2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
Paper ID #53648

Authentic Industry Simulation: A Role-Based Project-Centric Framework for Technology Education

Mr. Ivan Gappy, Excelsior College

Ivan Gappy is a faculty member at Excelsior University's School of Technology, where he teaches in the Bachelor of Science in Information Technology program. He brings industry experience in software engineering and product management from roles at DTE Energy, Ford, and Amazon. At Excelsior, he contributes to curriculum development, serves on the Academic Integrity Committee, and supports the growth of the BS in Computer Science program as a subject matter expert. Gappy also launched a community of practice to strengthen faculty collaboration. His teaching emphasizes real-world application, ethical decision-making, and equipping students for success in a rapidly evolving tech landscape.

Authentic Industry Simulation: A Role-Based Project-Centric Framework for Technology Education

Ivan Gappy
Excelsior University

Abstract

A persistent challenge in undergraduate Computer Science education is helping students translate conceptual knowledge into professional practice. Many students complete programming assignments successfully yet still struggle to collaborate in teams, manage ambiguity, communicate progress, and use industry tools with discipline. This paper presents a role-based, project-centric instructional framework that simulates authentic software development environments within a Computer Science curriculum. The model organizes learning around long-running team projects and iterative Agile cycles, using industry-standard collaboration and delivery tools such as Jira for work planning, GitHub for version control and code review, and Confluence-style documentation practices for shared knowledge. Students are assigned rotating roles such as project manager, developer, quality assurance (QA) analyst, and DevOps and release engineer, and they are expected to perform responsibilities associated with those roles while building a shared product incrementally.

The framework is grounded in social constructivist and collaborative learning theories, which view learning as a socially mediated process where students build understanding through interaction and shared problem-solving (Vygotsky, 1978; Dillenbourg, 1999). Structured interdependence is created through role assignment, shared team artifacts, and coordinated work breakdown, which makes each learner's contribution visible and consequential. The framework also draws on experiential learning principles by positioning students as active participants who develop knowledge through applied experience and reflective practice (Kolb, 1984; Schön, 1983). The simulated workplace becomes both a cognitive and social environment where learners connect theory with professional practice, develop professional identity, and practice norms of modern software engineering.

This paper describes the instructional design of the framework, how it is operationalized through weekly routines and graded artifacts, and how it can be adapted across modalities and student populations. It also outlines an assessment plan that combines learning analytics from tool use with student reflections and team-based performance indicators. By integrating collaborative learning theory with authentic, industry-aligned practice, this role-based, project-centric model offers a scalable approach for preparing learners for the teamwork, accountability, and adaptability required in contemporary software development.

1. Introduction

Computer Science programs are often evaluated by their ability to produce graduates who can build software and contribute to technical teams. Yet many learners experience a gap between academic work and the realities of professional practice. Traditional course structures frequently emphasize individual assignments, short projects, and narrowly scoped requirements. Students

may succeed in producing correct code but still leave with limited experience coordinating work, tracking progress, negotiating scope, documenting decisions, and responding to peer feedback. In industry, these behaviors are not optional. They are central to how teams deliver value and manage risk.

This paper addresses that gap through a teaching framework designed to simulate an authentic software development environment while still supporting learning, feedback, and equitable assessment. The framework is project-centric, meaning that learning revolves around a shared product developed by a team across multiple iterations. It is also role-based, meaning that students assume responsibilities aligned with common software team roles, and their work is coordinated through shared artifacts and routines.

Tooling is not treated as an add-on but as part of the instructional environment. A rotating DevOps and release engineer role makes integration, build health, and release readiness visible and assessable rather than implicit. Jira-style issue tracking, GitHub-based collaboration, and shared documentation serve as the operational backbone of the learning environment. Research on social coding and GitHub-based collaboration suggests that transparency, visibility of work, and structured collaboration can meaningfully influence how teams coordinate and learn (Dabbish et al., 2012; Zagalsky et al., 2015). Research in computing education also indicates that using GitHub in the classroom is associated with improved learning outcomes and student experiences when students engage with more platform features and workflows (Hsing & Gennarelli, 2019). These findings support the idea that tool-mediated collaboration can become a genuine learning environment rather than simply a submission method.

