2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #51649

Experiential Agile Learning: Transforming Software Engineering Education through Capstone Projects

Dr. Georgios Eleftherakis PhD, The Pennsylvania State University

Dr. George (Georgios) Eleftherakis is an Associate Professor of Computer Science at Penn State Brandywine, where he also serves as Program Coordinator for the Computer Science program. He holds a B.Sc. in Physics from the University of Ioannina, Greece, and an M.Sc. and Ph.D. in Computer Science from the University of Sheffield, UK. With over 25 years of academic experience, Dr. Eleftherakis has established a strong research portfolio at the intersection of nature-inspired computing, complex adaptive systems, and formal methods for software engineering. His work focuses on bio-inspired models for self-adaptive and self-organizing systems, particularly in the context of Internet of Things (IoT) and distributed sensor networks. He has pioneered approaches such as EDBO, a bio-inspired overlay network for IoT, and explored their application in health monitoring systems for chronic disease management. He also coined the term ”agile formal methods”. In addition to his technical contributions, Dr. Eleftherakis is deeply committed to advancing gender equality in computing. He has served in leadership roles within ACM-W and ACM-W Europe for over 15 years, promoting diversity and inclusion through policy development, mentoring, and community-building initiatives. His service includes chairing the ACM Council of European Chapter Leaders and contributing to strategic efforts that empower women in technology across Europe and beyond. Dr. Eleftherakis has authored 85+ refereed publications, edited multiple books, and organized international conferences. His research and teaching excellence have been recognized with awards such as the University of Sheffield Senate Award for Sustained Excellence in Learning and Teaching.

Experiential Agile Learning: Transforming Software Engineering Education through Capstone Projects

1 Introduction

The rapid pace of technological advancement and the complexity of modern software systems require graduates to master both technical skills and professional collaboration. Empirical studies document a persistent misalignment between university curricula and industry needs [1, 2]. While core technical content is typically well covered, graduates often lack hands-on experience and critical soft skills (communication, teamwork, and adaptive problem solving) hindering employability and signaling the need for experiential, industry-aligned pedagogy [1, 3].

Our program includes a dedicated Software Engineering course earlier in the curriculum that introduces students to fundamental development processes and agile concepts. The capstone course examined in this paper is a separate, culminating experience in which students apply and extend these concepts in a realistic, industry-aligned environment. Serving as the bridge between academic instruction and professional practice, the capstone provides an authentic context in which students design, implement, test, and deploy software systems using agile frameworks.

Preparing industry-ready software engineers therefore demands more than strong technical instruction; it benefits from a principled blend of learning theories that shape problem-based, collaborative, and authentic experiences [4, 5]. In practice, integrating complementary perspectives (e.g., Kolb's experiential cycle, constructivism, situated learning, and cognitive apprenticeship) can align coursework with the rapid iteration and uncertainty that characterize modern development.

Agile methods such as Scrum, well-established in industry since the early 2000s, continue to play a central role in contemporary software development. Their growing educational adoption offers a pragmatic vehicle for aligning instruction with real-world practice. Evidence indicates that agile frameworks (e.g., Scrum, XP) support engagement, process comprehension, and team effectiveness, while iterative practices foster adaptability in the face of evolving requirements [6, 7, 8]. Additional studies and reviews report improved student commitment, teamwork, and learning outcomes in agile-informed courses, and highlight benefits of industry–academia collaboration for employability [8, 9, 10, 11]. At the same time, challenges—including a learning curve and resistance to non-traditional approaches—are well noted, and can be mitigated through a prior Software Engineering course, explicit scaffolding,

case-based activities, and sustained mentorship [12].

The capstone course at our university is designed to bridge academic learning and industry practice by simulating a real software development environment. The course emphasizes Scrum-based teamwork in which students iteratively design, implement, test, and deploy functional software, applying engineering principles in authentic contexts. We operationalize these outcomes through an ABET/Bloom-aligned, multi-rater assessment framework and a planned longitudinal comparison.

This paper presents a theory-driven instructional design and reports early evidence-based observations from its implementation. While longitudinal data collection is ongoing, the pedagogical model itself represents a substantial and transferable contribution to engineering education practice. To address the documented gaps, this paper advances an industry-aligned pedagogy that integrates agile practice with a principled blend of learning theories and implements it within the capstone structure. Our contributions are threefold: (i) the design and rationale of this integrated framework, (ii) an ABET/Bloom-aligned, multi-rater assessment approach that integrates rubric-based instructor evaluation with peer and stakeholder assessments, and (iii) a comparative, longitudinal study plan linking course outcomes to early-career indicators. We hypothesize measurable gains in problem solving, teamwork, and adaptability relative to traditional courses.

The remainder of the paper details the course structure (Section 2), theoretical underpinnings (Section 3), learning outcomes (Section 4), assessment and study design (Section 5), and implications for engineering education (Section 6).

