levels of participation or that certain roles were more suited for a constructivist or social constructivist learning environment.

Since some students took many prerequisite courses online in recent years (because of COVID-19) and quality control was less strict due to increasing class sizes, we felt that a significant portion of low-performing students got by in team projects but did not do their fair share of the work. Thus, we removed role assignments in order to place more emphasis on individual competencies. In 2023, we adopted a fully agile process where teams were asked not to have assigned roles because we wanted to emphasize the importance of technical skill assessment in this course where everyone is expected to be competent in programming, testing, and reviewing code. Although we still saw power dynamics between high-performing and low-performing students within a team, we believe this approach enables students to contribute more equally and is more appropriate for facilitating a social constructivist environment for student learning.

D. Weekly Interaction and Assessments

In 2012, the instructor would meet with every team weekly to review progress and troubleshoot problems. Since there were very few teams, the instructor could spend 30 to 60 minutes with each team weekly. This format provided a collaborative and constructivist environment where the instructor acted as a facilitator to assist students in solving their problems. No formal rubrics were used to assess weekly performance at this time.

Teams were asked to have hour-long meetings with their client on a weekly basis as well, although in the early years, this was not enforced. All the team members were expected to attend these client meetings, however, some students would miss these meetings occasionally and, in rare instances, the client may cancel meetings. These meetings aimed for the teams to present their ideas, prototypes, and progress to the client and ask for clarifications where needed. Following a social constructivist viewpoint, clients used these meetings to communicate their expectations and feedback to the students, helping them understand industry norms.

To ensure consistency and transparency between the instructor and the students, the instructor began to employ a rubric in 2016 to evaluate individuals on their professionalism (e.g., collegiality, attitude, following processes, punctuality, mature responses, work ethics), planning and management (e.g., time logging, time management, task focus, plan development, plan evaluation, plan adjustment), and the work completed (e.g., number of tasks completed, number of tests written, number of tasks reviewed, number of hours spent). This rubric was used during the weekly in-class meetings which gave students immediate feedback on their progress as well as areas of improvement. Although this rubric intended to follow a social constructivist approach by incorporating industry standard practices (such as professional behavior, time management, and work done), the idea of grading students based on their weekly demonstrated behaviors is naturally behaviorist because the grades function as a reward for an observed behavior.

Furthermore, the majority of the points in this weekly assessment are allocated to the amount of work completed individually, which is measured based on individual work output rather than individual learning. Nonetheless, students were rewarded bonus marks if they did extra work, giving them the flexibility to focus on constraints from other courses or personal lives. This flexibility helps create a less stressful academic environment for students.

As the class size increased, the amount of time dedicated to meeting each team decreased. This forced the instructor meetings to change from collaborative problem-solving to progress update reports only. In 2020, the class increased to 20 teams and we decided to change the client model to have two to three student teams matched to every client. Since the number of contact hours did not increase, each client met multiple teams simultaneously. This created confusion for the clients as they had a hard time keeping track of the teams, individuals, and team prototypes. This arrangement also imposed scheduling constraints and increased unanticipated communication difficulties (see Section III-E3 for further details below). For the instructor-to-student meetings, the weekly rubric was no longer feasible so alternative approaches were needed (we discuss individual evaluations further in Section III-E2 below). The increase in class sizes has made these meetings less about knowledge construction and more about progress and end-product evaluation.

E. Evaluation Criteria

In 2012, 70% of the course grade was based on group work, such as weekly activity reports, major requirements and design documents, a test plan, intermediate prototypes produced, and final project products. The remaining 30% was based on individual work, divided evenly into individual contributions, peer evaluations, and individual presentations. Since we observed different levels of contributions across the individuals in the teams, the future years weighted the team marks and the individual marks more evenly. Moreover, due to some disrespectful behavior exhibited toward the client, a client component was added in all subsequent years.

In 2017, the team component was worth 35% of the grade, the individual component was worth 45%, and the client component was worth 20%. These percentages fluctuated slightly from year to year. For example, the client component ranged from 15% to 25% depending on the specific year.

In 2023, the course evaluation criteria shifted to 55% for the team component, 40% for the individual component, and 5% for the client component. One major change was reducing the client component because the student teams no longer held weekly client meetings and, therefore, did not have to manage client expectations or learn to communicate professionally with clients. Instead, clients were asked to evaluate submitted work twice in the course. A second major change is adding more weight to the team component to measure their weekly progress and collaboration process, which gave more of an emphasis on the learning process rather than the deliverables produced by the students.