The framework described in this paper is intentionally specific and illustrative. It is not merely project-based learning in a generic sense. It is a structured simulation of a modern development team, with defined roles, repeatable sprint routines, concrete artifacts, and explicit norms that mirror the flow of professional software work. Its uniqueness lies in the combination of: role rotation tied to responsibilities, consistent use of industry tools and workflows as learning mechanisms, and an assessment model that treats collaboration and professional practice as measurable outcomes rather than informal expectations.

2. Theoretical Foundations

2.1 Social Constructivism and Collaborative Learning

Social constructivist theory emphasizes that learning occurs through social interaction, language, and shared activity. Vygotsky's work argues that higher-order cognitive development is mediated through social processes and supported by interaction with more capable peers and structured guidance (Vygotsky, 1978). In a software development context, this aligns with how novice developers learn in teams, where understanding is built through discussion, critique, and joint problem-solving.

Collaborative learning theory extends this idea by focusing on how learners build shared meaning and coordinate cognitive work. Dillenbourg describes collaborative learning as a situation in which two or more people learn or attempt to learn something together, with attention to how interactions are structured and how learning tasks are shared (Dillenbourg,

1999). This paper uses role assignment and shared team artifacts to create intentional interdependence and structured collaboration. The aim is not simply to place students into groups, but to create conditions where collaboration is necessary, observable, and supported.

2.2 Experiential Learning and Reflective Practice

Experiential learning theory frames learning as a cycle of concrete experience, reflective observation, abstract conceptualization, and active experimentation (Kolb, 1984). A project-centric development environment naturally supports this cycle. Students produce a working increment, reflect on what succeeded or failed, connect those outcomes to concepts, and then apply revised strategies in the next iteration.

Reflective practice further emphasizes that professionals learn through reflection during and after action, especially when dealing with uncertain or complex situations (Schön, 1983). Software engineering is filled with ambiguity. Requirements shift, bugs appear, and tradeoffs emerge. The framework uses retrospectives and structured reflection prompts to help students develop the habit of learning from experience rather than treating outcomes as isolated grades.

2.3 Situated Learning and Professional Identity

Situated learning argues that knowledge is embedded in the context and culture where it is used, and learners develop competence through participation in authentic practices (Lave & Wenger, 1991). This is especially relevant to professional readiness. Students benefit when they can see themselves as legitimate participants in a professional community and when their learning activities resemble the tasks and norms of that community.

The role-based simulation described here supports this by making professional practices routine. Students do not only learn about Agile, version control, or quality assurance. They practice these processes repeatedly, with feedback, in a community of practice that resembles a development team.

3. Related Work and Motivation

3.1 Industry Tooling as a Learning Environment

GitHub has become a central platform for collaboration in modern software development, and research has explored how its visibility and social features shape collaborative work. Dabbish and colleagues describe social coding as a form of collaboration enabled by transparency and traceability in open repositories (Dabbish et al., 2012). Zagalsky and colleagues discuss how GitHub can function as a collaborative platform for education, providing workflows that align with real development practices while also introducing learning challenges (Zagalsky et al., 2015). These findings support the framework's emphasis on using GitHub not only as a submission tool but as a core part of the learning process. This framework extends that logic by assigning responsibility for CI and release practices to a rotating DevOps and release engineer role so that delivery discipline becomes part of the learning design.

Computing education research also connects GitHub use to student outcomes. Hsing and Gennarelli report that increased use of GitHub features correlates with more positive learning outcomes and classroom experiences, suggesting that deeper engagement with collaboration

workflows can matter (Hsing & Gennarelli, 2019). Other classroom studies emphasize that Git and version control can serve as a course platform and can improve workflow discipline when integrated intentionally (Haaranen & Lehtinen, 2015).

3.2 Agile and Iterative Development in Education

Agile approaches are commonly introduced in software engineering courses, but they vary in depth. Some courses use Agile vocabulary without reproducing the rhythms and accountability structures that make Agile effective. Work examining Scrum and Kanban in capstone contexts suggests that structured iteration and visual work tracking can support team organization and learning, especially when adapted to the academic environment (Matthies et al., 2018). This supports the framework's focus on sprint routines, work breakdown, and visible progress rather than simply discussing Agile as a concept.

