

IMPROVING LEARNING OUTCOMES THROUGH AN AI-ASSISTED GAMIFIED LEARNING ENVIRONMENT

by Misan Paul Etchie

A Thesis

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science in Computer Science

Northern Arizona University

May 2026

Approved:

Igor Steinmacher, Ph.D., Chair

Marco Gerosa, Ph.D., Committee

Priyanka Parekh, Ph.D., Committee

ABSTRACT

IMPROVING LEARNING OUTCOMES THROUGH A GAMIFIED SOFTWARE ENGINEERING LEARNING ENVIRONMENT

MISAN PAUL ETCHIE

Open-source software (OSS) is both a critical part of modern computing and a valuable context for software engineering education. However, newcomers, most especially students, often struggle with the practical and social complexity of contributing through GitHub-based workflows. At the same time, maintainers are repeatedly burdened by questions and support requests from newcomers. To alleviate this, this thesis examines whether an AI-assisted, quest-centered learning system can scaffold OSS-relevant workflows for students while improving clarity of next steps, pacing, and formative feedback. Our research is based on two classroom deployments of OSS-Doorway, a GitHub-integrated learning platform that uses gamified quests and AI-assisted validation to deliver course activities, evaluate submissions, and provide immediate feedback through GitHub issues and pull requests.

The thesis reports two classroom deployments of OSS-Doorway, our AI-assisted, quest-centered learning system, in software engineering and open source software courses at Northern Arizona University. We used post-course surveys and descriptive and thematic analysis to examine students' perceptions of task scaffolding, workflow clarity, automated feedback, and readiness for GitHub-based contribution. Results across both deployments indicate that students valued the hands-on GitHub environment, progressive quest structure, and rapid feedback loop, but also experienced recurring friction with validator reliability, unclear rejection messages, and format sensitivity. The findings suggest that GitHub-based gamified systems can support learning and OSS onboarding when structured progression is paired with transparent and actionable feedback.

Across both deployments, this thesis provides an empirical account of how AI-assisted, gamified systems can support learning and OSS onboarding, where it works best, and where

friction persists. The findings suggest that structured progression and workflow-integrated feedback can improve clarity, pacing, perceived preparedness, and will to contribute to OSS projects.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation, Award #2247929 (A Learning Environment for an Open-Source Contribution Model)

Contents

1	Introduction	1
1.1	Research Questions	3
1.2	Research Objectives	4
1.3	Thesis Contributions	5
2	Related Work	7
2.1	Software Engineering Learning Environments and Workflow-Embedded Education	7
2.2	Scaffolding and Guided Progression in Computing Education	8
2.3	Gamification as Structural Design in Software Engineering Education	10
2.4	Automated Assessment and Formative Feedback Loops	11
2.5	LLMs for Educational Support and Assessment	12
3	System Design, Architecture, and Integration	14
3.1	System Context and Core Idea	14
3.2	Design Goals	15
3.3	Users, Roles, and End-to-End Workflow	16
3.4	Content Model: Quests, Tasks, and Progression	17
3.5	Gamification Model and Learning Mechanics	18
3.6	Task Types and Validation Strategies	20
3.7	High-Level Architecture	24
3.8	Core Components	26

3.9	Integration Points and Deployment Pipeline	29
3.10	Summary	30
4	Overview of the Studies	31
5	CS386 Deployment: First Classroom Study of OSS-Doorway	33
5.1	Research Design	33
5.2	Course Context and Timeline (CS386)	34
5.3	Participants	35
5.4	Data Sources and Collection	36
5.4.1	Survey instrument structure	36
5.5	Data Analysis	37
5.6	Ethical Considerations	38
5.7	Researcher Positionality	38
5.8	Results	39
5.9	Perceived Learning and GitHub Onboarding	39
5.10	Gamified Mechanics	41
5.11	Use of External Large Language Models	43
5.12	Summary of Quantitative Results	43
5.13	Qualitative Feedback from Open-Ended Survey Responses	44
5.14	Discussion	50
5.14.1	GitHub-Native Structure as an Onboarding Mechanism	50
5.14.2	Engagement and Concept Reinforcement	50
5.14.3	Improving the AI Validator	51
5.14.4	Automated Feedback: Timeliness Versus Clarity	52
5.14.5	External LLMs as Compensatory Tools	52
5.15	Implications for System Design	53
5.15.1	Validator Robustness and Messaging	53

5.15.2	Reliability, Failure Handling, and System Hardening	55
5.15.3	What This Refinement Process Contributes to the Thesis	56
5.16	Limitations	57
5.17	Final Considerations	58
6	CS499/599 Deployment: Subsequent Classroom Study of OSS-Doorway	59
6.1	Overview and Study Motivation	59
6.2	Survey Constructs and Guiding Questions	60
6.3	Research Design	61
6.4	Course Context and Timeline (CS499/599)	64
6.5	Participants	65
6.6	Measures and Instruments	65
6.6.1	Survey instrument structure	67
6.7	Data Sources and Collection	68
6.8	Data Preparation	68
6.9	Data Analysis	69
6.10	Ethical Considerations	70
6.11	Results	71
6.11.1	Perceived OSS Readiness	72
6.11.2	GitHub Self-Efficacy	74
6.11.3	Task Scaffolding and Tool Experience	75
6.11.4	Validator Feedback Quality and Actionability	75
6.11.5	Gamification and Overall Recommendation	76
6.12	Open-Ended Responses: Themes from Qualitative Feedback	78
6.13	Overall Recommendation	88
6.14	Discussion	89
6.15	What Worked Well in CS499/599	90
6.16	Where Friction Persisted and Why	93

6.17	How CS386 Findings Informed This Deployment	94
6.18	Implications for GitHub-Centered OSS Education	95
6.19	Limitations	98
6.20	Summary	99
7	Conclusion	101
7.1	Implications for OSS and Software Engineering Education	103
7.2	Implications for Automated Validation and Feedback Design	104
7.2.1	Replication package	105
A	CS386 Qualtrics instrument	111
B	CS499/599 Qualtrics instrument	114

List of Tables

5.1	CS386 post-course questionnaire blocks, analysis linkage, and source provenance.	37
6.1	CS499/599 mapping from thesis research questions to data sources and answer locations.	62
6.2	CS499/599 analytic constructs, Qualtrics columns used in composites, and instrument-source provenance.	68
6.3	Construct-level survey results for CS 499/599.	71
6.4	Overall recommendation (NPS; 0–10).	89

List of Figures

3.1	Student dashboard showing gamified stats; progress, class rank, task streak, and badge, as well as a list of available quests as an entry into the system . .	16
3.2	(1) A GitHub issue for a data-response task. OSS-Doorway asks the student to locate the “Fork” button and submit a multiple-choice answer via an issue comment. (2) Successful validation feedback. The bot confirms completion, awards points, and provides brief next-step guidance. (3) Failed validation feedback. The bot marks the response incorrect and prompts the student to revise and resubmit (e.g., by re-checking the repository UI and the task instructions).	20
3.3	(1) An LLM-validated, open-ended response task in a GitHub issue. OSS-Doorway posts a conceptual modeling prompt (identifying actors), the student submits a short textual answer as an issue comment, and the bot validates the response, marks the task complete, and unlocks the next step. (2) An artifact-based modeling task. OSS-Doorway asks the student to create a UML-style diagram for a dynamic product categorization system; the student submits the diagram, and the bot verifies completion, confirms acceptance, awards points, and directs the student to the next task.	21

3.4	(1) A GitHub issue for a GitHub Actions workflow task. OSS-Doorway provides step-by-step instructions and a starter YAML template for creating a CI workflow that runs the project’s Python tests. (2) The submission and validation loop in the same issue thread: the student posts the workflow link, OSS-Doorway detects that the workflow run is still in progress and asks the student to wait, then confirms completion once the run finishes successfully, awards points, and marks the task as validated/completed.	22
3.5	(1) An iterative code-review task delivered through a GitHub issue. OSS-Doorway posts the programming requirements and step-by-step submission instructions (create a branch, implement the function, push changes, and open a pull request), then has the student submit the PR identifier/link for validation. (2) The corresponding pull request review loop. OSS-Doorway posts structured review feedback (issues to address and next steps), the student revises and re-submits for re-review, and the bot approves and merges the pull request once the implementation satisfies the criteria (showing a full iterate–fix–recheck workflow inside GitHub).	23
3.6	(1) A GitHub pull request for a bot code-review task. OSS-Doorway posts the review prompt and requirements, and the student responds by identifying defects in the code (e.g., a missing null terminator and logic issues in the digit/length handling). (2) OSS-Doorway’s follow-up after applying the student’s review: the bot confirms the fixes, provides brief meta-feedback on the quality of the student’s review (what was identified well and what to improve in clarity/tone/actionability), then marks the task complete and links to the next task.	24
3.7	OSS-Doorway Design Overview.	25
3.8	Professor Interface showing general students’ stats; average score, completion rates, and list of students	27

4.1	Overview of the thesis design. The first deployment in CS386 served as a formative study that identified strengths and friction points in OSS-Doorway. These findings informed targeted refinements to the system, which were then evaluated in the second deployment in CS499/599.	32
5.1	Ranked Survey Items by Mean Score	40
5.2	Perceived Utility of Automated Validator Feedback	41
5.3	Survey Results Grouped By Construct	42
6.1	Composite metrics (Thesis Tables 6.2–6.6 + NPS).	72
6.2	Perceived OSS readiness (1–7) ($N = 23$ Finished).	73
6.3	GitHub self-efficacy (1–5) ($N = 23$ Finished).	74
6.4	Task scaffolding and tool experience (1–7) ($N = 23$ Finished).	76
6.5	Validator-labeled composite (1–5) after removing rows with missing entries or “N/A,” $N = 20$	77
6.6	Gamification and contribution intent (1–7) ($N = 23$ Finished).	78

CHAPTER 1 Introduction

Git and related developer platforms, such as GitHub and GitLab, have become quintessential for collaboration on software projects; whether small toy projects with groups of two, large publicly-funded industrial systems, or open source software (OSS) that the world’s technological systems run on [6, 9]. Consequently, software engineering courses increasingly ask students to practice skills that mirror real collaborative development work. These skills include using version control, navigating repositories, interpreting code reviews, collaborating through issues and pull requests, and following project conventions. Despite this, students often face difficulty turning course concepts into effective action in GitHub- and OSS-like workflows [6, 22, 21].

Common barriers include difficulty knowing where to start, uncertainty about what to do next, and limited access to timely, actionable guidance while working inside open source-like workflows [20, 21]. These challenges pertain to technical skills and also process knowledge and confidence when interacting with platform-centered development tools [6, 9].

Students often struggle not only with individual tools, but with understanding how different parts of the workflow fit together in practice. Prior work shows that newcomers and students face difficulties related to finding a starting point, interpreting repository structure and conventions, and understanding how to participate effectively in project workflows [6, 22, 21]. In classroom settings, these difficulties can be compounded when platform-based activities are introduced without enough guidance on process, expectations, and next steps. As a result, students may know the concepts in isolation, yet still have trouble applying them coherently in GitHub- and OSS-like environments [6, 20, 21].

Gamified learning environments offer one way to organize complex workflow-based learn-

ing. In this context, gamification is not primarily about rewards, but about structuring activity into smaller tasks, making progress visible, and supporting step-by-step advancement through a larger process [3, 25]. When combined with automated feedback, such environments can also support iteration by giving students timely responses across repeated submissions and by helping instructors manage feedback at scale [10, 13].

However, the value of this kind of environment depends on how well it supports students in practice. Students must be able to find tasks, interpret instructions, submit work correctly, and recover productively when a submission is rejected. This makes the feedback loop especially important. Formative feedback is most useful when it is timely, specific, and actionable, and when it supports revision without replacing student reasoning [7, 18]. Many AI-assisted and automated systems struggle to meet this standard for open-ended or artifact-based tasks, where evaluation is less straightforward than for fixed-answer or code-only work [1, 10]. When feedback is vague or inconsistent, students may resort to trial-and-error or turn to external tools to interpret what the system expects [11, 19].

Existing educational tooling, such as GitHub Classroom, helps with repository provisioning and assignment distribution, but provides limited support for guided progression and formative feedback within GitHub-based workflows [14]. GitHub-based and gamified quest-based systems can fill this gap, but their effectiveness depends on iterative refinement driven by evidence from real deployments. Sustained usefulness requires understanding where students succeed, where they get stuck, and which design changes improve the experience in later offerings.

Despite growing interest in workflow-embedded and gamified learning environments, there is limited empirical work that follows an explicit deployment-driven improvement cycle for such systems, refining the system as per the visible constraints of classroom deployment and testing whether those changes improve performance and student experience. This thesis addresses that gap by reporting results from two classroom deployments at Northern Arizona University, using the first deployment to guide system improvements and the second

deployment to evaluate the revised system.

This thesis examines how an AI-assisted gamified GitHub-based learning environment can support student onboarding into collaborative software development and OSS-relevant workflows. To do so, it uses OSS-Doorway, a system that structures coursework as progressive quests delivered through GitHub issues and repository artifacts, with automated validation and feedback to support iterative refinement with reduced reliance on manual grading. The thesis adopts an iterative approach to improvement: deploy the system in a real course setting, collect feedback and usage data, identify weaknesses, implement targeted changes, and redeploy to evaluate whether the experience improves.

1.1 Research Questions

This thesis is guided by three research questions that examine OSS-Doorway as a gamified, GitHub-based learning environment. The questions focus on how students perceived the system’s task structure, how they experienced AI-assisted feedback, and what design lessons emerged across the two classroom deployments.

RQ1: How do students perceive OSS-Doorway’s quest-based scaffolding for learning GitHub-centered software engineering and OSS workflows?

This question is important because OSS-Doorway’s main instructional design is its structure: quests, tasks, GitHub issues, progress cues, and staged workflow practice. Understanding whether students perceive this structure as clear, useful, and supportive helps determine whether the system can reduce onboarding friction and make complex development workflows easier to learn.

RQ2: How do students perceive OSS-Doorway’s AI-assisted feedback and validation?

This question matters because feedback is central to the learning loop in OSS-Doorway. Students submit work, receive automated responses, revise, and try again, so the system’s

educational value depends on whether the feedback is timely, understandable, actionable, and fair enough to support learning rather than frustration.

RQ3: What design lessons and challenges emerge from refining OSS-Doorway across two classroom deployments?