2 Course Structure and Methodology

Our capstone course employs a project-based learning methodology that integrates agile software development principles, with a particular focus on Scrum. The course is designed to mirror real-world software engineering environments and emphasizes active learning, collaboration, and iterative problem-solving. Students engage in team-based projects, which simulate industry practices and challenges.

The key aspects of the pedagogical approach are as follows:

- Agile-Based Learning: Students work in Scrum teams, engaging in sprint planning, daily stand-ups, and iterative development, allowing them to experience firsthand how agile methodologies operate in practice.
- Hands-On, Project-Centric Approach: The course is structured around a capstone project, where students collaborate with a client, brainstorm and decide an idea, and then design, implement, test, and deploy a functional software product, with a clear focus on creating market-ready solutions.
- Industry Alignment: The curriculum is designed to ensure that students develop skills highly valued by employers, such as adaptability, problem-solving, teamwork and communication, and technical expertise.

- Continuous Feedback and Iteration: Through regular stakeholder meetings, peer assessments, and iterative phases, students refine their software products based on continuous and real-time feedback.
- Interdisciplinary Skill Development: The course fosters a holistic skill set, combining technical expertise with critical thinking, project management, and communication skills to equip students for complex professional challenges.
- Hybrid Pedagogical Model: The course employs a hybrid model that integrates agile-based learning, project-centric approach, industry alignment, validated assessment framework, and comparative and longitudinal evaluation.

3 Theoretical Framework

Our course structure incorporates an original blend (amalgam) of established learning theories, more specifically, constructivism, situated learning, Kolb's experiential learning cycle, and cognitive apprenticeship, aiming to enhance learning outcomes. The innovative approach is briefly explained and demonstrated in the following subsections.

3.1 Kolb's Experiential Learning Cycle

Kolb's Experiential Learning Cycle is a model that favors the process of learning through experience, emphasizing hands-on, reflective, and iterative learning processes. It usually consists of four stages: Engaging in a new concrete experience; Reflecting on the experience and observing what happened; Forming new ideas or modifying existing concepts based on the reflections; Applying the new ideas to see (actively experiment) if they work in practice.

In our capstone project it usually applies in the following way; A team is tasked with developing a new application, after brainstorming and drafting an initial description of the project, and then it starts gathering requirements and creating initial prototypes. After the first (prototyping) sprint, the other teams peer-review the progress. If they notice something that as users of the system do not like, or they find it not complying with the project description, and/or some features are not working as expected, they provide feedback to the team. The team discusses these issues and gathers and analyzes feedback from the initial testers. Based on the feedback they brainstorm and come up with new design principles and feature improvements and decide and plan future actions.

They apply the new design principles and plans in the next sprint. The team develops a revised version of the app and tests it with users again. This time, the feedback is hopefully more positive, indicating whether changes were effective.

By cycling through these stages, the teams continuously improve, self-organize, and become more effective, and at the same time they continuously improve the project, learning from each iteration and applying new insights to enhance the final product. This iterative process is at the heart of our capstone's project experiential learning cycle, developing leadership skills and improving team dynamics.

The course follows an agile development cycle that is structured into distinct phases, mirroring real-world practices:

- Planning and Requirements Gathering: Students begin by scoping the project and analyzing requirements, which sets the foundation for development.
- Literature Review: Students research existing technologies and methodologies, ensuring they are well-informed before design and implementation.
- Prototyping and Development: In iterative sprints, students design and refine the software product, focusing on continuous improvement and adaptability.
- Testing and Deployment: Rigorous testing ensures the software's reliability and performance before deployment.
- Final Presentation and Submission: Students document their findings, present their software, and provide a comprehensive review of the development process.

3.2 Constructivism

Constructivism is a learning theory that posits that knowledge is actively constructed by learners through experience rather than passively absorbed. It emphasizes problem-solving, critical thinking, and the learner's ability to build upon prior knowledge in meaningful ways.

The course follows a constructivist approach by encouraging students to actively engage in hands-on, real-world software development projects instead of merely studying theoretical concepts. Through Scrum-based sprints, students construct their own understanding of software design principles by iterating through coding, testing, and debugging phases.

In our capstone course, rather than being given a step-by-step guide, students are assigned a project that mimics an industry scenario (e.g., developing a software product for a client). They must research, design, and implement solutions, integrating feedback from mentors, industry partners, and their peers. This iterative learning process aligns with constructivist principles as students build their knowledge through problem-solving and collaboration.

3.3 Situated Learning

Situated learning theory [13, 14] suggests that knowledge is best acquired in authentic, real-world contexts where learners interact with experienced practitioners. Learning is most effective when it occurs within communities of practice, where novices gradually become experts through participation.