3.3 Project-Based and Authentic Learning

Project-based learning is widely used in computing education, but the degree of authenticity differs. Authentic learning frameworks emphasize tasks that resemble professional practice, sustained inquiry, collaboration, and reflection (Herrington et al., 2010). This framework builds on that tradition by specifying concrete roles, tools, artifacts, and iteration cycles that map directly to professional workflows.

3.4 Why Another Framework

The distinction of this framework is not the use of Agile practices alone, but the deliberate integration of role rotation, tool-mediated workflows, and artifact-based assessment into the learning design. Collaboration is treated as an explicit, measurable outcome rather than an assumed byproduct of group work. Students do not simply participate in a project. They generate traceable evidence of coordination, decision-making, review behavior, and integration discipline through shared tools. This positions the framework as an instructional model where professional practice is both enacted and assessed, rather than simulated informally.

4. The Role-Based, Project-Centric Instructional Framework

4.1 Design Goals

The framework was developed to simulate authentic software teamwork while remaining teachable and assessable within typical course constraints. The first goal is to simulate authentic team practice so students encounter real coordination problems such as dependencies, sequencing, integration risk, and shifting priorities rather than completing isolated tasks in parallel. The second goal is to make collaboration visible by producing artifacts that allow both instructors and students to see how work is distributed, how decisions are made, and where bottlenecks occur over time. The third goal is to support iterative learning by keeping the project persistent across multiple cycles, allowing routines and skills to compound rather than resetting weekly. The fourth goal is to support different student backgrounds, including traditional students, working adults, and career changers, by giving them a structured entry point into professional practices regardless of prior experience. The fifth goal is to enable scalable assessment by grounding evaluation in work products that naturally emerge from authentic workflows rather than requiring instructors to invent additional evidence.

4.2 Core Structure

The course is organized around a single long-running project that persists across the term. Teams build a product through multiple iterations, with each iteration short enough to provide frequent feedback yet long enough to produce meaningful progress. In practice, a common cadence is a one- or two-week sprint depending on course length, credit hours, and expected student workload.

Each sprint produces three categories of outcomes that together reflect both technical progress and professional practice. Product outcomes include a working increment, such as a feature, an API endpoint, a UI component, or an expanded test suite. Process outcomes include updated issue-tracking artifacts, sprint planning records, and evidence of collaboration workflows such as pull requests, reviews, and merges. Reflection outcomes include a team retrospective and an individual role reflection that connects experience to course concepts and expectations for professional growth. This structure is intentionally designed so that progress is not evaluated only as code output, but also as the disciplined coordination that makes shared work possible.

4.3 Team Roles and Rotation

Teams operate with defined roles that rotate across sprints so each student experiences multiple perspectives and responsibilities. Role rotation refers to the structured reassignment of roles across sprint cycles so that each student participates in multiple aspects of the development process rather than remaining in a single function. The project manager facilitates sprint planning, maintains issue-tracking hygiene, monitors scope and dependencies, and is responsible for creating and maintaining core project artifacts such as the project charter or vision, backlog structure, sprint goals, and the team working agreement. The developer implements user stories, participates in code review, and integrates work. The QA analyst defines acceptance criteria, writes or executes tests, verifies completed work, and documents defects. The DevOps and release engineer supports integration and delivery discipline by maintaining repository settings and continuous integration (CI) checks, confirming that pull requests pass required validations, and coordinating lightweight release practices such as version tagging and changelog updates when appropriate.

Role rotation is essential because it prevents students from staying in comfort zones and it creates empathy across responsibilities. Students who have served as QA analysts tend to write more testable acceptance criteria later as developers, while students who have served as project managers tend to communicate status and risk more effectively later in the course. Rotating roles also create a more balanced evidence trail of contribution because different types of work are made visible and valued.