This question connects the two studies presented in this thesis to the broader contribution of the thesis. It examines what worked, what created friction, and what needed refinement across deployments, producing practical lessons about scaffolding, automated feedback, gamification, reliability, and evidence-driven improvement for similar learning environments.

1.2 Research Objectives

This thesis pursues five main objectives:

1. **Establish how a gamified GitHub-based learning environment can support onboarding into OSS workflows.** Examine how students experience structured, quest-based learning in GitHub-centered coursework, particularly in relation to workflow clarity, task scaffolding, and perceived readiness for collaborative software development and OSS practices.
2. **Evaluate the efficacy of AI-assisted validation and feedback in supporting workflow-based learning.** Investigate the extent to which AI-supported validation and feedback help students interpret task expectations, revise submissions, and progress through GitHub-centered OSS learning activities, while identifying the conditions under which such support is effective or problematic.
3. **Identify the key design factors that shape effective automated feedback in guided OSS learning environments.** Investigate how students perceive the usefulness, clarity, and actionability of automated validation and feedback, and identify the main sources of friction that constrain productive iteration and learning.

4. **Examine how iterative deployment informs system improvement.** Analyze how findings from an initial classroom deployment can be used to guide targeted refinements to system design, validation behavior, and deployment practices in a subsequent iteration.
5. **Derive design principles and a structured improvement process for similar systems.** Use evidence from repeated classroom deployments to articulate the design lessons, refinement priorities, and iterative process that can guide the development and evolution of future gamified software engineering learning environments.

To meet these objectives, we relied on two classroom implementations at Northern Arizona University, and through post-course surveys, we analyzed survey responses descriptively and thematically, and used usage data to contextualize student-reported experiences. This design prioritizes ecological validity: students used the system as part of regular coursework, without experimental manipulation.

Although this thesis is concerned with OSS-Doorway, it also provides information about developing gamified learning environments. The findings are intended to help instructors and tool designers identify high-leverage components, use deployment data to drive refinement, and understand

1.3 Thesis Contributions

This thesis makes the following contributions:

- **Empirical evaluation of OSS-Doorway across two classroom deployments.**

This thesis documents student perceptions of learning, engagement, workflow clarity, task scaffolding, and feedback across two course contexts, identifying both the strengths of quest-based GitHub learning and the recurring limitations of automated validation.

- **Evidence-based identification of improvement priorities.** The initial analysis shows the strenght and weaknesses of a gamified learning environment and provides the main improvement targets for the next iteration informed by quantitative and qualitative data.
- **Documentation of a systems approach for improving gamified learning environments.** The thesis provides a structured deploy; measure; improve; redeploy process and evidence that deployment-driven iteration can guide the design and evolution of software engineering educational tools.

CHAPTER 2 Related Work

To inform this thesis, we draw on prior literature on task sequencing, developer-centered learning environments, automated feedback, and iterative improvement. Prior work spans workflow-embedded learning, scaffolding and progression, gamification, automated feedback, LLM-based support, and iterative design-oriented improvement.

2.1 Software Engineering Learning Environments and Workflow-Embedded Education

Recent work in software engineering education has examined the use of developer platforms such as GitHub as part of the learning environment, rather than only as submission infrastructure. Studies of GitHub in courses report that instructors use it to distribute materials, manage assignments, support collaboration, and expose students to workflows that resemble professional software development practice [6, 9]. This line of work positions developer platforms as both instructional tools and learning targets, since students are expected to learn not only course concepts but also how to work through repositories, issues, pull requests, and related artifacts.

Prior research also shows that working in such environments introduces additional challenges for learners, especially in open source settings. Studies on OSS onboarding consistently report barriers related to documentation, project structure, task discovery, communication, and understanding how to begin contributing [22, 20]. This literature further shows that these barriers are not only technical. Social interaction, mentoring, and responsiveness from the community also shape whether newcomers persist in early contribution attempts

[20, 2]. Together, these studies suggest that GitHub- and OSS-centered learning environments require support mechanisms that help learners interpret project context and navigate contribution workflows.

As a result, workflow-embedded education benefits from tooling that reduces ambiguity and supports progress within the same platform where work is performed. Prior onboarding tools, such as FLOSScoach, demonstrate the value of structured, step-by-step guidance for helping newcomers orient themselves and follow project conventions [21]. Likewise, guidance for maintainers emphasizes making processes explicit, curating discoverable tasks, and establishing responsive communication norms as practical mechanisms for lowering entry barriers [17]. Another complementary direction is task discovery support through curated labels (e.g., “good first issue”), though evidence suggests that labels alone are often insufficient when issue descriptions lack onboarding-relevant information [27]. This motivates approaches that proactively identify and recommend suitable starter tasks to newcomers rather than relying on existing labeling practices [8].

This thesis builds on these insights by treating workflow-embedded education as a design problem: if the platform is the learning environment, then the environment must provide structure, pacing, and feedback that help students navigate real repositories without excessive overhead. In particular, the literature points to two needs that are salient in GitHub-centered learning: clear, staged pathways that make “what to do next” explicit, and timely feedback mechanisms that support iteration [13, 11].

2.2 Scaffolding and Guided Progression in Computing Education

Instructional scaffolding refers to structured support that helps learners complete tasks that would otherwise be difficult to accomplish independently, with support reduced as competence develops. In computing education, this support is often implemented through guided

task sequences, templates, examples, and other forms of step-by-step assistance that make complex work more manageable [21, 13]. In OSS-related settings, scaffolding is especially relevant because newcomers must learn not only technical content, but also how to navigate repositories, follow project conventions, and participate in contribution workflows [22, 20, 2]. This thesis builds on that line of work by treating scaffolding as a way to structure participation in GitHub- and OSS-centered tasks while preserving the use of authentic tools and artifacts.

Prior work on OSS onboarding shows that newcomers face both technical and social barriers when entering projects. Systematic and empirical studies identify recurring difficulties such as finding a place to start, understanding repository structure and conventions, interpreting project expectations, and interacting effectively with the community [22, 20, 2]. This body of work characterizes OSS onboarding as more than a problem of technical skill alone, and it has informed the design of support mechanisms intended to lower early contribution barriers. For example, FLOSScoach provides structured guidance to help newcomers orient themselves and follow project conventions [21], while practitioner-oriented guidance emphasizes explicit processes, clearer newcomer pathways, and responsive support practices [17]. In this thesis, we use these studies as the basis for treating scaffolding in OSS-centered learning as support for navigating workflows and project expectations, not only for understanding content.

Related work on scaffolding in computing education also highlights guided progression as a way to reduce learner uncertainty by organizing tasks into smaller, ordered units. In this view, earlier steps provide context and prerequisites for later ones, making progress more legible and reducing the burden of deciding what to do next. We build on this idea in the context of software engineering workflows, where activities such as locating documentation, making changes, verifying outcomes, and participating in pull requests or reviews are naturally sequential. This makes progression-oriented structures, such as quests and tasks, relevant as a scaffolding mechanism for packaging complex workflows into manageable

steps. Prior work on feedback timing and automated feedback further suggests that such progression is more effective when learners receive timely responses that help them revise and continue [13, 11].

2.3 Gamification as Structural Design in Software Engineering Education

Gamification, the use of game design elements in non-game contexts to influence motivation, engagement, and persistence, has been widely studied [3]. In educational settings, the goal is typically not to suggest that points or badges directly produce learning or are the main motivator for it, but that appropriately chosen mechanics can structure participation, enhance engagement, sustain effort on difficult tasks, and make progress visible [4]. Evidence across higher-education contexts is mixed; reported benefits often depend on how well game elements are integrated with course goals and whether they support meaningful practice rather than acting as superficial add-ons [24].

A key distinction in the literature is between structural mechanics and surface rewards. Structural mechanics such as quests, levels, staged milestones, and progression gating can shape pacing and reduce ambiguity by clarifying what to do next and when a learner is ready to move forward. Surface rewards such as points, badges, and leaderboards are more sensitive to student preferences and can be ignored or even viewed negatively when they feel arbitrary or misaligned with learning [4, 25]. Design frameworks and taxonomies of game elements help clarify these roles by enumerating reusable components (e.g., quests, levels, badges, challenges) and mapping them to intended outcomes [4]. In software engineering education specifically, this is relevant because the skills being taught are multi-step and procedural, requiring repeated practice with tooling, collaboration, and process artifacts.

Prior work, therefore, often frames effective gamification in SE education as “structure-first.” Gamification has been explored as a way to increase student engagement in software

engineering activities such as testing, documentation, bug reporting, and process-related practices [16]. More recent studies continue to explore gamified designs that support repeated practice and feedback cycles in technically authentic environments [23]. For OSS-Doorway, the implication is that the quest-and-task layer primarily functions as guided progression: it chunks complex workflows, makes milestones explicit, and supports iteration. Rewards can still play a role, but their usefulness is contingent on perceived fairness and consistency. This aligns with the broader view that the most durable benefits of gamification come from mechanics that improve clarity, pacing, and feedback rather than competition for its own sake [24, 4, 25].

2.4 Automated Assessment and Formative Feedback Loops

Automated assessment has long been part of programming education because it scales evaluation and shortens the feedback cycle, enabling students to practice iteratively rather than waiting for traditional grading that is plagued by delays. Surveys of automated assessment systems emphasize that much of the educational value comes from rapid feedback, turnaround, consistency, and the opportunity to revise and resubmit in response to said feedback [1, 10]. Review of feedback research similarly argues that feedback is most useful when it clarifies the target performance, indicates the current gap, and provides actionable next steps that learners can apply immediately [7, 18]. With this in mind, automated assessment must be designed and developed with the acknowledgment that speed alone is not sufficient; feedback must also reduce uncertainty about what to do next.

In addition to this, the timing of this automated feedback is important to how these loops shape learner behavior. Immediate feedback can change how students distribute effort during problem solving and how quickly they move between attempts, often increasing iteration frequency and reducing time-to-correction [13]. At the same time, fast iteration has trade-

offs. If feedback is vague, repetitive, or easy to exploit, learners may fall into shallow trial-and-error patterns that optimize for acceptance rather than understanding. Prior work on automated grading environments, therefore, discusses the role of policies that structure the loop, such as attempt budgets, submission limits, staged checks, or other constraints intended to balance iteration with reflection [12]. These policies matter even more when the assessed outcome is not just a final answer, but a series of steps crucial for iterative learning and work.

For workflow-embedded coursework like ours, formative assessment extends beyond code and answer correctness to collaborative-process correctness. Students need feedback on whether they opened a pull request properly, addressed review comments in the correct manner, followed OSS community conventions, or configured automation correctly. Automation, therefore, is not confined to the submission of single artifacts but also broadened to include interaction across issues, commits, pull requests, and CI runs. In such settings, automated feedback becomes part of a larger instructional mechanism and is integrated in every facet of said mechanism.

2.5 LLMs for Educational Support and Assessment

LLMs are increasingly used in education and learning to provide explanations, hints, and feedback in natural language. Empirical studies show that LLM-generated guidance can be helpful for learning and iteration, but that reliability varies and learners may over-trust plausible text even when it is incorrect or misaligned with the task [15, 19]. Complementary work on AI coding assistants suggests these tools can also change interaction patterns by enabling rapid suggestion cycling and encouraging users to reformulate requests until they obtain an acceptable output [26]. In learning environments, this can be productive when it supports exploration and revision, but it can also become superficial when learners optimize for acceptance rather than understanding.

For assessment and gating, the central tension is flexibility versus predictability. LLMs can interpret paraphrases, tolerate minor wording variation, and generate richer feedback than exact matching rules, which is valuable for open-ended responses. However, unconstrained use can introduce instability: models may hallucinate criteria, miss required evidence, or implicitly enforce idiosyncratic formatting preferences unless they are carefully grounded. Recent evaluations of LLM-generated programming feedback highlight that a non-trivial portion of hints can contain errors such as incorrect problem identification or flawed explanations [19]. For these reasons, systematic reviews emphasize that dependable automated feedback typically depends on explicit criteria and structured checks [11].

This thesis adopts a hybrid stance: LLM-based feedback can be useful, but it should be constrained by explicit rubrics and paired with deterministic validation whenever feasible [19, 11]. In this framing, the LLM is treated as a component that can increase interpretive flexibility and improve natural-language guidance, not as an unbounded judge. The practical design goal is to preserve the benefits of open-ended evaluation while keeping outcomes legible and consistent for learners.

Finally, in OSS settings, conversational assistance is most meaningful when it is grounded in realistic, project-specific questions rather than synthetic prompts. Dataset work motivated by maintainer burden underscores the value of evaluating agents on authentic newcomer questions drawn from real discussions [5]. This motivation informs how this thesis situates conversational support: as part of a workflow-embedded system where help and feedback are tied to real artifacts, conventions, and constraints.

CHAPTER 3 System Design, Architecture, and Integration

OSS-Doorway is a gamified coursework system that delivers *quests* (multi-step learning modules) through GitHub repositories and issues, and supports an automated evaluator and feedback loop to help students iterate on answers and actions. The system is designed to (i) reduce context switching by keeping students within GitHub, (ii) provide structured onboarding to OSS and GitHub workflows, and (iii) give instructors a configurable pipeline for authoring, testing, and deploying learning content, as well as assessing students by viewing their grades and metrics. This section presents a unified overview of the system and architecture, covering the core idea, design goals, roles, workflow, components, and integration points.

3.1 System Context and Core Idea

In learning through many software engineering and data-oriented courses, students are expected to use GitHub for collaboration, issue tracking, pull requests, code reviews, and contribution workflows. This also carries on into the industry, where GitHub is the go-to, proprietary platform for collaboration. Despite the importance of being well-versed in using GitHub, students often struggle, whether that be with identifying what to do next within a repository, translating conceptual knowledge into concrete GitHub actions, or receiving timely feedback on whether their actions are correct. OSS-Doorway addresses these challenges by embedding course activities into a quest-to-task progression executed through GitHub issues, with these quests and tasks being validated by an integrated feedback mech-

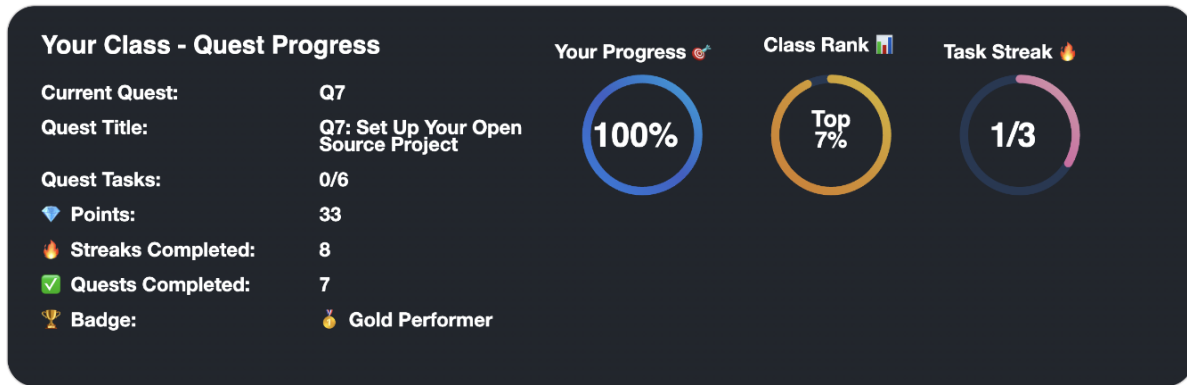
anism.