None of the learning theories reviewed gave any indication of how each of these components should be weighed. However, we believe the learning theories provide insights into how each component should be assessed. We discuss the assessment details of each component in more detail.

1) Team Component: In all years except 2023, the team component is measured based on the major course deliverables mentioned in Section III-B. The instructor evaluated all the team deliverables to calculate a base score. Peer evaluations were used to create a weight multiplier to the team scores so that the resulting individual mark could be lower or higher than the base score. We provide more context on the peer evaluations in Section III-F below.

In 2023, we dedicated a portion of the team component to assess the team's collaboration process. In a software engineering context, this includes conducting project management activities regularly, reporting on project progress effectively, (re)prioritizing tasks while considering changing requirements and project constraints, coordinating with the team, and following a list of standard programming practices. The introduction of this criterion is meant to make students more accountable to the team while ensuring they are engaging in proper team conduct and professional coding practices. Assessing students on their collaboration process also expressed the importance of industry norms and practices over a team's ability to build software that has many working features but does not conform to standard conventions and is not easy to maintain. This approach encourages students to focus more on the learning aspects that are central to social constructivism and have the grades reflect those aspects of student learning.

2) Individual Component: In principle, this component represents the amount of work an individual contributes to the team. Before 2020, this assessment was largely subjective based on the weekly in-class interactions with each team and student measured by the rubric mentioned in Section III-D. Since students were measured based on their ability to apply knowledge learned from other classes to the project, their ability to evaluate online resources and subsequently overcome technical obstacles, and how much work they completed, this rubric assessed higher-order learning outcomes which aligned closely with constructivism and social constructivism. With small classes, instructors felt it was still possible to accurately discern how much each student was working and check code contributions as needed to evaluate individual work fairly.

In the years when peer evaluations were used, a small portion of the individual component also included a grade that assessed the quality of the submitted evaluations to ensure the content reflected critical thinking, which supports the ideas promoted by constructivism and social constructivism.

In 2019, a self-reflection activity was introduced to help individuals improve their team communication at the end of each milestone. This activity asked everyone to identify their strengths and areas of improvement based on their experience in the last milestone. When they completed their form, all the team members were asked to review each other's responses and discuss as a group potential strategies to strengthen one or two problematic areas. Sadly, many teams did not take this activity seriously and refused to complete the forms during the dedicated class time. Rather than forcing students to do self-reflection just for the grade, we discontinued the activity and sought alternative ways to improve team communication and individual self-regulation.

Since 2020, class sizes increased and automated scripting tools have been used to garner statistics about individual con-

tributions with limited success. At first, simple code metrics such as line and file contributions were used. These metrics were normalized based on team contributions and weighed relative to the expected weekly contributions. However, many students disagreed strongly about using these metrics to directly calculate their grades and found ways to game the system by collectively inflating their code contributions as a team. Clearly, students felt certain coding behaviors were not good indicators of their learning and productivity.

Subsequently, these metrics were used to inform student progress and not their grades directly. In 2023, code metrics not only measured lines of code but also the number of tests added and code reviews done for other team members. The idea was to enforce good software development practices so that the programs they build would be properly tested and reviewed by others. Unfortunately, most students did not spend time writing meaningful tests and code reviews, but they did what was needed to get them marks in the rubric. Furthermore, counting the number of completed features, tests, and code reviews made the students think only quantity matters. Overall, there is still resistance and fear that this approach weighs too heavily on the quantity of work produced. Ideally, the individual component could incorporate a systematic way that measures the quality of the work produced and reflect an individual's learning gains over the course of the project.

3) Client Component: Starting in 2013, each team received a client evaluation based on identical surveys completed by their client in the middle of and at the end of the project. Since they had weekly meetings with the teams, clients were expected to report on the team's professionalism, teamwork ability, project progress, and confidence in receiving a high-quality final deliverable. These questions were written as 5-point Likert scale questions where we converted the responses into a numeric score to provide a client grade. The client feedback survey also had open-ended questions about team strengths and areas of improvement. With a small number of teams, this process was very manageable and ensured we met client expectations for the course project. Students also got immediate feedback on their professional conduct and work quality based on industry expectations. This process promotes a social constructivist learning environment for students.