Table 1. Team roles and observable evidence

Role	Primary responsibilities	Typical artifacts produced	Where evidence is found
Project Manager or Scrum Lead	Sprint planning, scope management, coordination, board maintenance, project artifact ownership	Project charter or vision, sprint plan, backlog refinement notes, task breakdown, retrospective action items	Issue tracker, documentation space

Developer	Story implementation, pull requests, review participation, integration	Commits linked to tickets, pull requests, review replies, merged changes	Version control platform, issue tracker
QA Analyst	Acceptance criteria, verification, defect reporting, regression checks	Test notes or test cases, defect tickets, verification comments	Issue tracker, version control platform
DevOps and Release Engineer	Integration discipline, CI checks, branch protections, release coordination	CI configuration updates, release tags, workflow notes, issues for failing checks, PR merge readiness confirmations	Repository, CI logs, issue tracker

4.4 Weekly Sprint Routine

A typical sprint follows a repeatable pattern that emphasizes transparency and traceability. Planning begins with backlog refinement and selection of sprint items. Teams break selected items into tasks, define acceptance criteria, and assign ownership. The project manager ensures tasks are realistic and dependencies are visible before execution begins. During execution, students implement tasks using feature branches and pull requests. Developers are expected to request and respond to code review and to link commits and pull requests back to tickets so the relationship between planning and execution remains explicit. Quality and integration focus on verifying acceptance criteria and surfacing gaps. The QA analyst role confirms that "done" reflects agreed expectations rather than informal claims, and defects or incomplete criteria are documented in the issue tracker. The DevOps and release engineer role monitors required checks and helps teams resolve failed validations so that integration remains a routine habit rather than a last-minute scramble. Completed work is merged into the main branch using a defined workflow, which may include automated checks when appropriate for the course level. The sprint closes with a review and retrospective where teams demonstrate the increment, reflect on what occurred, and select one or two process improvements to try next. These improvements are captured as explicit action items, so they become part of the next sprint plan rather than a one-time conversation.

This routine aligns with research on social coding environments that emphasizes transparency and traceability as coordination mechanisms (Dabbish et al., 2012). It also reflects findings suggesting that deeper engagement with GitHub workflows can improve student experiences and outcomes in software engineering education (Hsing & Gennarelli, 2019).

4.5 Student Activities and Active Participation

To address critiques that many collaborative frameworks lack specificity, this section clarifies what students do in concrete terms. Students write user stories in the issue tracker and include acceptance criteria. For example, a story may specify, "Given a valid login, when the user submits the form, then the system returns a token and redirects to the dashboard." Students estimate work in story points, discuss tradeoffs, and negotiate what can realistically fit into the sprint. Students open pull requests that include a short description, a linked ticket, test evidence, and notes for reviewers. Students conduct code reviews using a rubric that emphasizes readability, correctness, test coverage, and adherence to team conventions. Students document

decisions in a shared log, such as choosing REST over GraphQL because it better matches course scope and reduces integration complexity. Students run or write tests, file defects in the issue tracker, and verify fixes. Students complete retrospectives using prompts such as what slowed the team down, what was unclear, and what single process change the team will try next sprint, supporting reflective practice (Schön, 1983).

These activities are not optional enrichment activities. They are required artifacts that are graded with explicit expectations so that active participation is a core feature of the learning design rather than an informal hope.

5. Industry Alignment and Uniqueness

5.1 Alignment Through Workflows and Norms

Industry alignment is often reduced to adopting the same tools used in professional practice, but tool adoption alone does not guarantee authentic learning. This framework proposes that alignment occurs through workflows and norms that tools operationalize. Students practice incremental delivery, transparent tracking, honest status updates, routine code review, explicit definitions of done grounded in acceptance criteria, and decision documentation that reduces reliance on memory. These behaviors reflect common professional expectations and support the development of collaboration competence in ways that transfer beyond the course context.

Although the framework is described using software development tools, its underlying principles are transferable to non-code contexts. Versioning, review, and traceability can be implemented through collaborative document platforms, design tools, or project management systems. The key mechanism is not the specific tool, but the use of shared artifacts that make contributions, revisions, and decisions visible over time.