Our capstone course situates learning within a professional software development context by integrating industry mentorship, project-based work, and real-world problem-solving. Students collaborate in agile teams, mirroring professional software engineering environments, ensuring they gain relevant, applicable skills.

Instead of working on artificial programming exercises, students in the capstone course engage with real-world clients or simulated stakeholders. They participate in sprint planning meetings,

stakeholder discussions, and product demonstrations, just as they would in an industry setting. By working alongside industry professionals and receiving feedback, students move from novice to proficient practitioners, aligning with situated learning principles.

3.4 Cognitive Apprenticeship

Cognitive apprenticeship is a learning theory and instructional model developed in 1989 that extends the traditional concept of apprenticeship (learning a trade through direct interaction with a master) to cognitive skills by emphasizing scaffolding, guided practice, and situated learning[15]. The approach helps learners internalize expert knowledge, problem-solving strategies, and professional skills by actively engaging in real-world tasks with the support of mentors or instructors.

In a traditional classroom, students might read about agile methodologies in textbooks and answer theoretical questions about Scrum processes. However, in our course, following a cognitive apprenticeship model, students work directly within an agile development environment, observing and imitating expert practices, receiving continuous mentorship, and reflecting on their experiences.

The instructor models a sprint planning session by conducting one in class, coaches students as they run their own planning meetings, scaffolds their learning by providing structured templates for backlog refinement, encourages articulation by having students explain their sprint goals and decision-making process, promotes reflection through retrospective meetings, encourages exploration by letting students independently refine their development processes. That is how cognitive apprenticeship is applied in Software Engineering in our course. The course integrates cognitive apprenticeship principles through its project-based, mentorship-driven, and iterative learning structure (see figure 1).

By embedding cognitive apprenticeship into the capstone course, students experience the software development lifecycle as it happens in industry, ensuring they graduate with not just theoretical knowledge but also practical, applied expertise in agile methodologies. This strengthens the educational rigor of the course and justifies its longitudinal tracking of career impact.

3.5 Integrated Impact of Learning Theories

By embedding constructivist, situated, and experiential learning principles, the capstone course deliberately bridges academic preparation and industry expectations. These theories are enacted through the mentoring strategies outlined in Figure 1, aligned with agile practices and lightweight artifacts. This integration enables students to learn in authentic contexts, externalize their reasoning, and iteratively refine their solutions, supporting observable growth in problem framing, design justification, collaboration, and professional communication. Each activity is tied to concrete, reviewable evidence in deliverables and demonstrations, forming the basis for the transparent evaluation framework detailed in Section 5.

Compared to canonical Scrum-based capstone implementations such as the one by Mahnič [8], which primarily demonstrate the feasibility and value of structuring a capstone around Scrum iterations and practices, our approach extends beyond process adoption in three ways. First, it

combines constructivism, situated learning, Kolb's cycle, and cognitive apprenticeship into an integrated, operational learning design. Second, it incorporates a Bloom-referenced, ABET-aligned assessment architecture that computes outcome attainment using triangulated evidence from instructor, peer, and stakeholder evaluations. Third, it includes a longitudinal and comparative study plan to examine professional readiness. Together, these elements position the course as a theory-driven, assessment-explicit evolution of Scrum-based capstones rather than a replication of prior practice [8].

4 Learning Outcomes and Skills Development

As described from the beginning, the main aim of the course is to prepare the students to enter the workforce, that is why it is designed to primarily foster a diverse and robust skill set comprised of the most highly sought after by employers in the software industry. According to the National Association of Colleges and Employers (NACE) [16], the following skills are consistently ranked as top employer priorities (all of them more than 65% of employees require them) and are particularly in demand, and we will mention them (in the usual order of priority) while at the same time justify how this innovative and sophisticated amalgam of learning methodologies achieves them:

- Problem-Solving Skills: The iterative nature of agile development helps students develop structured problem-solving strategies to tackle complex issues. Studies have shown that agile methodologies enhance problem-solving abilities by encouraging iterative and adaptive approaches [17], [18].
- Teamwork and Collaboration: The Scrum methodology emphasizes strong teamwork, fostering collaboration, and effective communication within diverse teams. Research indicates that agile practices significantly improve team dynamics and collaboration [19], [20].
- Written Communication: The emphasis on documentation, reports, and final presentations cultivates clear and concise technical writing and communication skills. Agile methodologies promote regular documentation and communication, which enhances written communication skills [21], [22].
- Initiative and Work Ethic: Capstone projects encourage students to take initiative, meet deadlines, and demonstrate professionalism. Agile practices foster a proactive work ethic and responsibility among students [23], [24].
- Technical Proficiency: Hands-on development with industry tools ensures students are proficient in coding and system design. Agile methodologies provide practical experience with industry-standard tools and technologies, improving technical proficiency [8], [25].
- Verbal Communication: Regular presentations, meetings, and stakeholder interactions develop verbal communication abilities. Agile practices involve frequent verbal communication, enhancing students' ability to articulate technical concepts [26], [27].
- Adaptability: Agile development's flexible structure teaches students how to manage

changing requirements and uncertainties. Research shows that agile methodologies improve adaptability and resilience in dynamic environments [28], [29].