For teams that struggled with client management, the midpoint survey helped identify the problems and gave students a second chance at improving the process for the second semester of the project. This approach adopts some of the core ideas of mastery learning and provides added incentive for students to improve in the course.

In 2023, this client interaction was reduced due to a new approach we experimented with. Modeled after a hackathon competition, in this course offering, only 4 clients were available and teams chose the project they wished to work on. Subsequently, clients gave feedback on the project submissions in the middle and at the end of the project. Arguably, this client feedback still provides a sense of industry norms and external validation for the students, but we have unfortunately lost the feedback on professional conduct from the clients. For details on the hackathon client model, please refer to [10].

F. Peer Evaluations

After each deliverable submission, students were asked to complete a peer evaluation of their team. When answers were released to the team, students typically played nice and generally did not report problems. However, when answers were kept confidential with the teaching staff, we saw some cases where students expressed concerns about team dynamics when they arose. We felt this approach was effective in helping the teaching staff identify and manage team conflicts early.

As mentioned earlier, peer evaluations are compiled and converted into a numeric multiplier to weigh the base score of the team deliverable. In the early years, this weight ranged from 0 to 1.0, and from 2017 onwards, this weight ranged from 0 to 1.2 so that students have a chance to receive bonus marks for extraordinary work if the weight is over 1.0. In 2023, peer evaluations were conducted weekly and the accumulative average peer scores were used as the weighted multiplier for the team deliverables. Using peer evaluations in this way gives students a chance to explain discrepancies in workload distributions among the team members which supports the goal of situating their learning in a social context.

IV. Reflections

A. RQ1: Necessary Evaluation Components

Based on our discussion of the various assessment activities from Section III, we highlight the ones that foster deeper learning opportunities. In terms of course design, we find the following aspects promote a social constructivist environment and enable students to engage in experiential learning:

- A team project that assesses the collaboration process in addition to the deliverable quality
- Regular peer evaluations where members provide constructive feedback to each other
- A real-world industry project that offers external validation and, ideally, regular client interaction
- Problem-solving sessions with the teaching staff to assist with knowledge construction and promotion of a constructivist and social constructivist mindset

Furthermore, students who graduate from our program should have mastered all the competencies of a "computer science undergraduate student". To better prepare them for industry-quality work, the course should also give students opportunities to engage in industry-standard tools and practices. The following points summarize these ideas:

- Measuring students' work against all relevant competencies (a departure from enforcing team roles)
- Using industry tools and following industry standard practices in collaborating in the software development process

Certain assessments have successfully evaluated effective teamwork, professionalism, and depth of understanding. The following items should be used for this purpose:

- Weekly check-ins with a rubric to evaluate preparedness, planning, management, and professionalism

- Peer testing sessions to give feedback and suggestions on development deliverables
- Written or oral explanations on project inner-workings
- In-person client presentations on project functionality

Written reports resembling business documents can be used as project deliverables, but the administrative requirements should be altered to foster constructivist thinking.

B. RQ2: Alignment with Learning Theories

Section III discussed each assessment and how they align with the learning theories presented in Section II. Generally, most assessments were developed from a constructivist or social constructivist perspective. However, their effectiveness varied due to three key issues. First, certain mechanisms in grading students made students value behaviors over enhancing their knowledge. Many students prefer exhibiting behaviors associated with rewards (i.e., high scores) even if it means not agreeing with the purpose of the assignments, over working on the activities for the sake of learning and self-improvement. We saw this behavior in the written reports, weekly coding behaviors, and video presentations. When students do not buy into the learning goals of the assessments, they complain about the fairness of the assessments and devote their effort to alternative activities instead.

Second, increasing class sizes has made our approach become less constructivist and more behaviorist because there is less time available to engage with students in meaningful discussions. This happened with in-class contact, client communication, project presentations, and individual assessments. At the heart of the problem is the limited resources available to properly support student learning. Although TAs can provide some relief to the resources problem, adequate training is necessary to ensure TAs have the technical competency and teaching experience to support the students. Attempting to solve a problem by introducing another challenge does not truly remove the problem in the end.

Third, some students come with a behaviorist mindset and they just want to know what they need to do to get a good grade in the course. For example, despite knowing that feedback is important to learning, many students ignore feedback from others and most students do not take time to provide constructive feedback for their peers. Also, when faced with an open-ended project, rather than embracing the freedom, some students struggle with the uncertainties and would rather be told what they need to do instead. Certain students may have been ingrained in behaviorist paradigms which makes them more resistant to classrooms that celebrate constructivist approaches.