5.2 Authenticity Through Transparency

Research on social coding emphasizes transparency as a defining feature of modern collaborative development (Dabbish et al., 2012). In this framework, transparency is not merely a platform feature. It is a pedagogical mechanism. Transparent artifacts allow instructors to see how teams operate and allow students to see the consequences of planning, communication patterns, and workflow discipline. For example, if a team claims progress but the issue tracker remains unchanged, that mismatch becomes an instructional moment about reliability and communication. If a student contributes primarily through code review rather than large commits, the platform still captures that contribution, supporting fairer recognition of the distributed labor that makes software teamwork succeed.

5.3 Partnerships With Practitioners

The framework can be implemented without external industry partners, but it is designed to support practitioner involvement when available. Practitioners can serve as guest reviewers for sprint demos, mentors during backlog refinement, or judges for final presentations. Their feedback is most valuable when it targets process, tradeoffs, and collaboration rather than only technical perfection. Even without formal partnerships, the framework mirrors professional team practices closely enough to support students in building professional identity through situated participation (Lave & Wenger, 1991).

6. Assessment and Evidence Strategy

6.1 What Is Assessed

The framework assesses three domains: technical product quality, process quality, and professional readiness. Technical product quality includes correctness, completeness, and maintainability. Process quality includes how work is planned, tracked, reviewed, tested, and integrated. Professional readiness includes collaboration, communication, accountability, and reflective practice. Treating these as explicit assessment targets reflects the position that professional practice outcomes should be measured as real learning outcomes rather than treated as informal expectations.

6.2 Assessment Artifacts

Artifacts are intentionally selected so they are both authentic and assessable, meaning they emerge from normal work rather than being manufactured for grading. Evidence from the issue tracker includes backlog items, sprint plans, assignments, status updates, defect reports, and retrospective action items. Evidence from version control includes commit history, pull requests, review comments, merged changes, CI check outcomes, release tags when used, and workflow discipline. Evidence from documentation includes decision logs, meeting notes, release notes, and technical summaries. Evidence from reflection includes individual role reflections and team retrospectives. This mix supports a more complete view of contribution than relying on code output alone, which is particularly important in team-based contexts where essential work is often non-code.

Table 2. Assessment artifacts and instructional value

Evidence source	Examples	What it reveals	How it supports assessment
Issue tracker	Stories, acceptance criteria, sprint plan, defect tickets, action items	Planning quality, scope control, accountability, QA discipline	Measures process quality and role performance
Version control	Pull requests, commits, review threads, merges	Collaboration patterns, review quality, integration habits	Measures contribution and workflow competence
Documentation space	Decision logs, meeting notes, release notes	Reasoning, tradeoffs, team memory, clarity	Measures professional communication and alignment
Reflection artifacts	Role reflections, retrospectives	Metacognition, learning transfer, growth over time	Measures readiness and continuous improvement

6.3 Formative and Summative Assessment

Formative assessment occurs weekly through feedback on sprint artifacts and periodic instructor check-ins. Students receive guidance on writing more testable acceptance criteria, creating higher-quality pull request descriptions, and conducting constructive code reviews. Summative assessment occurs at sprint boundaries and at the end of the project. Teams are evaluated on

delivered increments and the quality of their collaboration artifacts. Individuals are evaluated on role performance, contribution evidence, and the quality of reflection.

6.4 Measuring Outcomes Over Time

To respond directly to the reviewer's question about what will be assessed and how evidence will be gathered, the framework uses a mixed evidence strategy that combines observable collaboration traces from tooling, structured reflection, and instructor evaluation. The intent is not to treat platform activity as a proxy for learning on its own, but to triangulate patterns across multiple sources so that collaboration, professional readiness, and product quality can be evaluated in a more defensible way over repeated offerings.

Tool-based indicators are useful because they capture team behavior in the moment and leave an audit trail of how work evolved. For instance, an instructor can observe whether work is flowing through the board, whether acceptance criteria are being used consistently, whether review is happening as a routine, and whether defects are being documented and resolved. At the same time, these indicators must be interpreted carefully. High activity does not necessarily mean high quality, and some valuable contributions are not reflected in commit volume alone. For that reason, the framework pairs analytics with reflective artifacts that make student reasoning visible, and with instructor rubric-based judgments grounded in the same evidence stream.
A detailed set of outcome measures and instruments is provided in Appendix A. Moving this material to an appendix keeps the main narrative readable while still making the assessment plan explicit and reusable across implementations. The evidence strategy aligns with the broader argument that social coding platforms and issue-tracking systems can serve as learning environments with observable collaboration behaviors, while still requiring pedagogically informed interpretation (Dabbish et al., 2012; Zagalsky et al., 2015). It also supports the claim that deeper engagement with workflows and features in GitHub can be educationally meaningful when paired with structured guidance and reflection (Hsing & Gennarelli, 2019).