Each task includes a clear prompt describing the expected work, an explicit acceptance condition, and a validation strategy (for example, repository metadata checks, multiple-choice questions, pull requests, code reviews, or text-based validation). After validation, the system posts immediate success or error feedback and unlocks subsequent tasks or quests according to defined prerequisites, reducing ambiguity about next steps while maintaining an authentic GitHub workflow.

3.2 Design Goals

The design goals of OSS-Doorway were informed by both prior literature and the practical demands of GitHub-centered coursework. Prior work on workflow-embedded learning motivated keeping students in the same environment where collaborative work is performed. Research on scaffolding and newcomer barriers motivated the need for structured progression and clearer onboarding. Work on automated assessment and formative feedback motivated the need for timely, actionable feedback that supports revision. Guided by these considerations, OSS-Doorway was designed around the following goals:

- **G1: GitHub-first learning surface.** Keep students in the same environment used in real OSS practice (issues, pull requests, repository browsing) rather than relying on a separate learning management system.
- **G2: Structured onboarding and pacing.** Provide a guided sequence that helps students build competence incrementally through quests, tasks, and progressive learning.
- **G3: Fast feedback with revision support.** OSS-Doorway offers near-immediate validation results and actionable feedback to enable iterative improvement rather than one-shot submissions.



⚙️ Available Quests

Q1 - Q1: Understanding OSS Projects and GitHub Basics

- ~~T1~~ — Explore the issue tracker [\[COMPLETED\]](#)
- ~~T2~~ — Explore the pull request menu [\[COMPLETED\]](#)
- T3 - Explore the fork button [\[Click here to start\]](#)
- T4 - Explore the readme file
- T5 - Explore the contributors
- T6 - Quiz

Figure 3.1: Student dashboard showing gamified stats; progress, class rank, task streak, and badge, as well as a list of available quests as an entry into the system

3.3 Users, Roles, and End-to-End Workflow

OSS-Doorway supports three primary roles: **students**, **instructors** (professors and teaching assistants), and **the system** (bot and evaluator services). Students interact with quests and tasks within their GitHub repositories. Instructors author and deploy quest content, define validation criteria and feedback messaging, and monitor aggregated progress. The system coordinates task delivery, monitors student submissions and repository events, validates responses, posts feedback, and manages progression and unlocking.

A typical student workflow follows an end-to-end validation loop that constitutes the core interaction primitive of OSS-Doorway:

1. **Repository Creation.** The system creates individual private repos for students when instructors provide their GitHub usernames or students submit their usernames in a

provided invite link.

2. **Task discovery.** Students locate the active task through the README of their personal GitHub repository or issues created by the system as illustrated in Figure 3.1. Each task includes an explicit prompt describing the objective and completion criteria.
3. **Submission through GitHub.** Students complete the required action or produce an answer and submit it via issue comments or the specified GitHub artifact.
4. **Automated evaluator.** The system validates the submission using the task’s validation strategy. This may include deterministic checks (for example, metadata counts or exact answers) and AI-powered text validation for open-ended responses and code reviews.
5. **Feedback and iteration.** The system posts success or error feedback directly into the issue thread. If rejected, students revise and resubmit, forming an iterative loop until acceptance (or until a configured attempt policy is reached).
6. **Progression.** Upon successful validation, the system unlocks the next task or quest and updates the progression state, making the next step explicit.

3.4 Content Model: Quests, Tasks, and Progression

OSS-Doorway structures coursework using a hierarchical configuration model.

A Quest is a themed learning unit (for example, “Understanding GitHub Basics”) containing multiple tasks. Quests can be ordered and may depend on prerequisites, such that completion of one quest is required before unlocking subsequent quests.

A Task is an atomic unit of work within a quest. Tasks may require knowledge responses (for example, multiple-choice questions or short explanations), GitHub navigation (for example, locating issues, pull requests, or README sections), and repository metadata checks (for example, counting open issues or identifying contributors).

Progression is enforced at both the task and quest levels. Students complete tasks in order, and successful validation unlocks subsequent tasks or quests. System mechanics (for example, points, experience, badges, or hints) are enabled to support pacing and iterative feedback loops. These gamified mechanics are designed as scaffolding to help encourage and reward learning.

3.5 Gamification Model and Learning Mechanics

OSS-Doorway incorporates gamification as a learning scaffold rather than a standalone reward layer. The system maps pedagogical progress to game-inspired mechanics so students can understand what to do next, how they are performing, participate in friendly competition, and how to improve after each attempt. In our design, gamification is tightly coupled with task completion, feedback quality, and progression rules.

Core mechanics. The gamified layer is built around six primary constructs:

- **Quests and tasks:** content is organized into quest units (macro-level learning goals) and task steps (micro-level actions), creating a visible progression path.
- **Points and XP:** successful task completion awards points/experience, providing immediate reinforcement and a cumulative signal of effort.
- **Progress indicators:** students can track quest completion and overall course progress through dashboard indicators and issue-level updates.
- **Streaks:** consecutive successful submissions are tracked to encourage consistency and sustained engagement.
- **Badges/achievements:** milestone-based rewards are issued when students complete defined goals.

- **Class rank:** comparative status can be shown to support social motivation, depending on instructor configuration.

Feedback-coupled gamification. Unlike static point systems, OSS-Doorway links rewards to the validation loop. After each submission, the bot posts dynamic feedback (success or error) and updates the progression state. Correct responses trigger positive reinforcement (e.g., XP gain, task unlock, streak updates), while incorrect responses guide revision. This coupling makes gamification informative (what changed and why), not only motivational.

Hints, attempts, and trade-offs. To support students who are blocked, tasks may provide staged hints. Hint usage introduces a controlled trade-off (e.g., point reduction), balancing support with challenge. This mechanism preserves forward progress while discouraging random guessing and helping instructors interpret performance more fairly by considering both completion and assistance usage.

Personalization and learner control. The system supports configurable game elements (for example, ranking visibility) so instructors can align mechanics with course context and students can engage with features that fit their preferences. This avoids a one-size-fits-all gamification strategy and helps reduce unnecessary pressure in mixed-skill classrooms.

Instructional intent. The primary objective of gamification in OSSDoorway is to improve onboarding, clarity, and persistence in GitHub-based workflows. In other words, game mechanics are used to make progress legible, increase feedback responsiveness, and sustain engagement through iterative practice, while preserving authentic OSS tasks and collaboration behaviors.

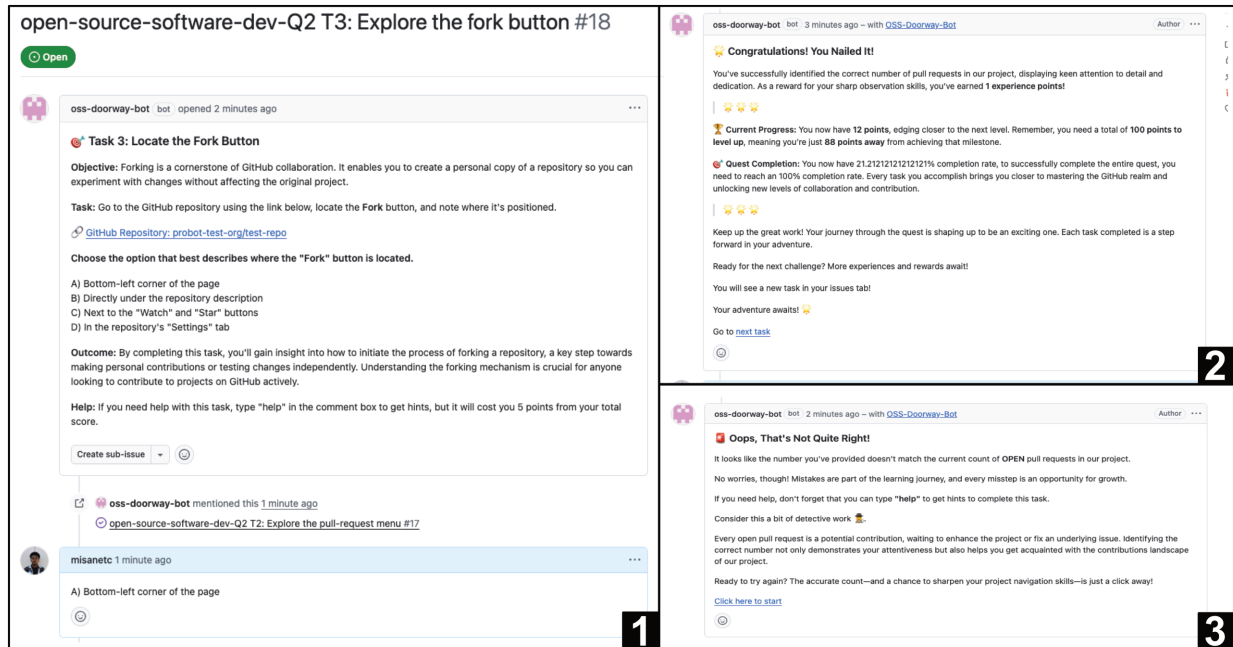


Figure 3.2: (1) A GitHub issue for a data-response task. OSS-Doorway asks the student to locate the “Fork” button and submit a multiple-choice answer via an issue comment. (2) Successful validation feedback. The bot confirms completion, awards points, and provides brief next-step guidance. (3) Failed validation feedback. The bot marks the response incorrect and prompts the student to revise and resubmit (e.g., by re-checking the repository UI and the task instructions).

3.6 Task Types and Validation Strategies

OSS-Doorway supports multiple task types, each mapped to a distinct validation strategy. The task type determines how the system interprets student submissions and which validation logic is invoked. The following enumeration describes each supported type, the expected student deliverable, and the validation mechanism.

Data response tasks. General tasks rely on deterministic validation. The student submits a response (e.g., a number or multiple-choice label) via an issue comment. The system validates by exact matching against expected values or by querying the GitHub API for repository metadata (e.g., count of open issues or pull requests). Quiz tasks use a structured letter format and are validated against a predefined key as seen in Figure 3.2. Hints may be offered at a point penalty.

LLM validation. These tasks accept open-ended textual responses. The student posts their answer in an issue comment. An LLM-based validator evaluates the response against

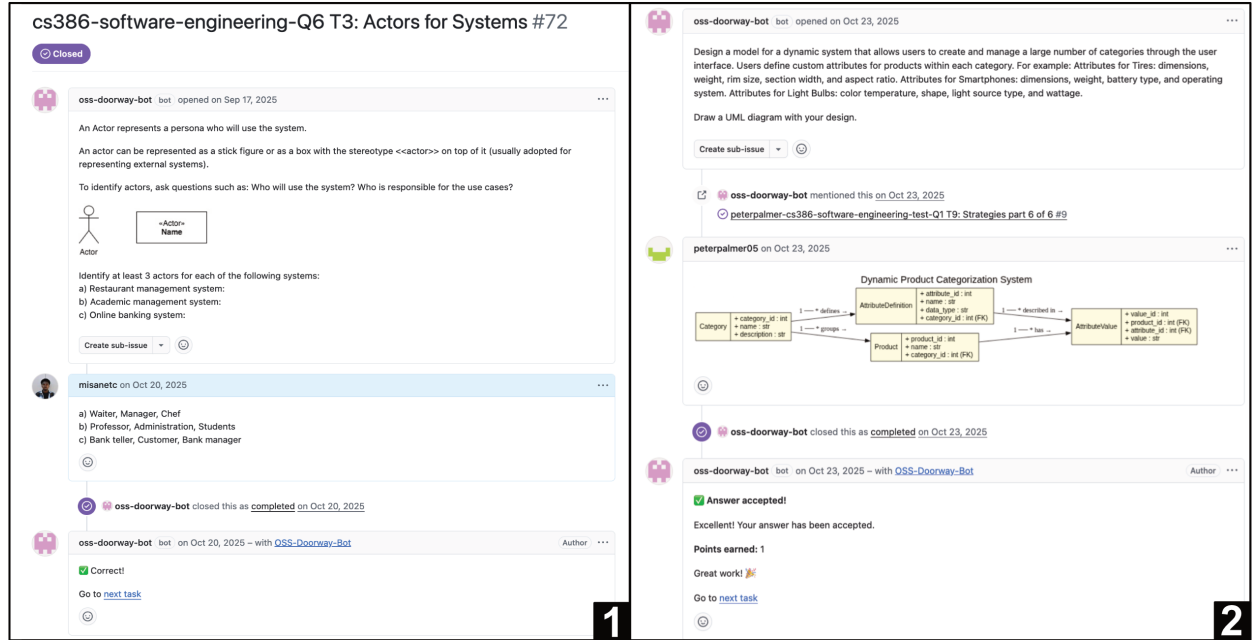


Figure 3.3: (1) An LLM-validated, open-ended response task in a GitHub issue. OSS-Doorway posts a conceptual modeling prompt (identifying actors), the student submits a short textual answer as an issue comment, and the bot validates the response, marks the task complete, and unlocks the next step. (2) An artifact-based modeling task. OSS-Doorway asks the student to create a UML-style diagram for a dynamic product categorization system; the student submits the diagram, and the bot verifies completion, confirms acceptance, awards points, and directs the student to the next task.

task-specific parameters and criteria (e.g., required concepts, constraints, or example answers) as illustrated in Figure 3.3. The validator is configured for educational assessment: it accepts semantically equivalent answers and provides detailed, actionable feedback on rejection without expecting answers in a particular format.