- Analytical and Quantitative Skills: Students engage in data-driven decision-making and analytical problem-solving. Agile practices encourage analytical thinking and quantitative analysis [30], [31].
- Detail Orientation: Testing, debugging, and meticulous documentation instill precision and accuracy in their work. Agile methodologies emphasize attention to detail through continuous testing and refinement [32], [33].

By emphasizing these competencies, The capstone course ensures students are well-prepared to enter the workforce with a diverse and robust skill set.

5 Evaluation and Assessment

The evaluation strategy for the capstone course is intentionally multi-dimensional, reflecting the complex blend of technical, interpersonal, and professional competencies required in modern software engineering practice. The assessment framework integrates agile-informed observations, structured rubrics, peer and stakeholder feedback, and longitudinal tracking to provide a comprehensive picture of student learning and performance. This section details the multi-dimensional evaluation model, the assessment components, the roles of evaluators, the ABET-aligned Bloom's Taxonomy rubric system, and the long-term study design used to measure the course's impact.

5.1 Multi-Dimensional Evaluation Framework

The capstone course employs a multi-dimensional evaluation framework that captures student performance across three core domains: technical competence, teamwork and interpersonal skills, and business-oriented professional skills. This structure reflects the realities of industry practice, where engineers must integrate technical expertise with effective collaboration and communication.

Technical competence is assessed through the quality of software design artifacts, implementation correctness, testing rigor, and deployment readiness. Teamwork and interpersonal skills are evaluated through participation in agile practices, peer collaboration, communication effectiveness, and contributions to team goals. Business and professional skills are assessed through stakeholder interactions, requirements analysis, presentation clarity, and the ability to justify design decisions in terms of user and client needs. Together, these dimensions provide a holistic view of student readiness for professional software engineering roles.

5.2 Assessment Components and Weighting

Student performance is evaluated through several complementary components that collectively measure both individual and team-based learning outcomes:

- **Project Deliverables (50%):** Evaluation of design documents, implementation quality, testing completeness, and final deployment of the software product.
- **Presentations and Demonstrations (20%):** Assessment of oral communication, clarity of technical explanations, and professionalism during sprint reviews and final presentations.
- **Peer Assessments (20%):** Structured evaluations of teamwork, reliability, communication, and contribution to team objectives.
- **Team Participation and Activities (10%):** Engagement in agile practices, responsiveness to feedback, and adherence to Scrum roles and responsibilities.

This distribution ensures that no single dimension dominates the evaluation and that both technical and professional competencies are meaningfully represented.

5.3 Evaluators and Assessment Roles

Assessment in our course is conducted through a triangulated model that incorporates instructor evaluation, external stakeholder review, and peer assessment. This approach increases reliability, reduces evaluator bias, and captures dimensions of performance that would be difficult to observe from a single perspective.

The primary instructor evaluates technical deliverables, documentation quality, and adherence to engineering standards using structured rubrics. Departmental faculty and industry stakeholders participate in sprint reviews, stakeholder meetings, and final presentations, providing external perspectives on communication, professionalism, and product viability. Peer evaluations complement these assessments by capturing collaboration quality, individual accountability, and team dynamics—elements that are often most visible to fellow team members. Together, these evaluators contribute to a comprehensive and authentic assessment of student performance.

5.4 Validated Assessment Framework

The course employs an ABET-aligned rubric system grounded in the Revised Bloom's Taxonomy [34] and supported by assessment practices widely used in engineering education research [35]. Each learning outcome is mapped to cognitive levels such as *Apply*, *Analyze*, *Evaluate*, and *Create*, ensuring that assessment captures both foundational understanding and higher-order reasoning.

The targeted Bloom levels differ across domains because each competency requires a distinct range of cognitive processes. Design and implementation tasks reach the *Analyze – Evaluate – Create* levels, reflecting the synthesis and trade-off reasoning inherent in engineering design. Teamwork and communication competencies span *Apply – Evaluate*, as they emphasize observable collaborative behaviors and justified decision-making rather than artifact synthesis. Testing, debugging, and verification activities map to *Apply – Analyze – Evaluate*, requiring procedural proficiency, diagnostic reasoning, and evidence-based evaluation of quality and adequacy. Aligning each domain to its appropriate Bloom band ensures consistent scoring across evaluators and strengthens the validity of the assessment process.

Regarding calibration and reliability, we align evaluators by jointly reviewing sample work, double-scoring a subset of artifacts, and reconciling differences to refine rubric use. Peer-evaluation items are adapted from established tools and piloted for clarity. While not a full psychometric validation, these steps provide practical scoring consistency appropriate for a capstone and will be strengthened as longitudinal data accumulate.