7. Illustrative Implementation in a Course Context

To demonstrate how the framework operates in practice, this section presents an illustrative implementation within an upper-level software engineering or programming course. The goal is to provide a concrete example of enactment rather than a formal evaluation study.

7.1 Course Context and Structure

The framework is designed for courses that include sustained project work over multiple weeks. A typical implementation includes 20 to 40 students organized into teams of four to five.

In a typical 8-week course, teams complete 3 to 4 sprint cycles, each producing a functional increment and associated process artifacts.

Each team is assigned a long-running project that persists across the term. Example project types include web applications, data-driven dashboards, or multi-component systems requiring coordination across roles. Projects are scoped to require interdependence rather than parallel individual work.

7.2 Team Formation and Role Rotation

Students are assigned to teams early in the term and remain in those teams. Within each team, students rotate through the roles of project manager, developer, quality assurance analyst, and DevOps and release engineer. These roles align with the responsibilities defined earlier in Section 4.3 and are enacted through the same artifact-driven workflows.

Roles rotate on a one- to two-week cadence aligned with sprint cycles. This ensures that each student experiences coordination, implementation, validation, and integration responsibilities.

7.3 Sprint Execution in Practice

Each sprint follows a consistent structure including planning using issue tracking and backlog refinement, implementation through branching and pull requests, review and validation through code review and testing, integration supported by CI checks and merge workflows, and reflection through retrospectives and role-based reflection.

Artifacts generated through this process serve as both learning tools and assessment evidence.

7.4 Example Project Progression

For example, a team building an event management application may begin with authentication and basic functionality. Early sprints emphasize workflow discipline and artifact quality. Later sprints introduce more complex features such as notifications or role-based access.

As roles rotate, students encounter different responsibilities and develop a broader understanding of system development and team coordination. The project repository captures both the final product and the process of development.

7.5 Initial Observations

Initial implementations suggest several patterns. Students require structured onboarding to understand workflows such as pull requests and CI checks. Role rotation increases engagement and exposes students to different aspects of development. Tool-based transparency improves accountability but requires clear expectations. The instructor’s role shifts toward evaluating patterns across artifacts rather than isolated submissions. These observations serve as preliminary qualitative indicators that the framework can support engagement with professional workflows, though formal evaluation is part of future work.

8. Implementation Guidance for Instructors

8.1 Onboarding and Scaffolding

Students can struggle when tools and teamwork are introduced simultaneously. The framework addresses this through staged onboarding. Early in the course, students complete guided setup activities focused on branching, pull requests, issue tracking, and interpreting basic CI checks

used to validate merges. Students receive templates for user stories, acceptance criteria, pull request descriptions, and decision logs. The first sprint is intentionally small and focuses on establishing routines rather than delivering complex features. This scaffolding supports the progression from novice to active participant in a team practice environment consistent with situated learning principles (Lave & Wenger, 1991).

8.2 Managing Team Dynamics

Role rotation helps prevent one student from dominating leadership responsibilities or another from remaining disengaged. Teams also establish working agreements that specify response time expectations, meeting cadence, and review norms. Retrospectives surface issues early and give teams a structured way to address conflict. Students are taught to frame issues as process problems to solve rather than personal failings, consistent with collaborative learning approaches that emphasize productive interaction and shared meaning-making (Dillenbourg, 1999).

8.3 Supporting Online and Asynchronous Modalities

The framework is designed to work in asynchronous environments by treating artifacts as the primary communication channel. Issue boards and pull request threads function as the collaboration substrate, while documentation replaces informal hallway conversation. Short recorded sprint demos can replace live meetings when schedules do not align. Interaction remains required, but it occurs through traceable work products rather than relying on real-time coordination, which supports distributed learners while maintaining authentic collaboration.