Repository-action validation (forks, files, commits, pull requests, and workflows). These tasks verify that a student completed a concrete GitHub action rather than only answering in text. Students submit a URL (fork, file, commit/SHA, pull request, repository, or workflow) as specified by the prompt. The system uses the GitHub API to check that the artifact exists and matches task constraints: for *forks*, that the URL is a valid fork of the required upstream and that the student has access (with optional storage of the fork URL for later tasks); for *file existence*, that the path exists in the expected repository and, when configured, that it belongs to the student; for *file content*, that the fetched file satisfies rule-based checks (e.g., headings, signatures) or semantic checks via an LLM when configured, with feedback indicating unmet criteria; for *pushes/commits*, that the SHA or commit

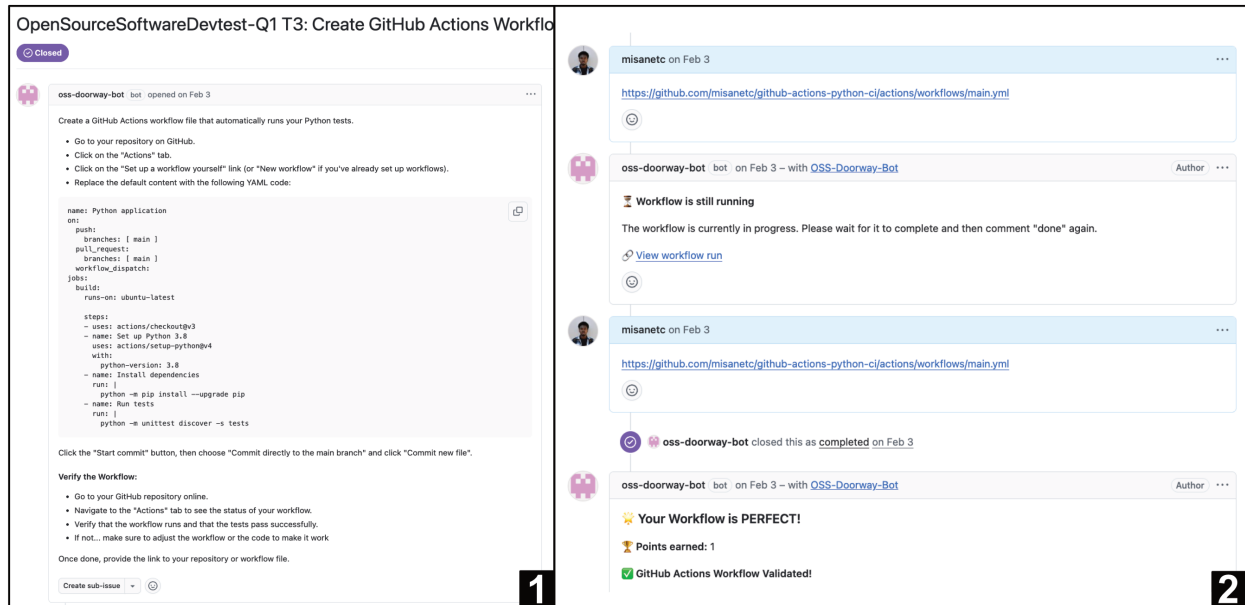


Figure 3.4: (1) A GitHub issue for a GitHub Actions workflow task. OSS-Doorway provides step-by-step instructions and a starter YAML template for creating a CI workflow that runs the project's Python tests. (2) The submission and validation loop in the same issue thread: the student posts the workflow link, OSS-Doorway detects that the workflow run is still in progress and asks the student to wait, then confirms completion once the run finishes successfully, awards points, and marks the task as validated/completed.

URL exists in the target repo, optionally matching a repository pattern, with optional recency or branch rules; for *pull requests*, that the PR targets the specified base repository, that authorship matches, and optionally that the PR is open; and for *GitHub Actions*, that the workflow file meets declared criteria (e.g., runner, Python version, triggers, test command) and, when configured, that a run completes successfully. Together, this family of validators supports authentic OSS workflows while keeping validation objective and tied to observable repository state.

Iterative code review. Tasks of type *iterative-code-review* require the student to implement code in a branch and open a pull request. The bot reviews the PR against task-defined criteria, posts feedback as comments, and iterates until the code meets the criteria or a maximum iteration count is reached. On success, the bot may approve and merge the PR. This type supports structured, multi-round feedback on student implementations.

Bot-led code review. Tasks of type *bot-code-review* cast the student as a reviewer. The bot presents intentionally buggy code; the student posts a review identifying issues.

ImpressionComtest-Q1 T1: Implement a function to add two positive numbers. #1

Closed

oss-doorway-bot bot opened on Jan 30

Implement a function to validate password length.

Requirements:

- Function signature: `def isValidPassword(password)`
- Must return `True` if password is valid, `False` otherwise
- Valid password must be at least 8 characters long
- Must handle `None` input (return `False` if password is `None`)
- Must use `len()` to get password length

How to Submit:

- Create a new branch in your repository:

```
git checkout -b password-validation
```

- Create a file called `password.py` with your implementation
- Add and commit your code:

```
git add password.py
git commit -m "Add password validation function"
```

- Push your branch to GitHub:

```
git push origin password-validation
```

- Create a Pull Request from your branch to `main`:

- Go to your repository on GitHub
- Click "Pull requests" → "New pull request"
- Select your branch (`password-validation`) → base branch (`main`)
- Click "Create pull request"
- Copy the PR link (e.g., <https://github.com/YourOrg/YourReps/pull/2>)

- Post the PR link in a comment on this issue, or just type the PR number (e.g., #2)

Code Language

python

Create sub-issue

oss-doorway-bot added quest-q1 task-t1 on Jan 30

misanetc on Jan 30

#2

Add password validation function #2

Merged oss-doorway-bot[bot] merged 3 commits into main from password-validation on Jan 30

Conversation 3 Commits 3 Checks 0 Files changed 1

misanetc commented on Jan 30

No description provided.

Add password validation function 3b64269

misanetc mentioned this pull request on Jan 30

ImpressionComtest-Q1 T1: Implement a function to add two positive numbers. #1

Closed

oss-doorway-bot bot commented on Jan 30

Code Review Feedback (Iteration 1/3)

Issues to Address:

- Check for `None` before using `len()` to avoid errors
- Return `False` for `None` input to handle invalid cases
- Ensure `len()` is used to get password length

Next Steps:

- Address the feedback above
- Push your changes to this PR
- Comment "ready" or "done" in the task issue when you're ready for re-review

Automated review by OSS-Doorway Bot

Add password validation function a8f11bb

oss-doorway-bot bot approved these changes on Jan 30

View reviewed changes

oss-doorway-bot bot left a comment

Code Review Passed!

APPROVED

Automated approval by OSS-Doorway Bot (Iteration 3/3)

oss-doorway-bot bot merged commit 257938e into main on Jan 30

Revert

Pull request successfully merged and closed

You're all set — the `password-validation` branch can be safely deleted.

Delete branch

Figure 3.5: (1) An iterative code-review task delivered through a GitHub issue. OSS-Doorway posts the programming requirements and step-by-step submission instructions (create a branch, implement the function, push changes, and open a pull request), then has the student submit the PR identifier/link for validation. (2) The corresponding pull request review loop. OSS-Doorway posts structured review feedback (issues to address and next steps), the student revises and re-submits for re-review, and the bot approves and merges the pull request once the implementation satisfies the criteria (showing a full iterate-fix-recheck workflow inside GitHub).

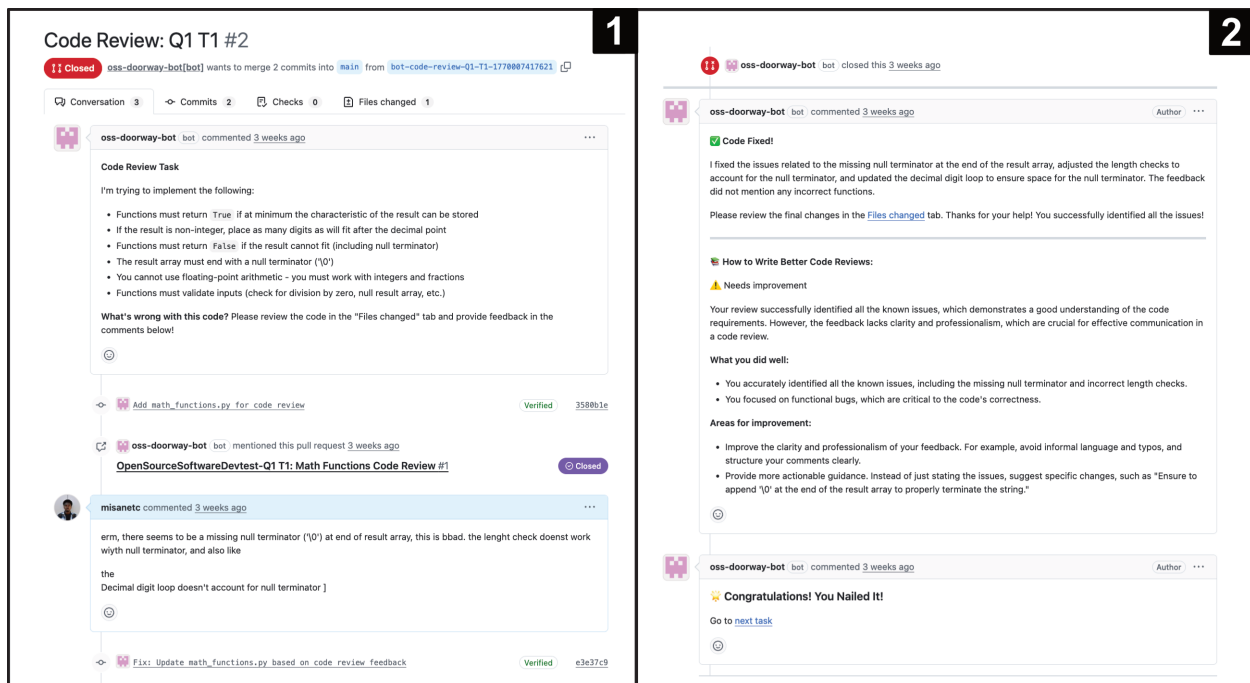


Figure 3.6: (1) A GitHub pull request for a bot code-review task. OSS-Doorway posts the review prompt and requirements, and the student responds by identifying defects in the code (e.g., a missing null terminator and logic issues in the digit/length handling). (2) OSS-Doorway’s follow-up after applying the student’s review: the bot confirms the fixes, provides brief meta-feedback on the quality of the student’s review (what was identified well and what to improve in clarity/tone/actionability), then marks the task complete and links to the next task.

The bot evaluates the review for correctness and completeness (e.g., identification of known defects) and may provide meta-feedback on review quality. Optionally, the bot creates a PR with a corrected implementation for validation.

Together, these task types cover knowledge checks (general, LLM text), GitHub workflow artifacts (fork, file, push, PR, repository, workflow), and code-focused feedback (iterative and bot-led code review). The validator routes each submission to the appropriate validation logic based on the task type and configuration, enabling a single submission channel (issue comments) to support diverse learning objectives.

3.7 High-Level Architecture

OSS-Doorway’s high-level architecture is detailed in Figure 3.7. OSS-Doorway follows a three-tier architecture with explicit separation between instructor authoring and student

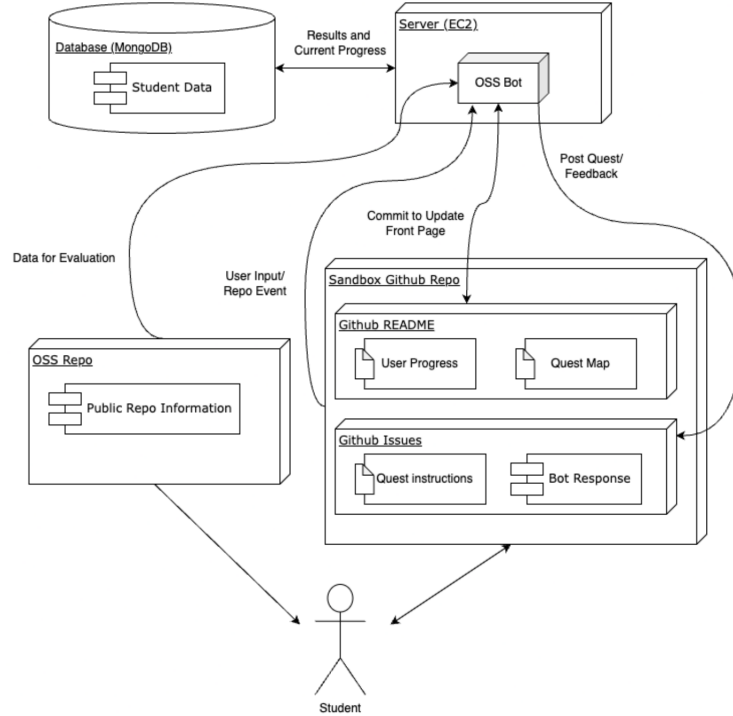


Figure 3.7: OSS-Doorway Design Overview.

runtime:

Presentation layer. An instructor-facing web interface (*Quest Builder*) used for quest authoring, configuration management, and deployment.

Application layer. RESTful backend services are responsible for configuration storage, deployment orchestration, analytics aggregation, and repository management.

Execution layer. A GitHub-integrated bot and validator service that monitors student activity, validates submissions, posts feedback, and manages task and quest progression.

These layers communicate through REST APIs and GitHub webhooks. A key architectural decision is separating *draft configurations* (used during authoring and testing) from *production configurations* (used by the runtime validator for student interactions). This separation enables instructors to iterate on content during an active course without disrupting in-progress student work.

3.8 Core Components

Quest Authoring Interface (Quest Builder)

The Quest Builder is a React-based web application that provides instructors with a visual quest builder. Core capabilities include quest sequence management (create, edit, reorder, delete), task definition across multiple task types, and a draft system for isolating edits from production content. The interface supports importing reusable quests from a shared quest bank, generating test repositories for validation, and previewing or exporting the underlying JSON configuration as illustrated in Figure 3.8.

Instructors configure and manage learning content through an authoring interface (the Quest Builder) and supporting backend services. To minimize risk when modifying live course content, OSS-Doorway separates draft and production content. Draft quests can be edited and tested without affecting students currently working on deployed content. When a quest is ready, it is deployed to the production sequence using a controlled publish operation, and the runtime reads the latest production configuration.

The Quest Builder communicates with backend services via REST endpoints for draft and production configurations. Changes are persisted through auto-save or manual save operations.