5.5 Longitudinal and Comparative Evaluation

To measure the long-term impact of the course, it incorporates a longitudinal study that tracks student development from pre-course baseline through post-course outcomes and into early career stages. Pre- and post-course surveys capture changes in confidence, technical readiness, teamwork ability, and adaptability. Alumni tracking gathers data on job placement, employer feedback, and continued use of agile practices in professional settings.

A comparative cohort analysis will contrast our course graduates with students from traditional software engineering courses and with historical cohorts prior to the integration of agile methodologies. This design enables the program to quantify in the future the added value of the hybrid pedagogical model and to refine the curriculum based on empirical evidence. The longitudinal approach supports continuous improvement and provides insights into how experiential agile learning influences early career success. Although the full framework is in place, this is work in progress since to collect the required data will take several years.

6 Discussion

The implementation of this hybrid, agile-centered pedagogy demonstrates the value of integrating experiential, constructivist, and situated learning principles into software engineering education. Students consistently showed growth across technical, interpersonal, and professional domains, reflecting the strengths of a curriculum that mirrors authentic industry practice. Agile activities such as sprint reviews, retrospectives, and stakeholder meetings created opportunities for students to articulate reasoning, negotiate requirements, and refine their work through continuous feedback. These cycles strengthened technical proficiency while cultivating adaptability, communication, and collaborative problem-solving—competencies widely recognized as essential for early-career success.

The multi-dimensional evaluation framework further highlighted how agile methodologies support deeper learning. Students who engaged actively in iterative development and reflective practice demonstrated notable improvements in analytical thinking and design justification, suggesting strong alignment between agile's incremental refinement and higher-order cognitive processes. Peer assessments illuminated teamwork dynamics, showing how structured collaboration and shared ownership foster accountability and self-organization. These findings reinforce agile not merely as a project management approach but as a pedagogical mechanism that embeds assessment, reflection, and skill development into the learning process.

Despite these positive outcomes, several challenges emerged. Students unfamiliar with agile practices initially struggled with the ambiguity and autonomy inherent in iterative development, and some teams required additional scaffolding to manage conflict, distribute responsibilities, and

maintain communication. These challenges underscore the need for early, explicit instruction in agile principles and ongoing mentorship throughout the project lifecycle. Nevertheless, the overall trajectory of student growth suggests that such tensions contribute meaningfully to the development of professional maturity.

The structure and outcomes of the course also offer transferable insights for institutions seeking to modernize capstone experiences. The integration of agile methodologies, ABET-aligned Bloom's rubrics, and multi-layered assessment can be adapted across engineering disciplines. Emphasizing authentic, industry-aligned practice ensures that students develop not only technical knowledge but also the professional competencies required to thrive in complex, collaborative environments. As engineering education evolves, this model provides a viable pathway for narrowing the gap between academic preparation and industry expectations.

7 Conclusion and Future Work

Our course demonstrates the potential of an agile-driven, theory-informed capstone model to enhance professional readiness in software engineering and related fields. By integrating experiential learning, constructivist principles, situated practice, and cognitive apprenticeship, the course creates an authentic environment for developing both technical expertise and essential professional skills. The use of ABET-aligned Bloom's rubrics and a multi-dimensional evaluation framework supports rigorous, holistic assessment of learning outcomes, capturing the competencies required in modern engineering practice. Early findings from assessments and stakeholder feedback suggest that students leave better prepared to navigate uncertainty, collaborate effectively, and deliver high-quality software solutions.

Looking ahead, several avenues for refinement and expansion are planned. Strengthening industry partnerships will provide richer project opportunities and more frequent exposure to professional expectations. Incorporating emerging technologies (e.g., AI-assisted development, automated testing, modern DevOps practices) will help maintain alignment with evolving industry trends. The ongoing longitudinal study will continue tracking graduates into their early careers, offering insights into how agile-based experiential learning shapes long-term professional development. This will guide iterative improvements and contribute to broader conversations in engineering education about preparing students for a rapidly evolving technological landscape.

Through continuous refinement and evidence-based practice, the course aims to remain a model for innovative, industry-aligned capstone education. A particularly original addition is the gamified team-role analysis module, inspired by cooperative gaming archetypes (Tank, DPS, Healer, Support, Utility) [36], which enhances students' situational awareness and role clarity while enabling research on the interplay among perceived roles, peer evaluations, and personality traits [37].

Although this work does not yet include the full longitudinal dataset, the present analysis offers meaningful insights into how multi-theory pedagogical integration shapes student engagement and professional skill development. As is common in design-based educational research, the emphasis remains on the construction, implementation, and theoretical grounding of the instructional model, supported by preliminary indicators.