8.4 Scalability and Instructor Workload

The framework is designed to remain manageable in larger classes by focusing on structured artifacts and selective evaluation rather than exhaustive review of all activity. Instructors do not need to review every commit or discussion thread. Instead, evaluation can focus on representative samples such as selected pull requests, sprint summaries, and retrospectives.

Rubric-based assessment aligned with role responsibilities allows consistent evaluation across teams. Templates for stories, pull requests, and retrospectives reduce variability and grading ambiguity. In larger sections, instructors can prioritize key checkpoints such as sprint reviews and milestone submissions while using tool-generated evidence to identify teams that require deeper inspection.

9. Discussion

9.1 Benefits

The framework supports repeated practice of professional routines so that skills compound over time rather than remaining isolated. It produces visible evidence of collaboration and contribution, which supports transparency and fairness in team grading. It supports professional identity development by allowing students to enact real roles and responsibilities in a structured environment. It also provides instructors with a concrete mechanism for making teamwork teachable and assessable rather than treating it as an abstract expectation.

9.2 Challenges and Risks

The framework also introduces challenges. Tooling can create overhead if introduced too quickly or without sufficient scaffolding. Team conflict can increase when collaboration becomes consequential and when accountability is made visible. Assessment can become complex if instructors do not use clear rubrics and consistent evidence expectations. Students may resist role rotation if they feel unprepared for leadership or QA responsibilities. These risks reinforce the value of staged onboarding, shared templates, and steady feedback. Reflective practice structures help students interpret difficulties as learning opportunities rather than failure (Schön, 1983).

9.3 What Makes the Framework Distinct

This model is distinct in its operational specificity. It is not only project-based or Agile-inspired in a general sense. It defines roles, routines, artifacts, and measurable behaviors that link collaborative learning theory to tool-mediated teamwork (Vygotsky, 1978; Dillenbourg, 1999) and link experiential learning to iterative sprint cycles and reflection (Kolb, 1984; Schön, 1983). The framework also aligns with research on transparency and collaboration in social coding environments (Dabbish et al., 2012; Zagalsky et al., 2015) and with findings that deeper engagement with GitHub workflows can improve educational outcomes (Hsing & Gennarelli, 2019).

10. Conclusion and Future Work

Computer Science graduates need more than technical correctness. They need to collaborate, communicate, document, and adapt within team-based development environments. This paper presented a role-based, project-centric instructional framework that simulates authentic industry practice through iterative Agile cycles and consistent use of professional collaboration tools. Grounded in social constructivism, collaborative learning, experiential learning, and reflective practice, the framework treats teamwork as a structured learning process with observable artifacts and measurable behaviors.

Future work will focus on formal evaluation of outcomes across multiple offerings and course contexts. Planned assessment includes combining learning analytics from version control and issue-tracking artifacts with student reflections and instructor ratings of role performance. Additional research will examine how role rotation affects professional identity development, how different student populations experience the framework, and what scaffolds best support equitable participation in distributed or asynchronous environments. By making professional practice teachable, assessable, and repeatable, this framework offers a practical path for bridging academic learning and modern software development work.

Appendix A. Outcome Measures and Instruments

Table A1. Outcome measures and instruments for the framework

Outcome domain	What is measured	Instrument or indicator	Evidence source	Collection frequency	Notes on interpretation
Collaboration participation	Whether students engage in	Pull request participation, review	Version control	Weekly	Participation is interpreted alongside role

	shared work rather than isolated completion	participation, issue assignment patterns, response latency in review threads	platform, issue tracker		expectations. Review quality matters more than raw count.
Coordination and workflow discipline	Whether teams plan, track, and update work in ways that support coordination	Ticket status hygiene, backlog refinement completeness, task breakdown quality, use of acceptance criteria, linkage of PRs to tickets	Issue tracker	Weekly	Looks for traceability between plan and execution. Mismatches become instructional signals.
Integration and release discipline	Whether teams keep the build healthy and merge-ready	CI pass rate on PRs, frequency of broken main, time to repair failing checks, presence of release tags when used	Repository, CI logs, issue tracker	Per sprint	Emphasis is on consistency and recovery habits, not tool complexity
Code review quality	Whether review is substantive and improves code	Rubric-scored review comments, evidence of requested changes, follow-through on review feedback	Version control platform	Weekly to per sprint	Low volume with high substance can outperform high volume superficial comments.
Technical product quality	Whether increments function, are maintainable, and meet requirements	Sprint demo success, acceptance criteria pass rate, defect density,	Repository, issue tracker, instructor demo rubric	Per sprint	Expectations should scale with course level. Test quality can matter more