Backend Services

The backend is implemented as a Node.js/Express REST API and provides class and group management through CRUD operations for courses, student rosters, and class metadata, persistence and retrieval of draft and production quest configurations, a pipeline for draft quest deployment, aggregation of student progress and task completion rates, and repository management through creation of student and test repositories with initialized configurations.

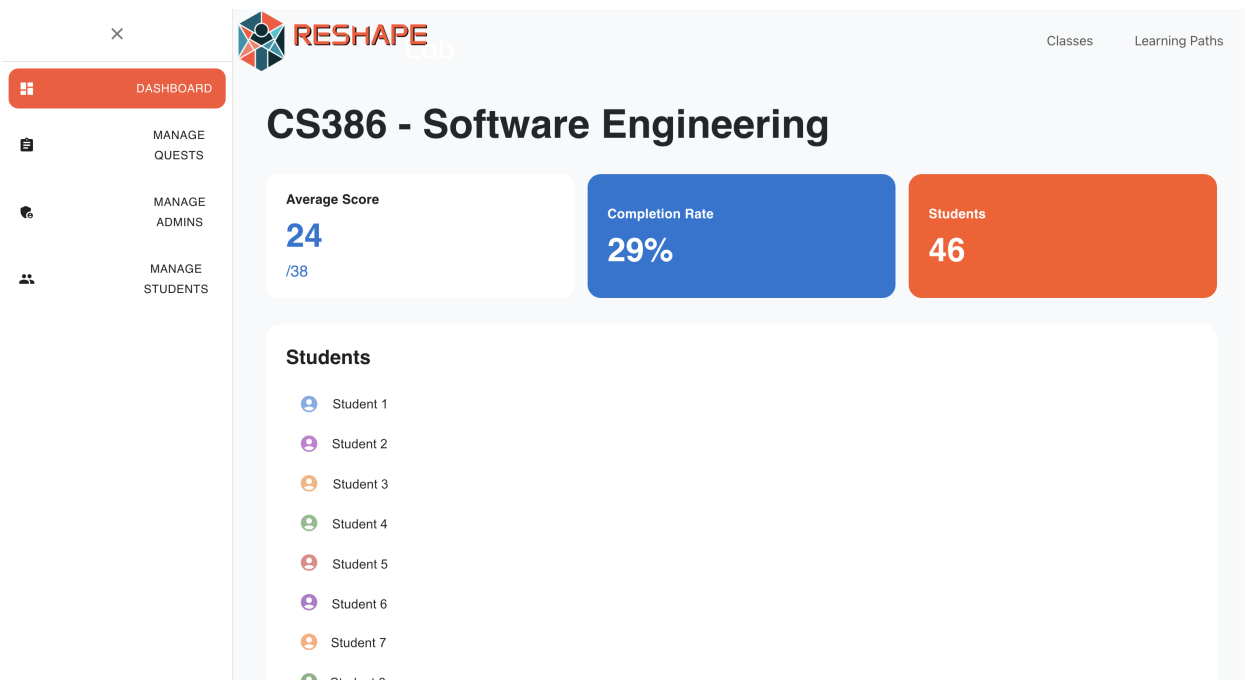


Figure 3.8: Professor Interface showing general students' stats; average score, completion rates, and list of students

GitHub Integration

GitHub serves as the primary interaction surface for students. OSS-Doorway integrates through:

- **Repository provisioning:** automated creation of student repositories with quest configurations preloaded.
- **Issue-based task delivery:** tasks are surfaced as GitHub issues containing prompts, instructions, and progression updates.
- **Comment-based submission:** students submit responses through issue comments, preserving a single conversational context per task.
- **Event monitoring:** GitHub webhooks trigger validation workflows on relevant events (comments and selected repository actions).
- **Feedback posting:** validation outcomes are posted directly to issue threads, keeping feedback co-located with the task prompt.

This design reduces context switching and aligns coursework with authentic OSS contribution workflows, while making task state and feedback visible in the same artifact students are learning to use.

Validator and Feedback Engine

The validator service executes task-specific validation logic and manages feedback and progression:

- **Deterministic evaluation:** exact matching for multiple-choice and quiz tasks, and API-backed checks for repository metadata tasks.
- **Text/Image evaluation:** an integrated evaluator assistant used for open-ended and media-based responses, configured with task-specific parameters such as required concepts and constraints.
- **Feedback generation:** structured success and error messages intended to support revision and clarify next steps.
- **Retry management:** tracking attempt counts and supporting iterative resubmission.

The evaluator assistant operates within the system’s validation loop and is distinct from external LLM tools (for example, ChatGPT, Copilot, or Gemini) that students may use independently. This distinction is important for interpreting student behavior and for attributing feedback effects to the system rather than to external tools.

Data Storage and Analytics

OSS-Doorway persists data across several collections, supporting both instructional oversight:

- **Quest configurations:** production configurations stored in `questconfigs`; draft configurations stored separately in management collections.

- **Student progress:** task acceptance, completion, retry counts, and quest progression.
- **Collected responses:** validated answers saved when artifact collection is enabled (for example, tasks configured to store validated data).
- **Analytics events:** timestamps and timing signals (for example, time-to-completion) and optional hint usage.

These data support instructor dashboards for progress monitoring and identification of bottlenecks, such as tasks with unusually high retry rates that may indicate unclear prompts or insufficiently actionable feedback.

3.9 Integration Points and Deployment Pipeline

GitHub Issues as the Task Surface

Each task is represented as a GitHub issue thread. The system creates and updates issues automatically, uses comments as the submission channel, and can reflect completion via issue state and progression comments. This reinforces issue-based workflows as a learning objective while providing a consistent surface for prompts, submissions, and feedback.

Validation Triggers

Validation is triggered by events routed from GitHub to the validator service:

- **Comment events:** when a student posts a comment, the system parses the submission content and maps it to the active task.
- **Repository events:** for tasks that depend on repository state (for example, file changes or pull request activity), webhooks signal relevant changes for validation.

Routing decisions are task-driven: the system uses task configuration metadata (task type, expected criteria, validation parameters) to select the appropriate validation strategy.

Draft-to-Production Deployment Pipeline

The deployment pipeline supports safe iteration during an active course:

1. **Draft creation and editing.** Instructors create or modify quests in a draft configuration isolated from production.
2. **Testing.** Instructors validate task behavior using test repositories or controlled environments to confirm prompt clarity, validation logic, and unlocking rules.
3. **Deployment (publish).** The `DeployQuest` operation promotes a draft quest into production by appending it to the live quest sequence, assigning sequential quest identifiers, and creating a new versioned configuration document in `questconfigs`.
4. **Runtime update.** The validator loads the latest production configuration and applies unlock logic to make newly deployed content available to eligible students.

This separation prevents draft edits from disrupting active student work and supports controlled rollout of new content.

3.10 Summary

OSS-Doorway operationalizes coursework as a GitHub-embedded quest system with an automated validation loop that provides feedback and controls progression. Students experience structured, repository-centered tasks with iterative feedback in GitHub issues, while instructors author and deploy content through a draft-to-production pipeline that supports safe iteration. Architecturally, the system combines an instructor-facing authoring interface, REST backend services, and a GitHub-integrated execution layer centered on validation, feedback, and progression management. The next section details the quest and feedback model that governs task types, validation behavior, and progression rules.

CHAPTER 4 Overview of the Studies

This chapter provides an overview of the overall study design and explains how the thesis unfolded across two classroom deployments of OSS-Doorway. Rather than treating the two studies as independent evaluations, this thesis follows an iterative design in which findings from the first deployment informed targeted refinements that were then evaluated in the second deployment. Figure 4.1 summarizes this overall process.

As shown in Figure 4.1, the thesis is organized around two deployments connected by an evidence-driven refinement process. The first deployment, conducted in CS386: Software Engineering, served as an initial classroom evaluation of OSS-Doorway in a general software engineering context. Its purpose was to examine how students experienced the quest-based GitHub workflow and automated feedback, and to identify the main strengths and friction points of the system. Evaluation relied primarily on post-course survey data, including both quantitative items and open-ended responses.

Findings from the first deployment showed that students valued the hands-on structure and GitHub-based workflow, but they also revealed recurring challenges, particularly around submission conventions, feedback clarity, and the interpretability of automated validation. These findings directly informed refinements to OSS-Doorway before the second deployment.

The second deployment was conducted in CS499/599: Open Source Software, a mixed undergraduate/graduate course. In this setting, OSS-Doorway was evaluated in a course context more closely aligned with the OSS contribution workflows it was designed to support. The second study focused on students' perceptions of OSS readiness, GitHub self-efficacy, task scaffolding, validator feedback quality, and the value of the quest structure in supporting learning. By deploying the revised system in this second context, the thesis examines not

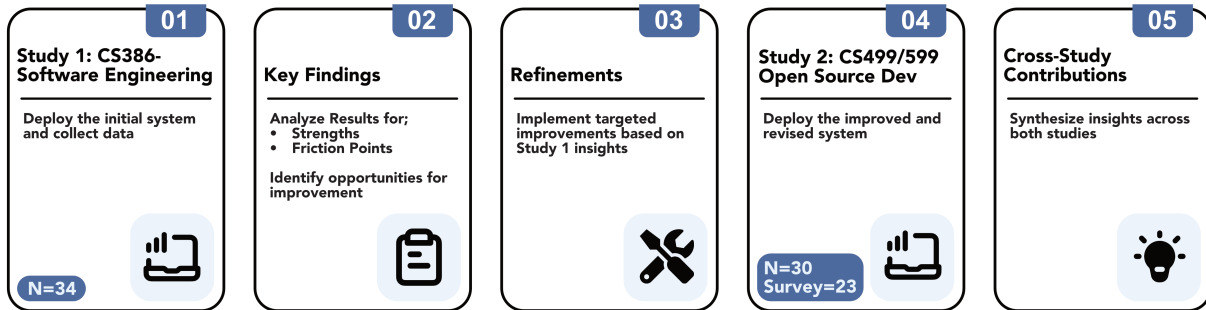


Figure 4.1: Overview of the thesis design. The first deployment in CS386 served as a formative study that identified strengths and friction points in OSS-Doorway. These findings informed targeted refinements to the system, which were then evaluated in the second deployment in CS499/599.

only how OSS-Doorway functioned as a learning environment, but also how evidence-driven refinements shaped the student experience.

Taken together, the two studies form a coherent iterative process: Study 1 served as a formative deployment that surfaced strengths and weaknesses, while Study 2 evaluated the revised system after targeted improvements. This overall design is central to the methodological contribution of the thesis, as it shows how classroom evidence can be used to guide the refinement of gamified GitHub-based learning environments.

CHAPTER 5 CS386 Deployment: First Classroom Study of OSS-Doorway

This chapter reports the pilot classroom deployment of OSS-Doorway in CS386 (Software Engineering). The goal of this first study was formative; to evaluate whether a GitHub-based quest workflow and bot-mediated validation/feedback could support students as they learned software engineering best practices, as well as to identify early friction points that would inform system improvements for a subsequent deployment.

5.1 Research Design

We used a classroom-based field study design and collected primarily self-reported data focused on students' perceived readiness, self-efficacy, and experience using the bot and quest structure, complemented by open-ended feedback describing what was most helpful, what was frustrating, and what should be improved in order to better grasp students' overall experience.

OSS-Doorway was implemented as part of a live course where we focused on capturing realistic student use in a natural classroom setting. We treated this deployment as an initial iteration intended to (i) assess the feasibility and student uptake, (ii) surface usability and workflow issues early, and (iii) generate actionable design requirements for improving tasks, submission conventions, and validator feedback before a second deployment.

5.2 Course Context and Timeline (CS386)

Although OSS-Doorway was originally designed to support OSS onboarding and contribution-style workflows, it was intentionally built to be customizable and reusable for any software engineering coursework more broadly, which was needed in the context of CS386. In practice, that interoperability made it feasible to adapt the same GitHub-based quest-and-task model to a general Software Engineering course like this one, even when many activities were not “OSS contribution tasks,” but rather software development patterns, use cases, UML diagrams, unit testing, and software architecture and design.

In CS386, OSS-Doorway functioned as a delivery and practice layer across the semester. Quests were deployed on a regular cadence twice per week, typically introduced during class, after the first session of instructions and learning have taken place, and students could continue working through the quests outside class time as needed. In the context of CS 386, quests deployed were complementary to learning in class, and use of OSS-Doorway was not mandatory as students were allowed to fulfill coursework through Canvas; the traditional learning management system used in preceding CS 386 sessions. Students interacted with OSS-Doorway entirely through GitHub: the course repository README served as the entry point for discovering work, and GitHub artifacts (issues, labels, milestones, comments, and other repository features) were used as the primary submission surface. As students progressed, the README was automatically updated to reflect their current status and available quests, providing a single, always-current roadmap of what was next.

Coursework delivered via OSS-Doorway was organized as a sequence of quests (Q1-Q22) aligned with weekly course topics and modules, spanning introductory software engineering concepts through later units on processes, modeling, testing, design, and quality. Early quests covered course orientation and foundational concepts (e.g., software complexity, estimation, coordination challenges, ethics, and reflections). Students then moved into workflow- and tooling-centered activities through a dedicated “Software Engineering Tools” quest,

which emphasized practical GitHub literacy: using the issue tracker, writing and discussing issues, labeling issues, creating milestones, assigning and mentioning collaborators, linking issues, searching issues, locating issues in a real project, and practicing markdown, alongside reflection and feedback prompts.

Subsequent quests tracked core lecture content, including requirements concepts, use cases, user stories, software processes (e.g., waterfall and agile concepts), conceptual modeling (including repeated class diagram exercises and use case diagram tasks), testing and types of testing (unit tests, code coverage, TDD, mocks, integration and acceptance tests), and architecture and design. Later in the term, OSS-Doorway continued to deliver more advanced, multi-part material through structured quest sequences on class-level design (class diagrams and sequence diagrams), design decisions and strategies, polymorphism, and design patterns (e.g., strategy, template method, command, composite, singleton, bridge, observer), followed by code quality content such as readability, conventions, code smells, and metrics. Across these quests, students repeatedly practiced course concepts through staged tasks, with reflection and feedback checkpoints embedded at the end of most quest units, while maintaining a consistent GitHub-based interaction model for task discovery, submission, and progress tracking.

After course completion, students were provided with a post-course survey. We collected background information relevant to interpreting students' experiences with the system, including prior familiarity with GitHub, internship/work experience, and related contribution workflows.