C. RQ3: Added Challenges for Large Classes

To summarize, several challenges have been introduced due to large class sizes that prevented the successful implementation of our capstone course:

- Number of available industry partners to serve as clients
- Increasing time commitment in managing client expectations and minimizing client confusion
- Reduced contact time per student, resulting in shallow progress updates and adopting code metrics as a proxy to progress and levels of individual contribution
- Lack of time to engage collaboratively with students in overcoming their technical obstacles and team conflicts
- Harder to change student mindset due to reduced interaction
- Instructors had less frequent student and team interactions since TAs serve as surrogates for the instructor check-ins
- Inadequate TA training to provide TAs with the technical expertise and teaching experience to support the teams
- Tradeoffs between team size and number of teams; larger teams involve more complex dynamics and management, smaller teams are more productive but having too many teams makes it difficult to provide reasonable guidance
- Increasing difficulty in monitoring team progress closely and ensuring no individual student feels lost in the project

Overall, the biggest impact is that these challenges resulted in removing constructivist and social constructivist aspects from the course. This implies that large classes are more suited to be run as a behaviorist classroom.

V. Guiding Principles in Large Classes

Using our findings to the research questions, we formulate the following principles to guide the development of assessments for software engineering teams in large classes.

1) Course evaluation components should consist of a client component to allow for external validation, a team component to emphasize the importance of teamwork, and an individual component so students can demonstrate their accountability to the team and client. How much each component is worth should reflect the pedagogical objectives of the course. Having all three components helps situate learning in a social constructivist context.
2) The client component should be based on the interaction experience between the client and the students, as well as the client's satisfaction with the prototype. The goal of this component is to help students understand industry norms.
3) The team component should evaluate how well the students function as a team, how well the team follows industry-standard processes, and the quality of their required deliverables. The grades for the deliverables may be individualized via the use of peer evaluations. The purpose of this component is largely to teach students how to work in teams effectively, while exercising industry-standard practices.
4) The individual component should evaluate how well students demonstrate a list of "necessary" computer science competencies. This evaluation may need to be separate from the team deliverables, for example, through additional individual tests. This component ensures each student has the basic competencies required for a graduating computer science undergraduate student. Instructors in under-resourced classrooms must design these assessments carefully so that students have time to demonstrate the necessary skills and the teaching staff has enough time and technical expertise to evaluate those outcomes reliably.

5) Deliverables should be meaningful for the students so they are more likely to take ownership of the work. For example, computer science students who wish to get a programming job are unlikely to value writing business documents. Thus, the knowledge we wish the students to demonstrate in those documents should ideally be incorporated into the software development process instead (e.g., in the README documentation). Making the deliverables relevant for students increases the value of constructivist ideals.
6) When used to identify discrepancies in workload distributions, peer evaluations should protect student confidentiality so that the responses are not released to students without explicit instructor approval. This way, students have a safe platform to voice their concerns. This approach helps make outcomes more fair in a social constructivist learning environment. Although we did not experiment with this, it should also be possible to teach students how to provide constructive and critical feedback in the peer evaluations as well.
7) Short, weekly surveys can serve as quick, effective temperature checks in large classes (e.g., [8], [13], [16], [20]). Questions should help identify issues in team dynamics, such as the fairness in workload distribution. The purpose here is to provide early alerts to the instructor in case interventions are required. This approach alleviates the administrative overhead of monitoring teams in large classes.
8) When project progress data is collected periodically (e.g., via self-reported logs, project management reports, regular check-ins), teams should be taught to use this information to self-regulate their own learning by enacting, evaluating, and monitoring their progress [25]. This process must be repeated and the data must be updated regularly. Although we did not experiment with this idea, we believe this is a necessary next step in helping students take an active role in their own learning. Likewise, the same can be said for data collected on individual skills (e.g., via self-reflections, code repository data) where students can be taught to self-regulate their individual learning process [25]. Releasing the data back to the students closes the loop in the learning process by helping students take ownership of their education and become more effective learners.