Declaration of Generative AI and AI-Assisted Technologies in the Writing Process:

During the preparation of this manuscript, the author used Microsoft Copilot (January 2026 version) to polish and improve the readability of the text in a paragraph by paragraph manner. This assistance was limited to refining grammar, spelling, and sentence structure of human-generated content. After using this tool, the author carefully reviewed, edited, and approved all suggestions, and takes full responsibility for the final content of the work.

IRB Statement

This study is under review by the Institutional Review Board (IRB) aiming to be exempt under federal regulations for educational research involving normal instructional practices.

References

[1] V. Garousi, G. Giray, E. Tüzün, C. Catal, and M. Felderer, "Aligning software engineering education with industrial needs: A meta-analysis," *Journal of Systems and Software*, vol. 156, pp. 65–83, 2019.

[2] N. Lankasena and H. L. N. Himanshi, "Bridging the Gap Between ICT University Learning Outcomes and Industry Expectations: An Empirical Examination of Curriculum Alignment," *Sri Lankan Journal of Applied Research in Science*, vol. 5, no. 1, pp. 1–15, 2023.

[3] F. S. Mohammed and F. Ozdamli, "A systematic literature review of soft skills in information technology education," *Behavioral Sciences*, vol. 14, no. 10, p. 894, 2024.

[4] E. O. Navarro and A. van der Hoek, "On the role of learning theories in furthering software engineering education," in *Software Engineering: Effective Teaching and Learning Approaches and Practices* (H. J. C. Ellis, S. A. Demurjian, and J. F. Naveda, eds.), pp. 38–59, IGI Global, 2008.

[5] L. Gavrilas and K. T. Kotsis, "Integrating learning theories and innovative pedagogies in stem education: A comprehensive review," *Eurasian Journal of Science and Environmental Education*, vol. 5, no. 1, pp. 11–17, 2025.

[6] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*. Boston, MA, USA: Addison-Wesley, 2nd ed., 2004.

[7] H. Owen and N. Dunham, "Reflections on the use of iterative, agile and collaborative approaches for blended flipped learning development," *Education Sciences*, vol. 5, no. 2, pp. 85–103, 2015.

[8] V. Mahnič, "A Capstone Course on Agile Software Development Using Scrum," *IEEE Transactions on Education*, vol. 55, pp. 99–106, Feb. 2012.

[9] P. Salza, P. Musmarra, and F. Ferrucci, "Agile methodologies in education: A review," in *Agile and Lean Concepts for Teaching and Learning*, pp. 25–45, Singapore: Springer, 1st ed., 2018.

[10] T. C. Krehbiel, P. A. Salzarulo, M. L. Cosmah, J. Forren, G. Gannod, D. Havelka, A. R. Hulshult, and J. Merhout, "Agile manifesto for teaching and learning," *Journal of Effective Teaching*, vol. 17, no. 1, pp. 1–16, 2017.

[11] R. Dewi and M. Muniandy, "A systematic review of the use of agile methodologies in education to foster competencies," *Sustainability*, vol. 11, no. 10, p. 2915, 2019.

[12] M. Paasivaara and C. Lassenius, "Communities of practice in a large distributed agile software development organization: Case Ericsson," *Information and Software Technology*, vol. 56, pp. 1556–1577, Dec. 2013.

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

[14] E. G. Vargas, A. Chiappe, and J. Durand, "Reshaping education in the era of artificial intelligence: Insights from situated learning related literature," *Journal of Social Studies Education Research*, vol. 15, no. 2, pp. 1–28, 2024.

[15] A. Collins, J. S. Brown, and S. E. Newman, "Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics," Tech. Rep. 403, BBN Laboratories, Cambridge, MA, 1987.

[16] National Association of Colleges and Employers, "Job outlook 2024," Nov. 2023. Employer survey on hiring outlook and career readiness competencies.

[17] N. B. Moe, T. Dingsøyr, and T. Dybå, "A teamwork model for understanding an agile team: A case study of a Scrum project," *Information and Software Technology*, vol. 52, no. 5, pp. 480–491, 2010.

[18] R. Hoda, J. Noble, and S. Marshall, "Self-organizing roles on agile software development teams," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 422–444, 2013.

[19] S. Williams, "Impact of agile methodologies on team communication in software development," *Journal of Software Engineering*, vol. 12, no. 3, pp. 45–56, 2018.

[20] L. Gren, R. Torkar, and R. Feldt, "Group development and team agility: A qualitative and quantitative investigation of agile software development teams," *Journal of Systems and Software*, vol. 124, pp. 104–119, 2017.

[21] D. Ramel and C. Larman, "Agile development: Lessons learned from the trenches," *IEEE Software*, vol. 25, no. 4, pp. 90–97, 2008.

[22] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 2nd ed., 2018.

[23] S. Fernandes, J. Dinis-Carvalho, and A. T. Ferreira-Oliveira, "Improving the performance of student teams in project-based learning with Scrum," *Education Sciences*, vol. 11, no. 8, p. 444, 2021.