5.3 Participants

The course enrolled 34 undergraduate students with a total of $N = 34$ students completing the post-course survey (100% completion rate), which served as the primary data source for the results reported in this study. In addition to Likert-style items, the survey collected

open-ended reflections and background information (e.g., course level and prior GitHub/OSS experience) to contextualize the responses. Survey data were analyzed and reported in aggregate, as well as individually.

5.4 Data Sources and Collection

We collected data from the CS386 deployment using a combination of survey-based measures and students’ written reflections about their experience using OSS-Doorway. The primary data source was a course survey administered after the deployment period, which included both closed-ended items (e.g., Likert-style questions) and open-ended prompts. The closed-ended questions were designed to capture students’ perceived readiness and confidence with GitHub-based and OSS-related workflows, along with their perceptions of task scaffolding, workflow clarity, and the usefulness of the system. The open-ended prompts asked students to describe what they found most helpful, what they found most frustrating, and what changes would improve OSS-Doorway as a learning tool.

5.4.1 Survey instrument structure

The questionnaire was authored for this thesis project and administered through Qualtrics after the semester. It was comprised of sequential blocks: (i) brief introduction to the OSS-Doorway evaluation, (ii) closed-ended Likert-style items probing GitHub onboarding, concept reinforcement versus traditional coursework, perceived engagement, timeliness, clarity and actionability of automated feedback, multimodal submissions, perceived value of superficial gamification, and self-reported use of external LLMs, (iii) open-ended reflections on strengths, frustrations, and concrete improvements (helpful/frustrating/change prompts), (iv) short background prompts (experience, internships, familiarity with workflows). Appendix A reproduces the administered block flow and stems. Table 5.1 lists the same structure by thematic block, answer type, and instrument-source provenance.

Table 5.1: CS386 post-course questionnaire blocks, analysis linkage, and source provenance.

Block	Answer types	Relation to constructs used in analyses	Source
Intro / instructions	explanatory text only	Frames scope of responses	Author-created
Experiential ratings	Likert (agreement / usefulness)	onboarding, scaffolding, feedback, validator, engagement, extern. LLMs	Author-created
Reflections	open text	thematic coding (helpful/frustrating/change prompts)	Author-created
Background	categorical / Likert-lite	contextual covariates (experience, tooling familiarity)	Author-created

5.5 Data Analysis

We analyzed the CS386 deployment using a survey-based approach that combines quantitative summaries with qualitative interpretation of open-ended responses. Quantitative survey items were summarized descriptively and visualized using diverging stacked bar charts. We then ranked items to surface the strongest and weakest aspects of the experience. Where conceptually appropriate, we grouped related items into construct-level summaries (e.g., onboarding/navigation, concept reinforcement, engagement vs. traditional assignments, feedback timeliness, and validator clarity/actionability), while still interpreting results primarily at the item level so that specific friction points were not obscured by a single composite score.

Open-ended survey responses were analyzed using inductive thematic analysis. We performed an initial pass to identify recurring topics and recurring types of friction (e.g., unclear submission conventions, validator format sensitivity, vague instructions, and difficulty interpreting rejection messages). We then grouped responses into themes and refined theme definitions iteratively to ensure themes captured the most frequent and most consequential issues described by students. Finally, we triangulated qualitative themes with quantitative patterns to provide explanation and nuance (for example, when a quantitative item scored lower, we used open-ended responses to clarify whether the cause was unclear expectations, inconsistent task formatting, or feedback that did not provide a clear next step).

Throughout, we avoid overclaiming. Results are interpreted as evidence about student perceptions and experience in a specific course context, and as inputs into an evidence-driven iteration cycle where findings inform improvements to OSS-Doorway for subsequent deployments.

5.6 Ethical Considerations

The CS386 deployment involved minimal risk and was conducted in the context of normal coursework. Participation in any research-oriented data use was treated as voluntary when applicable, and survey or reflective responses were handled as confidential.

To protect students, we report results in aggregate and avoid including identifying information in any dissemination (e.g., names, GitHub handles, or links that could directly reveal student identity). Data access is restricted to the research team, and materials are stored in access-controlled locations.

5.7 Researcher Positionality

I served as the primary developer and operator of OSS-Doorway during the CS386 deployment. This dual role (researcher and system builder) shaped both data collection and interpretation. On one hand, it enabled close visibility into system behavior and rapid iteration on deployment logistics. On the other hand, it introduces potential bias: as the system’s creator, I may be predisposed to interpret feedback more favorably or to focus on issues that are easier to address technically.

To mitigate this, I treated student feedback as primary evidence for identifying pain points, explicitly distinguished between what the data shows versus what it suggests, and used simple descriptive summaries rather than inferential claims. When interpreting qualitative feedback, I prioritized recurrent themes across multiple students rather than isolated comments.

5.8 Results

This section reports results from the post-course survey ($N = 34$) and supporting system usage artifacts. We organize findings around perceived learning and onboarding outcomes, feedback utility, gamified mechanics, and the use of external large language models (LLMs). Survey responses are interpreted alongside behavioral data, including task attempts, retries, and progression patterns, to contextualize students' reported experiences.

5.9 Perceived Learning and GitHub Onboarding

Students reported strong positive outcomes related to learning and GitHub onboarding. Figure 5.1 reveals that gamified tasks reinforced key lecture concepts (25 positive, out of 31) and tasks were more engaging than traditional assignments (22 positive, out of 31), all while helping students familiarize themselves with GitHub (17 positive, out of 31). This result indicates that the system helped students connect lecture concepts to hands-on GitHub work while making the coursework experience more engaging.

System usage artifacts reinforce these perceptions. Students used GitHub issues not only to complete tasks but also to ask clarifying questions and interact with course staff. These interactions occurred within the same issue threads as task submissions, indicating that OSS-Doorway normalized GitHub as both a technical and instructional communication space.

Perceptions of feedback utility were mixed on the other hand. As shown in Figure 5.2, students reported that automated validation feedback moderately influenced how they revised their responses (18 positive, out of 31).

Usage data provides additional insight into these perceptions. Many tasks exhibited multiple submission attempts before acceptance, indicating that students engaged in iterative revision rather than abandoning tasks after initial rejection. In several cases, retries occurred

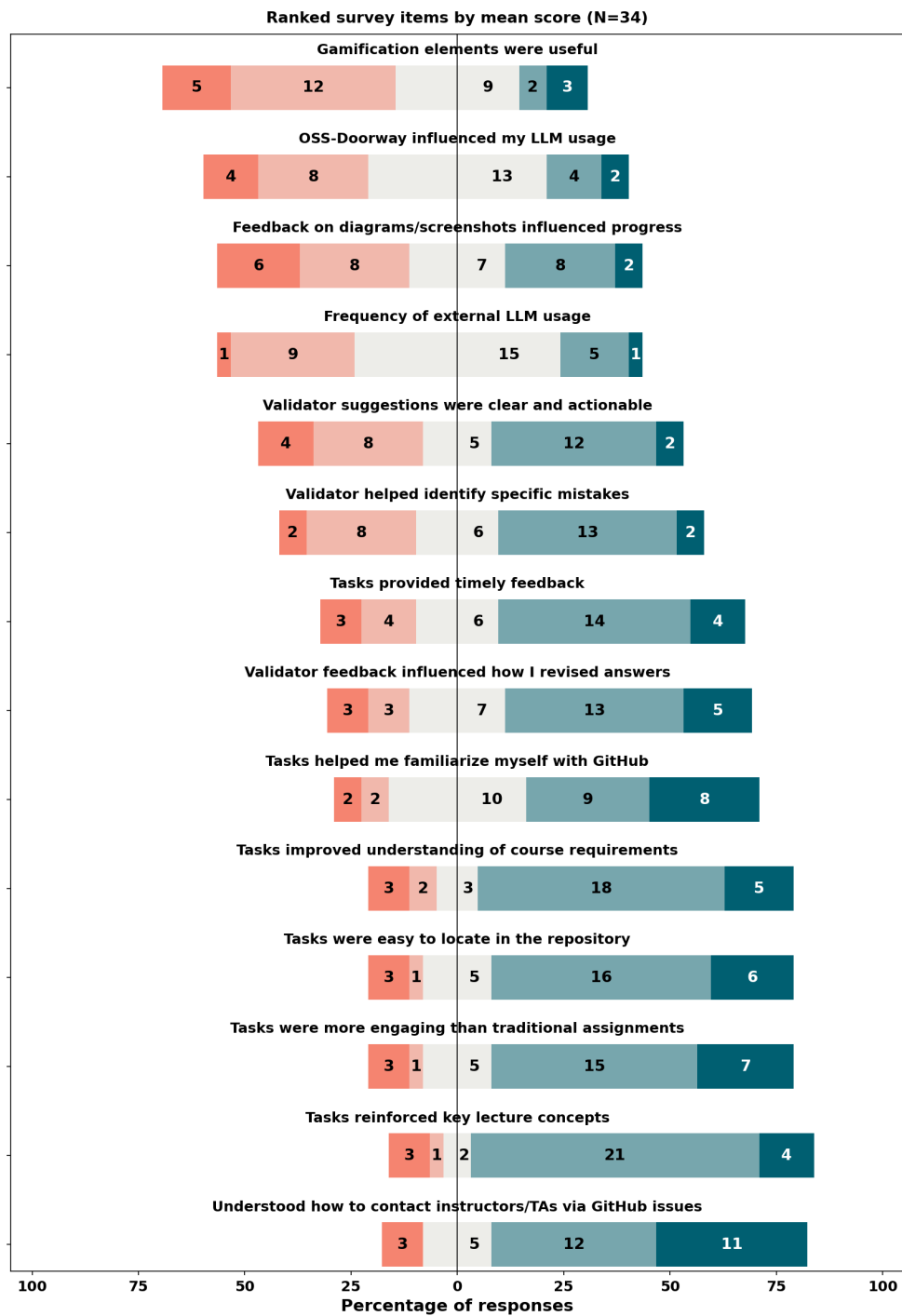


Figure 5.1: Ranked Survey Items by Mean Score

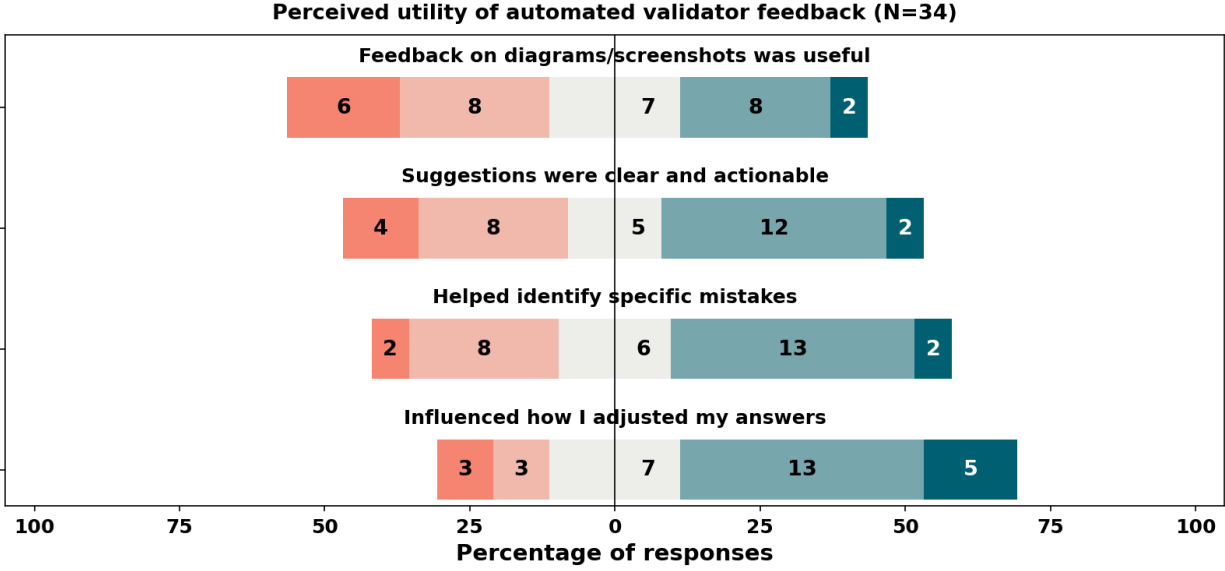


Figure 5.2: Perceived Utility of Automated Validator Feedback

within short time intervals, consistent with the availability of near-immediate feedback.

However, lower ratings for feedback clarity and actionability align with observed retry patterns. Tasks involving open-ended text validation and non-textual artifacts were associated with higher retry counts and longer completion times, suggesting that students often struggled to interpret rejection feedback and determine how to improve their submissions.

5.10 Gamified Mechanics

The perceived usefulness of gamified mechanics was low. As summarized in Figure 5.3, gamification received the lowest positivity score among all constructs (5 out of 31), and the highest negativity score (17 out of 31). Students did not identify points or badges as primary motivators for engagement or learning.

Artifact data corroborates this finding. While progression indicators were visible, students' pacing and completion behavior appeared driven primarily by task structure and course requirements rather than by reward accumulation. Gamified elements thus functioned as background scaffolding rather than as central engagement drivers.