		code-quality rubric, test coverage expectations as appropriate			than coverage percent.
Quality assurance behavior	Whether teams define and verify “done” rather than relying on informal completion	Presence and quality of acceptance criteria, defect reporting quality, verification comments, regression notes	Issue tracker, repository	Per sprint	Also used to evaluate role performance for the QA analyst rotation.
Documentation and decision quality	Whether the team maintains shared understanding and records tradeoffs	Decision log completeness, clarity of meeting notes, release notes accuracy, rationale for major choices	Documentation space	Per sprint	Focus is on usefulness and clarity, not length. Documentation is assessed as a coordination tool.
Reflective practice and improvement	Whether teams learn from experience and apply changes	Retrospective quality rubric, presence of concrete action items, evidence that action items are implemented next sprint	Retrospectives, issue tracker	Per sprint	The key measure is follow-through, not only reflection quality.
Professional identity and confidence	Whether students can describe their role, contributions, and growth using professional language	Role reflection prompts, confidence ratings, narrative evidence of role	Individual reflections	Per sprint and end of term	Helps capture growth for students who contribute in non-code ways or enter with limited experience.

		understanding and transfer			
Equity of contribution and team dynamics	Whether work is distributed fairly and whether norms support participation	Contribution distribution across artifacts, peer feedback, working agreement adherence, conflict resolution evidence	Version control, issue tracker, peer survey	Midterm and end of term	Distribution is interpreted in context of role rotation. Short-term imbalances may be expected within a sprint.
Overall professional readiness	Whether students can operate in an industry-aligned workflow with accountability	End-of-term portfolio package, final demo, rubric-based evaluation of artifacts across the full project	Combined artifact review	End of term	Portfolio is evaluated as a coherent story of practice, not a scrapbook of disconnected items.

References

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. *Proceedings of CSCW 2012*. ACM. https://dl.acm.org/doi/10.1145/2145204.2145396

Dillenbourg, P. (1999). What do you mean by collaborative learning? In P. Dillenbourg (Ed.), *Collaborative learning: Cognitive and computational approaches* (pp. 1–19). Elsevier.

Haaranen, L., & Lehtinen, T. (2015). Teaching Git on the side: Version control system as a course platform. *Proceedings of ITiCSE 2015*. ACM. https://dl.acm.org/doi/10.1145/2729094.2742608

Herrington, J., Reeves, T. C., & Oliver, R. (2010). *A guide to authentic e-learning*. Routledge.
Hsing, A., & Gennarelli, V. (2019). Using GitHub in the classroom predicts student learning outcomes and classroom experiences. *Proceedings of SIGCSE 2019*. ACM. https://dl.acm.org/doi/10.1145/3287324.3287460

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development*. Prentice Hall.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge University Press.

Matthies, C., Kowark, T., & others. (2018). Scrum2Kanban: Integrating Kanban and Scrum in a university software engineering capstone course. *Proceedings associated with SE education workshops (ACM)*. https://dl.acm.org/doi/10.1145/3194779.3194784

Patani, R., Tiwari, S., & Rathore, S. S. (2024). The impact of GitHub on students' learning and engagement in a software engineering course. *Computer Applications in Engineering Education*. https://onlinelibrary.wiley.com/doi/10.1002/cae.22775

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. Basic Books.

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Harvard University Press.

Zagalsky, A., Feliciano, J., Storey, M. A., Zhao, Y., & Wang, W. (2015). The emergence of GitHub as a collaborative platform for education. *Proceedings of CSCW 2015*. ACM. https://dl.acm.org/doi/pdf/10.1145/2675133.2675284