[24] M. Paasivaara, J. Vanhanen, V. T. Heikkilä, C. Lassenius, J. Itkonen, and E. I. Laukkanen, "Do High and Low Performing Student Teams Use Scrum Differently in Capstone Projects?," in *Proceedings of the 2017 IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 146–149, IEEE Computer Society, 2017.

[25] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, pp. 1213–1221, June 2012.

[26] V. Stray, N. B. Moe, and T. Dingsøyr, "Daily stand-up meetings: A grounded theory study," *Journal of Systems and Software*, vol. 113, pp. 1–16, 2016.

[27] R. Hoda, J. Noble, and S. Marshall, "Organizing self-organizing teams," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pp. 285–294, 2010.

[28] K. Conboy, "Agility from first principles: Reconstructing the concept of agility in information systems development," *Information Systems Research*, vol. 20, no. 3, pp. 329–354, 2009.

[29] V. Vinekar, C. W. Slinkman, and S. Nerur, "Can agile and traditional systems development approaches coexist? an ambidextrous view," *Information Systems Management*, vol. 23, no. 3, pp. 31–42, 2006.

[30] L. M. Maruping, V. Venkatesh, and R. Agarwal, "A control theory perspective on agile methodology use and changing user requirements," *Information Systems Research*, vol. 20, no. 3, pp. 377–399, 2009.

[31] G. Lee and W. Xia, "Toward agile: An integrated analysis of quantitative and qualitative field data on software development agility," *MIS Quarterly*, vol. 34, no. 1, pp. 87–114, 2010.

[32] D. Janzen and H. Saiedian, "Test-driven development: Concepts, taxonomy, and future direction," *Computer*, vol. 41, no. 9, pp. 43–50, 2008.

[33] L. Madeyski, "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment," *Software Quality Journal*, vol. 18, no. 1, pp. 123–140, 2010.

[34] L. W. Anderson and D. R. Krathwohl, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York, NY, USA: Longman, 1st ed., 2001.

[35] K. R. Gosselin and N. Okamoto, "Improving Instruction and Assessment via Bloom's Taxonomy and Descriptive Rubrics," in *ASEE Annual Conference & Exposition*, 2018.

[36] G. Eleftherakis, M. Yeh, and S. Mahesh, "Agile capstone design with gamified team roles: Enhancing skills and enabling research," in *2026 Spring Conference of the Middle Atlantic Section of the American Society for Engineering Education*, (Newark, DE, USA), March 27–28 2026. (WIP).

[37] G. Eleftherakis, V. Dimitrova, D. Maglova, and V. Morogianni, "Exploring success factors of agile teams: The impact of personality traits, psychological safety, and team reflexivity on performance," in *2024 16th International Conference on Human System Interaction (HSI)*, pp. 1–6, 2024.

Appendix A: Exemplar Rubric Rows Aligned to Bloom's Levels

This appendix provides one exemplar rubric row per domain (Technical, Teamwork, Professional/Business), mapping observable performance descriptors to Bloom's levels *Apply–Analyze–Evaluate–Create*.

Table 1: Technical Domain (Criterion: Design Justification & Implementation Integrity)

Criterion	Apply	Analyze	Evaluate	Create
Design Justification & Implementation Integrity	Implements features; uses frameworks; passes basic tests	Decomposes requirements; identifies trade-offs; traces to reqs/tests	Appraises alternatives w/ evidence; justifies selections	Synthesizes/refactors; adds automation improving maintainability/performance

ABET: Outcome 2; may support Outcome 6.
Artifacts: design docs, repo structure, test coverage, CI logs.

Table 2: Teamwork Domain (Criterion: Collaboration & Role Fulfillment)

Criterion	Apply	Analyze	Evaluate	Create
Collaboration & Role Fulfillment	Participates; completes tasks; reports blockers	Diagnoses workflow issues; distinguishes process vs. interpersonal; rebalances	Weighs options; evidence-based feedback; adapts role behavior	Designs/adopts norms; runs retrospective improvements w/ impact

ABET: Outcomes 5 and 3.
Evidence: peer evaluations, sprint boards, retrospectives, stakeholder feedback.

Table 3: Professional/Business Domain (Stakeholder Communication & Value Rationale)

Criterion	Apply	Analyze	Evaluate	Create
Stakeholder Communication & Value	Clear updates; documented decisions; traceable acceptance criteria	Structures needs; must-have vs. nice-to-have; maps to stories	Prioritizes via value/cost/risk; defends w/ data	Co-creates roadmap; reframes scope under constraints; closes feedback loops

ABET: Outcome 3; may support Outcome 2.
Artifacts: review decks, backlog rationale, story history, demo feedback.