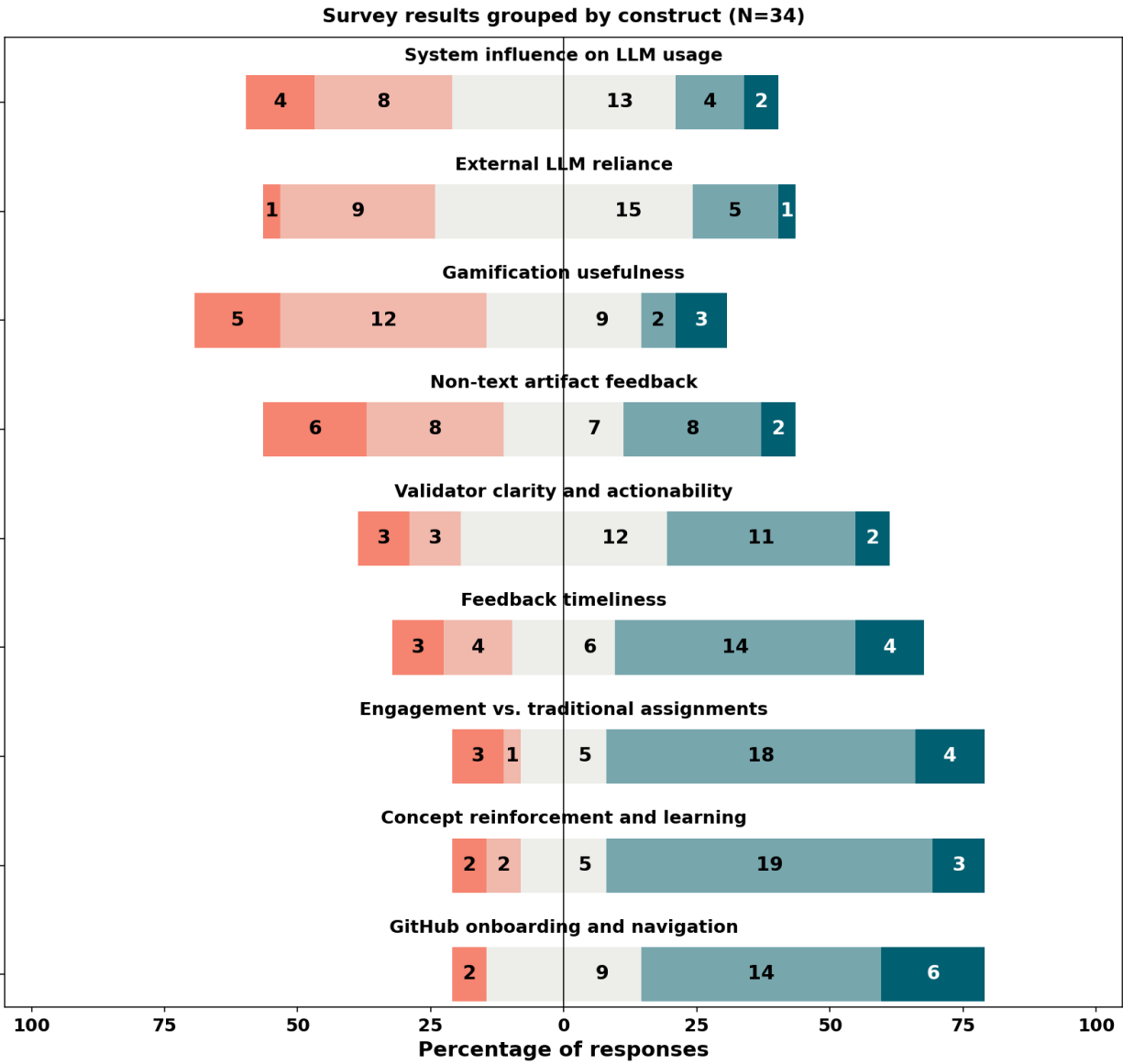


Figure 5.3: Survey Results Grouped By Construct

5.11 Use of External Large Language Models

Students reported moderate use of external LLMs and limited perceived influence of OSS-Doorway on that use. Reported frequency of external LLM usage was moderate (6 positive, 10 negative; Figure 5.1). However, students indicated that OSS-Doorway had relatively little influence on their decision to use LLMs (12 negative, as opposed to 6 positive; Figure 5.1), consistent with the construct-level summary for system influence on LLM usage.

Qualitative responses and usage artifacts suggest that LLM usage was often compensatory rather than habitual. Specifically, students described using external tools to reinterpret task requirements or infer what the validator expected when feedback felt broad or difficult to act on. This mechanism is consistent with the observed pattern that ratings for clarity and actionability (14 positive, 12 negative) and non-text artifact feedback (10 positive, 14 negative) were lower than the ratings for learning, onboarding, and engagement (Figure 5.1).

5.12 Summary of Quantitative Results

Figure 5.3 summarizes construct-level results and highlights a consistent pattern. Students rated GitHub onboarding and navigation positively (20 positive, 2 negative), engagement relative to traditional assignments positively (22 positive, 4 negative), and concept reinforcement and learning positively (22 positive, 4 negative). These construct-level results align with the strongest ranked items, including concept reinforcement (25 positive, 4 negative) and GitHub familiarization (17 positive, 4 negative; Figure 5.1).

In contrast, feedback-related outcomes were weaker and more mixed. Feedback timeliness was moderate (10 positive, 14 negative; Figure 5.3), while clarity and actionability of the validator were mixed to moderate (13 positive out of 31 respondents), and non-text artifact feedback was low (10 positive, 14 negative). Finally, gamification usefulness was low (4 positive, 17 negative), and OSS-Doorway’s influence on LLM usage was also low (6

positive, 12 negative; Figure 5.3). Together, these quantitative results suggest that OSS-Doorway’s main strengths lie in GitHub-based structuring for onboarding and engagement, while the highest-leverage limitation is the interpretability of automated validation feedback, particularly for open-ended and artifact-based tasks.

5.13 Qualitative Feedback from Open-Ended Survey Responses

In addition to Likert-scale ratings, students provided written comments that clarify why specific aspects of OSS-Doorway worked well and where friction remained. Overall, the open-ended responses echoed the quantitative pattern: students valued the GitHub-based structure and the immediacy of feedback, while describing recurring breakdowns when validator messages were not sufficiently specific, especially for open-ended and artifact-based tasks. Several students also described using external LLMs as a compensatory tool when feedback was difficult to interpret.

GitHub-based workflow and reduced ambiguity. Students frequently described the quest-to-task structure as helpful for making expectations concrete and reducing uncertainty about what to do next. Comments highlighted that having prompts, submissions, and instructor interactions co-located in GitHub issues made the workflow feel more coherent than juggling multiple course platforms.

I think it would have been surface level. I struggled a lot keeping up with them but if we hadn’t had them, I think I wouldn’t have learned as much as I did. They were also extremely helpful when we were finishing the deliverables.

I would not have learned anything honestly it’s the only thing that taught me anything for this class because I do not and lots of people do not like slide decks lectures

In particular, students emphasized that the issue threads supported communication with course staff in context, reinforcing GitHub as both a development and instructional space.

I think I honestly wouldn't even have learned anything. The lectures gave me concepts to memorize, and the quests helped me actually use those concepts. Without that, I would have a far lesser understanding of many of the things taught in class.

This honestly happened a lot. The lectures kind of gave a basic understanding, but the quests helped you really dive in and begin to use and understand what was taught. Also, having that immediate feedback made it better than homework as with the homework I was never too sure if I was doing it right.

Immediate feedback as a driver of iteration. Written responses commonly framed the validator as useful when it provided quick confirmation and enabled iterative refinement. Students noted that the ability to submit, receive a response, and revise within the same issue thread reduced waiting time and supported learning-by-doing.

Any time there was a diagram requirement, it was nice to have instant feedback of my diagram, instead of waiting until the assignment was graded

I felt like having the instant feedback was a main factor with the bot, without it I feel like every answer I gave out was right,, even though they could of been wrong.

These comments align with artifact patterns showing multiple short-interval retries on some tasks, suggesting that students engaged in rapid feedback loops when the system's signals were interpretable.

I think my learning would be slower and it would have required me to make mistakes and not know if I was doing the right thing until a grader could see it. By that time I would probably be onto the next thing in the class.

I really enjoyed this way of doing assignments. It was nice to know if I had to fix something right away and I enjoyed that it didn't take too much time out of my day. The task assignments always felt like I was actually learning something as I was able to use what I learned instead of just memorizing key concepts.

Clarity and actionability as the primary pain point. The most consistent critical theme concerned feedback specificity. Students reported that some rejection messages did not clearly explain what was wrong or how to fix it, which sometimes led to repeated trial-and-error submissions.

Several times when using the bot, my answer would be correct but the bot would mark it as wrong, usually because of some specific formatting it wanted. I remember one specific task that said to submit a screenshot, but the bot would not accept it so I think ensuring the submission type is clear for each task would help.

This concern was especially salient for tasks requiring open-ended explanations or involving non-textual artifacts (e.g., diagrams or screenshots), where students expressed uncertainty about the evaluation criteria or the reasons for rejection.

It would be difficult to do, but the AI often fails to recognize when I am describing the idea accurately, but not using the exact language it likes.

I wish it can be more specific where I got my answer wrong, instead of being broad and say my entire answer was incorrect.

More than 70% of the time the feedback would be to just reread the instructions, which I often found more irritating than helpful. I think that maybe trying to actually guide rather than have a comment that proves no use would be more beneficial.

These qualitative reports mirror the lower ratings for validator clarity and artifact feed-

back and provide a mechanism-level explanation for higher retry counts on those task types.

Response Latency and Progression Delays. Although many students described the validator as valuable for providing immediate confirmation and supporting rapid iteration, a smaller set of open-ended responses highlighted a contrasting experience: slow reply times that interrupted task momentum. Students reported that delayed bot responses sometimes prevented the next task from appearing quickly, creating a stop-and-go workflow within GitHub issues. This concern reflects a practical usability constraint of feedback-in-the-loop systems: even when feedback is actionable, latency can reduce its perceived usefulness by breaking the tight revision cycle that students found motivating.

I hated the github bot please get rid of it. It was too slow and I would have to use the bot through issues and the next task would never appear fast enough. Also the bot is extremely picky on what it wants

This feedback complements the broader qualitative pattern. It suggests that student experience depends not only on what feedback says (clarity and specificity) but also on when it arrives. In a quest-to-task progression model, response delays can feel especially disruptive because progression is gated: if validation or task unlocking is slow, students may perceive the system as unresponsive even when the underlying task structure is effective. Together with comments about strict formatting expectations, these responses indicate that improving perceived responsiveness (e.g., faster validation turnaround and clearer progress signals while waiting) is important for sustaining the intended rapid feedback loop and reducing frustration.

Limits of AI Validation for Open-Ended Responses. Several students attributed frustration to the behavior of AI-based text validation on open-ended tasks. Rather than feeling evaluated on conceptual correctness, they described an implicit requirement to match the validator’s preferred phrasing or include specific keywords to secure acceptance. This created a perception that success depended on “writing for the bot” rather than demonstrat-

ing, and that correct explanations could still be rejected when they did not mirror expected language patterns.

I feel like there were key words expected that I didn't get until half way through. My answers would be correct but miss rewriting the question in my answer. If you specify that in the beginning it think the frustration with the "feedback" would go down

These comments also clarify a concrete mechanism behind some negative feedback experiences: students reported discovering "what the validator wants" only after repeated attempts. This suggests that the AI validator struggled to interpret nuance and relied heavily on surface cues (e.g., keywords or expected phrasing) when deciding whether an answer met the acceptance criteria.

Often times I felt like I understood the concept, but I just had to shove in as many relevant words as possible so the AI would recognize it.

In practice, students described adapting by rephrasing responses to restate the question, inserting additional relevant terms, and iterating until the validator recognized the answer. While this behavior still reflects iteration and revision, it represents a less productive form of iteration because revisions are driven by guesswork about language matching instead of targeted correction of conceptual mistakes.

It would be difficult to do, but the AI often fails to recognize when I am describing the idea accurately, but not using the exact language it likes.

Together, these responses reinforce the need for clearer up-front guidance on expected answer structure (e.g., whether restating the prompt is required) and feedback that distinguishes missing required elements from mere wording differences.

Hands-on application made concepts concrete. Students repeatedly emphasized that quests were most valuable when they required doing rather than memorizing. Instead of

passively absorbing lecture material, students described moments where applying concepts in GitHub made the ideas feel “real” and easier to retain. This was especially salient for tasks that involved performing concrete actions (rather than describing them), which helped students understand both the workflow and the purpose of the Git tools.

Every time we had to put a concept into practice, the quest clarified the concepts

Several students explicitly contrasted the quests with lecture formats they found less effective, framing tasks as a hands-on alternative that better supported their learning. These comments reinforce the idea that OSS-Doorway’s value is not only assessment, but guided practice that turns abstract course concepts into operational skills.

Lectures were just slide decks I don’t know anyone who learns well off slide decks, the tasks were better because it was hands on, at least for me

The Version Control Systems module was perfect for the quests. Asking us to actually use GitHub and complete tasks made the concepts much clearer than lectures or written homework. Being able to apply everything helped me understand how version control works in real practice.

The strongest examples of “concepts becoming real” appeared in the version control module, where students noted that executing GitHub actions directly (and seeing their consequences) clarified what lectures and written homework could not.

Taken together, the open-ended responses deepen our understanding of the quantitative results. They suggest that OSS-Doorway’s strongest contribution is providing a GitHub-based, structured pathway that supports communication and iteration, while the system’s primary limitation is not speed but interpretability of feedback. Improving the transparency and specificity of validator messages, particularly for open-ended and artifact-based tasks, appears likely to reduce unproductive retry cycles and decrease the need for compensatory

use of external LLM tools.

5.14 Discussion

Our findings highlight both the promise and the current limitations of GitHub-based, task-based coursework systems with automated feedback. In this section, we interpret the results in relation to the system’s design goals and prior work, and discuss implications for the design of scalable feedback mechanisms in software engineering education.

5.14.1 GitHub-Native Structure as an Onboarding Mechanism

The strongest positive outcomes observed in our study relate to GitHub onboarding and navigation. Students reported increased familiarity with GitHub workflows, greater ease in locating tasks, and clearer understanding of how to contact instructors and teaching assistants through GitHub issues. These results suggest that embedding coursework directly within GitHub, rather than layering GitHub usage on top of an external learning management system, can reduce ambiguity and lower barriers for newcomers.

The quest and task structure played a key role in this effect. By explicitly sequencing tasks and controlling progression, OSS-Doorway reduced the cognitive overhead associated with determining “what to do next,” a challenge commonly reported by newcomers to both OSS projects and GitHub-based courses. This aligns with prior work emphasizing the importance of scaffolding and guided pathways for novice contributors, and suggests that structured progression can be an effective complement to authentic tooling.

5.14.2 Engagement and Concept Reinforcement

Students perceived the quest-based workflow as more engaging than traditional assignments and reported that tasks reinforced lecture concepts. These findings indicate that breaking coursework into smaller, actionable units can support sustained engagement and incremental

learning. Rather than submitting large, one-off assignments, students interacted repeatedly with course material through GitHub, which may have strengthened the connection between conceptual instruction and practical application.

Importantly, engagement in this context does not appear to be driven primarily by gamification elements. Despite the presence of optional mechanics such as points or badges, students rated these elements as relatively unimportant. This suggests that structure, clarity, and feedback were more influential than overt gamified rewards, reinforcing the idea that gamification should function as supportive scaffolding rather than as the central motivator.