VI. Conclusions

This work results from our experience in teaching the software engineering capstone course over the past twelve years. Our goal is to inform instructors and course designers on how the structure and implementation of assessments can influence student learning. We analyzed a variety of assessments from a learning theory perspective, with an emphasis on changing requirements mandated in under-resourced classrooms. A key finding revealed that, while many assessments were designed with constructivism and social constructivism learning opportunities in mind, the implementation and the student mindset can deter the benefits of those assessments. We hope this work fosters a discussion around these challenges so we can together develop effective strategies to improve the teaching and student learning experience.

References

[1] S. Ahmad, N. Sultana, and S. Jamil. Behaviorism vs constructivism: A paradigm shift from traditional to alternative assessment techniques. *Journal of Applied Linguistics and Language Research*, 7(2):19–33, 2020.

[2] B. Bates. *Learning Theories Simplified: And How to Apply them to Teaching*. Sage Publications Ltd., 3rd edition, 2023.

[3] R. Belbin and B. V. *Team Roles at Work*. Routledge: London, 3rd edition, 2022.

[4] B. Bloom. *Taxonomy of educational objectives: Cognitive and affective domains*. New York: David McKay, 1956.

[5] B. Bloom. Learning for mastery. *UCLA Evaluation Comment*, 1:1–12, 1968.

[6] J. W. (ed.). *Culture, Communication, and Cognition: Vygotskian Perspectives*. Cambridge: Cambridge University Press, 1985.

[7] K. Grindstaff and M. Mascarenhas. 'no one wants to believe it': Manifestations of white privilege in a stem-focused college. *Multicultural Perspectives*, 21(2):102–111, 2019.

[8] N. Heyl, E. Baniassad, and O. Ola. Team harmony before, during, and after covid-19. In *Proceedings of the ACM SIGPLAN International Symposium on SPLASH-E*, pages 52–61, 2022.

[9] N. Holmes, G. Heath, K. Hubenig, S. Jeon, Z. Kalender, E. Stump, and E. Sayre. Evaluating the role of student preference in physics lab group equity. *Physical Review Physics Education Research*, 18(010106), 2022.

[10] B. Hui, S. Hodge, and D. Samra. Transforming the client relationship to support large capstone classes. In *Proceedings of the 54th IEEE Frontiers in Education (FIE)*, 2024.

[11] R. D. Jr. A survey of computer science capstone course literature. *Computer Science Education*, 21(3):201–267, 2011.

[12] A. Millet and c. E. Din˙ Synthesizing a conceptual framework for assessment in online education. In *Proceedings of International Technology, Education and Development Conference (INTED)*, pages 7893–7899, 2023.

[13] M. Modell. Iterating over a method and tool to facilitate equitable assessment of group work. *International Journals of Designs for Learning*, 4(1):39–53, 2013.

[14] I. Pavlov. *Conditioned Reflexes: An Investigation of the Physiological Activity of the Vertebral Cortex*. London:Oxford University Press, 1927.

[15] J. Piaget. *Construction of Reality in the Child*. London:Routledge & Kegan Paul, 1957.

[16] K. Presler-Marshall, S. Heckman, and K. T. Stolee. Identifying struggling teams in software engineering courses through weekly surveys. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education (SIGCSE)*, page 126–132, 2022.

[17] R. Prokopetz. A reflection upon capstone eportfolio projects and their alignment with learning theories. *International Journal of ePortfolio*, 12(1):1–15, 2022.

[18] D. Singhal. Understanding student-centered learning and philosophies of teaching practices. *International Journal of Scientific Research and Management*, 5(2):5123–5129, 2017.

[19] B. Skinner. Reinforcement today. *American Psychologist*, 13:94–99, 1958.

[20] R. Tucker and C. Reynolds. The impact of teaching models, group structures and assessment modes on cooperative learning in the student design studio. *Journal for Education in the Built Environment*, 1(2):39–56, 2006.

[21] L. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA:Harvard University Press, 1978.

[22] J. Watson. *The Ways of Behaviourism*. New York:Harper & Brothers, 1928.

[23] M. Weegar and D. Pacis. A comparison of two theories of learning: Behaviorism and constructivism as applied to face-to-face and online learning. *E-Leader*, pages 1–20, 2012.

[24] S. Wilson and P. Peterson. Theories of learning and teaching: What do they mean for educators? Technical report, Best Practices: NEA Research. National Education Association, 2006.

[25] B. Zimmerman. Becoming a self-regulated learner: An overview. *Theory Into Practice*, 41(2):64–70, 2002.