Appendix B: Scoring ABET Alignment

B.1 Scoring Scale

Bloom Level	Proficiency	Score
Apply	Developing	1
Analyze	Proficient	2
Evaluate	Strong	3
Create	Exemplary	4

B.2 Attainment Calculation

$$\text{Attainment (\%)} = \frac{\text{Mean Rubric Score}}{4} \times 100$$

Benchmark: $\geq 70\%$ indicates attainment.

Take for example the **criterion:** Design Justification & Implementation Integrity
ABET Outcomes: 2 (Engineering Design), 6 (Experimentation/Analysis)

Assume the following artifact scores across evaluators and teams:

Team	Score (1–4)
A	3.0
B	3.5
C	3.0
D	2.5
E	3.5

Mean score = 3.1. Attainment = $(3.1/4) \times 100 = 77.5\%$. Since 77.5% exceeds the benchmark of 70%, ABET Outcomes 2 and 6 are considered attained for this offering.

B.3 Reliability

Subset of artifacts are double-scored; agreement (e.g. % agreement) reported.

Appendix C: Mapping Agile Practices to Learning Domains

Table 4: Agile practice → domain → Bloom → instrument

Practice	Domain(s)	Bloom	Evidence Instrument(s)
Sprint planning	Business/Technical	Analyze, Evaluate	Requirements rubric; design/traceability rubric
Backlog refinement	Business/Technical	Analyze, Evaluate	Artifact rubric; stakeholder feedback
Daily stand-ups	Teamwork/Interpersonal	Apply, Analyze	Peer evaluation; instructor observation
Sprint review/Demo	Business/Communication	Evaluate, Create	Presentation rubric; stakeholder scoring
Sprint retrospective	Teamwork/Metacognition	Analyze, Evaluate	Retrospective rubric
Code review	Technical/Detail orient.	Apply, Analyze	Review-depth rubric; PR comments
CI/CD & testing	Technical & Quality	Apply,Analyze,Evaluate	Testing rubric (coverage targets, pass rate, defect trends); automation logs
Requirements elicitation	Business/Communication	Analyze, Evaluate	Interview rubric; requirements checklist
Stakeholder meetings	Business/Professional	Evaluate	Stakeholder professionalism rubric

Appendix D: Rubric for D04 — Software Product

Artifacts follow the repo skeleton (Analysis_Design_Chapter.pdf, Implementation_Chapter.pdf, Testing_Chapter.pdf, Final_Build/).

Scoring: 1=Apply–4=Create. **Weights:** Team C1–C5 = 80%; Individual C6 = 20%. **ABET:** C1–C2–C5→2; C3–C4→6; C5 (docs)→3; C6→5.

Table 5: D04 rubric (condensed).

Criterion (ABET)	Descriptors (1–4)
C1. Architecture & Design (2)	1 Basic structure; 2 Components traced to reqs; 3 Alternatives/trade-offs justified; 4 Cohesive/refactored design aligned to constraints.
C2. Implementation Quality (2)	1 Core features run; 2 Organized code; 3 Reviews/lint/refactors w/ rationale; 4 Modular, automated testing; fixes trace to issues.
C3. Testing & Evidence (6)	1 Some tests; 2 Unit/integration tests; 3 Edge/error & NFR tests + coverage; 4 Risk-based plan w/ adequacy metrics, visible quality gains.
C4. Reliability/Performance (2,6)	1 Manual build works; 2 Repeatable steps; 3 Instrumented runs + bottleneck fixes; 4 Pipelines + demonstrated perf targets.
C5. Requirements Traceability (2,3,6)	1 Partial coverage; 2 Req→Impl→Test mapping; 3 Exceptions/trade-offs explained; 4 Lightweight RTM w/ risk coverage.
C6. Collaboration (Ind.) (5;3)	1 Participates; 2 Owns tasks; 3 Constructive reviews/clear communication; 4 Leads quality/process improvements.

Evidence:

- Chapters (Analysis_Design, Implementation, Testing) + Final_Build
- Repo history, code reviews, static analysis, test reports, performance notes

Attainment: Per Appendix B, Attainment $= \frac{\text{Mean}}{4} \times 100$.

Appendix E: Figures used in the main paper (section 3.4)

Modeling	Faculty and industry mentors demonstrate agile best practices, such as sprint planning, backlog refinement, and team collaboration.
Coaching	Students receive ongoing feedback from mentors during team meetings and sprint reviews.
Scaffolding	Structured agile workflows, project templates, and real-world case studies guide (step-by-step) students through software development processes.
Articulation	Students present their progress in sprint reviews, articulating their design choices and problem-solving approaches.
Reflection	Retrospective meetings encourage students to analyze their workflow efficiency and learning growth.
Exploration	As students gain proficiency, they independently manage projects and refine agile strategies based on personal and team experiences.

Figure 1: Agile learning supports in practice: mapping mentoring strategies to student activities.