5.14.3 Improving the AI Validator

Our findings suggest that the AI validator is the highest-leverage component to improve because it sits directly in the core learning loop (submit → feedback → revise → progress). Students valued the idea of immediate validation, but both survey ratings and open-ended responses indicate that it did not consistently feel like an assessment of conceptual understanding. Instead, several students described needing specific keywords or preferred phrasing to earn acceptance, which encouraged “writing for the bot” and turned revision into guesswork rather than conceptual correction.

A priority is to make expectations explicit at the task level. When a task requires a particular structure (e.g., restating the prompt, including a definition and example, or referencing a specific concept), that requirement should be stated up-front so students do not discover it through repeated rejection. In parallel, validation should be aligned to concept-level criteria rather than surface cues: prompts and decision logic should reward semantically equivalent explanations, synonyms, and alternative phrasing that demonstrate the same understanding, while rejecting only when a required idea is missing.

A second priority is to make feedback more diagnostic and actionable. Students reported that some rejection messages were broad or unhelpful, especially for open-ended and artifact-based tasks. Feedback should distinguish between (i) missing conceptual elements and (ii)

formatting or submission issues, and should point students to what to add or clarify rather than directing them to reread instructions. For tasks involving diagrams or screenshots, the system should prefer more machine-checkable submission formats (e.g., links, structured fields, required captions) or route ambiguous cases to instructor/TA review, reducing brittle multimodal validation while preserving the benefits of rapid iteration.

5.14.4 Automated Feedback: Timeliness Versus Clarity

While students valued the availability of immediate feedback, the mixed ratings for feedback clarity and actionability reveal a critical tension. Timely feedback enabled iteration and reduced waiting time, but insufficiently specific or opaque feedback limited its usefulness in guiding revisions. This was particularly evident for tasks involving open-ended responses or non-textual artifacts, where students struggled to infer why a submission was rejected or how to improve it.

These findings highlight that automated feedback systems must balance speed with interpretability. Simply providing rapid accept or reject signals is not sufficient; feedback must communicate the underlying evaluation criteria and offer concrete guidance for improvement. Without this clarity, students may enter unproductive trial-and-error loops, diminishing the pedagogical value of automated validation.

5.14.5 External LLMs as Compensatory Tools

The observed pattern of external LLM usage provides additional insight into how students navigate feedback limitations. Students reported moderate use of tools such as ChatGPT or Copilot, but largely as a fallback rather than as a system-induced behavior. Qualitative responses suggest that LLMs were used to interpret unclear feedback, restate task requirements, or generate alternative explanations when system feedback was insufficient.

This behavior suggests that external LLMs functioned as compensatory scaffolding rather than as replacements for the system’s instructional role. From a design perspective, this

raises two implications. First, improving feedback clarity within the system may reduce reliance on external tools for interpretive support. Second, rather than viewing external LLM usage as inherently problematic, future systems could consider how to more explicitly integrate or acknowledge such tools in ways that align with learning objectives and feedback transparency.

5.15 Implications for System Design

Taken together, our results suggest several design implications for GitHub-centered educational systems. First, maintaining a GitHub-integrated interaction model can effectively support onboarding and reduce context switching. Second, task sequencing and explicit progression are valuable for guiding novices through complex workflows. Third, automated feedback mechanisms must prioritize clarity and actionability, especially for open-ended and artifact-based tasks. Finally, the presence of external LLMs in students’ workflows underscores the need to design feedback systems that either reduce the need for compensatory tools or integrate them in pedagogically meaningful ways.

This section describes the main changes implemented after the first deployment, organized by the problems they were intended to address.

5.15.1 Validator Robustness and Messaging

We implemented a set of changes to increase validator fairness, robustness, and actionability.

- **Reduced sensitivity to superficial formatting.** Validator prompting and acceptance criteria were adjusted to focus more on required concepts and evidence, rather than exact phrasing or punctuation.
- **More diagnostic feedback templates.** Rejection messages were revised to explicitly identify missing elements (“you did not address X”) and suggest next steps (“revise by adding Y and clarifying Z”).

- **Clearer distinction between concept errors and submission errors.** When possible, the validator was designed to distinguish between “your submission is missing the required artifact” versus “your explanation is incomplete,” so students could correct the right problem quickly.
- **Improved behavior in uncertain cases.** When the validator could not confidently determine correctness, we prioritized prompting students to provide additional evidence or to re-check a specific artifact, rather than issuing a confident but unreliable rejection.

To this end, we added explicit instructions to the LLM prompt to promote leniency;

Accept answers that show knowledge, even if they use different wording or phrasing than expected. Accept answers that include extra information that does not hurt their answer. Accept answers that have minor typos, capitalization errors, or grammatical mistakes. Accept answers that provide partial answers that demonstrate understanding. Accept answers that use synonyms or alternative expressions for key terms. Accept answers that give examples instead of definitions, or definitions instead of examples, if both show knowledge. Accept answers that have typos in git commands or technical terms. Accept answers that use different separators or formatting in commands.

Likewise, we added examples on how the validator should interpret variance in student answers.

If asked for 10/15 as a Date, accept 10/15/2025, Wed 10/15, Wednesday 10/15 or October 15th.

Only reject answers that are completely wrong, show no understanding, or are completely off-topic. If the student demonstrates any understanding of the concept, give them the benefit of the doubt.

Because in the context of CS499/599, we would rely on other, more OSS-specific task types like iterative code review and bot review, we preemptively improved the prompting

to be even more specific and helpful, as well as point to actual code, and not just reply with vague summaries, though this task type hadn't been tested out just yet.

If the code does not meet all requirements, respond with constructive feedback.

Format feedback as a bulleted list of specific code issues.

Be specific and reference actual code patterns, lines, functions, or conditions.

Point to direct parts of the code with concrete examples.

List all issues found, not just a summary.

Use formats such as:

- `function_name()` can crash when ...
- `if (condition)` is wrong because ...
- The line with `X` can fail on `Y`.

Reference actual code snippets when possible.

These changes were aimed at shifting the retry loop from guesswork to learning-oriented revision.

5.15.2 Reliability, Failure Handling, and System Hardening

Between deployments, we also prioritized operational improvements that affect classroom experience.

- **More informative failure states.** When the system could not validate due to an external issue (timeout, service error), feedback was updated to explicitly say so and to instruct students how to proceed (retry later, check status, or contact the instructor if persistent).
- **Instructor-side configuration checks.** We improved guardrails that help catch

misconfigured tasks before students encounter them, reducing the chance that students are blocked by instructor-side setup errors.

- **Configuration caching and invalidation.** We implemented an in-memory cache for quest configurations, keyed by class/group context, to reduce repeated database reads and improve response latency during validation. Cached entries use a time-to-live policy (default: one hour) and are invalidated when configuration updates are saved or deployed.
- **Resilient cache-backed loading path.** Configuration loading now follows a layered strategy: cache first, then database (with bounded retry and exponential backoff), then file-system fallback. Successful fallback loads are re-cached and can be migrated back to the database, while misses degrade gracefully to default behavior rather than failing the student-facing flow.

These steps were intended to reduce “invisible” friction that students experience as inconsistency or unfairness.

5.15.3 What This Refinement Process Contributes to the Thesis

This section details the first steps taken to address friction points derived from the post-course survey. It represents a crucial part of the method: an iterative approach to improving a gamified learning environment.

- It shows how small design elements (feedback phrasing, recovery guidance) can become high-leverage components when a system is embedded in the workflow.
- It demonstrates that validator design is not a purely technical problem; it is a fairness and trust problem, and it must be treated as such in educational settings.
- It provides a concrete example of a deployment-data-driven refinement pipeline that can be applied beyond OSS-Doorway to other workflow-embedded learning systems.

In the next chapter, we detail the nature of the CS499/599 deployment and evaluate its results to examine how the refined system was experienced in the second deployment, and where friction persisted despite these improvements.

5.16 Limitations

This first deployment has several limitations that shape how its findings should be interpreted and how they are used in the broader thesis.

First, the CS386 deployment took place within a single course context at one institution. The student cohort, course structure, and instructor expectations do not reflect open source software development courses and may not fully reflect other software engineering courses, curricula, or student populations. As a result, observations from this deployment should be treated as context-specific, rather than broadly generalizable claims about GitHub-based coursework systems.

Second, our evidence in this deployment is primarily descriptive and deployment-driven. The goal of CS386 was to observe how OSS-Doorway operates in a realistic classroom setting and to identify friction points that affect student progress. We do not claim causal impacts on learning outcomes. In particular, we did not use a controlled comparison condition (e.g., a parallel group using a traditional LMS or non-gamified GitHub assignments), and we did not collect standardized pre-post measures that would allow us to quantify skill gains attributable to OSS-Doorway.

Finally, OSS-Doorway is a configurable system, and the deployment experience is partly shaped by authoring choices such as task wording, sequencing, and validation criteria. This means that some friction points may reflect the design of the specific quest set used in CS386 rather than limitations of the platform itself. These constraints reinforce the role of CS386 as a formative deployment intended to surface issues early and guide improvements, rather than as a definitive evaluation.

5.17 Final Considerations

The CS386 deployment provided an essential baseline for understanding OSS-Doorway as a GitHub-integrated learning environment. It demonstrated that progressive quests and task-based structure can operationalize software engineering concepts into authentic GitHub workflows, while also revealing where usability and feedback design can undermine student momentum. The main contribution of this first deployment is therefore formative: it exposed practical barriers related to onboarding clarity, submission conventions, task granularity, and the interpretability of automated feedback.

These insights directly motivate the refinement-driven arc of this thesis. Rather than treating CS386 and CS499/599 as independent studies, we use the first deployment to identify high-friction points and then evaluate whether targeted changes in the second deployment lead to a stronger student experience and clearer evidence about what OSS-Doorway supports well. In this sense, CS386 establishes both the initial design baseline and the rationale for the specific improvements introduced in CS499/599.

CHAPTER 6 CS499/599 Deployment: Subsequent Classroom Study of OSS-Doorway

6.1 Overview and Study Motivation

This chapter presents the second classroom deployment of OSS-Doorway in a mixed undergraduate/graduate open source software development course (CS499/599- Open Source Software, Spring 2026). In the context of CS499/599 OSS-Doorway delivers coursework as structured sequences where students submit evidence through authentic repository actions (e.g., commits, pull requests, and code review) and receive automated validation and feedback directly in the same threads. Compared to CS 386, the coursework focused solely on open source software development and contribution, putting students through the contribution process, progressing to more complex and expert OSS tasks like pull requests and exploring GitHub actions, and finally creating their own OSS project. This is in contrast with the more general software engineering concepts for the first deployment.

This deployment is intentionally positioned as a refinement over the first deployment. The CS386 results highlighted a clear strength of OSS-Doorway: students valued the hands-on structure and the reduced ambiguity created by sequencing tasks into quests within GitHub and revealed high-leverage friction points, especially around early interaction conventions (e.g., where to submit answers, what counts as a valid submission) and the interpretability of automated validation on open-ended tasks (format sensitivity, non-diagnostic rejection messages, and perceived inconsistency). The CS499/599 deployment, therefore, serves two purposes: (i) to evaluate OSS-Doorway in a course context that is closer to the contribution workflows it aims to teach (including pull-request-centric work and code review), and (ii) to

assess whether changes informed by CS386 improved the student experience, particularly in terms of workflow clarity and validator usability.

The CS499/599 deployment also matters because it targets a different point in the curriculum. While CS386 students are often encountering software engineering workflows for the first time, CS499/599 students were expected to engage more directly with OSS practices and with the coordination mechanisms that structure real OSS contributions and were assumed to have more experience with code contribution and software development. As a result, this study offers an opportunity to test whether OSS-Doorway remains useful when tasks move from introductory software engineering concepts to more advanced, contribution-like activities such as branching/merging, opening pull requests, responding to review feedback, and configuring CI through GitHub Actions. Essentially, CS499/599 gives us the opportunity to study OSS-Doorway’s efficacy as originally planned.

In this chapter, we evaluate student perceptions of OSS readiness, GitHub self-efficacy, workflow clarity, and validator feedback quality using a post-course survey and open-ended responses, and we interpret the results through the lens of the changes motivated by the CS386 deployment.

6.2 Survey Constructs and Guiding Questions

For CS499/599, the post-course instrument is organized around five analytic survey constructs. These are construct-level question groups used to organize item blocks, compute composite measures, and interpret quantitative and qualitative evidence about OSS-Doorway’s usefulness and friction points.

- **Perceived OSS readiness.** We measure, after completing OSS-Doorway quests, how prepared do students feel to participate in open source software contribution workflows (e.g., understanding the typical process, feeling capable of making a meaningful contribution, and feeling less intimidated about contributing)?

- **GitHub self-efficacy.** This construct organizes questions on how confident students feel performing core GitHub actions required for OSS participation (e.g., using the issue tracker, understanding pull requests, opening pull requests through GitHub, and contributing to projects more broadly)
- **Task scaffolding and tool experience.** This group assesses whether students experienced the quest-to-task flow as clear, actionable, and easier to follow than traditional assignments, and whether the overall GitHub-based workflow felt usable.
- **Validator feedback quality.** Here we focus on how students perceive the quality of automated feedback in OSS-Doorway, particularly whether it identifies specific mistakes and provides actionable next steps that influence revisions during task completion
- **Gamification and overall recommendation.** We measure students' perception of the motivational value of quest progress visibility and related gamified elements, and how willing are they to recommend OSS-Doorway to a peer?

To make the analysis traceable from thesis-level research questions to evidence, Table 6.1 provides a single mapping between each RQ, the quantitative measures used, the qualitative evidence used, and where answers are synthesized in this chapter.

6.3 Research Design

We conducted a classroom deployment study in CS499/599 using a post-course survey as the primary evaluation instrument, complemented by open-ended responses.

Intervention

The intervention in this deployment is the use of OSS-Doorway as the primary workflow for interacting with course activities. Tasks required students to execute authentic actions

Table 6.1: CS499/599 mapping from thesis research questions to data sources and answer locations.

RQ	Quantitative measures	Qualitative sources	Where answered
RQ1 (scaffolding / workflow learning)	OSS readiness, GitHub self-efficacy, task scaffolding/tool experience, with gamification and recommendation used as supporting perception context	Themes on hands-on GitHub learning, progressive quest structure, and onboarding clarity	Results and discussion synthesis in Sections 6.11, 6.12, and 6.14
RQ2 (AI feedback / validation)	Validator-labeled composite	Themes on validator friction, format sensitivity, and actionability of feedback	Validator-focused results and integration in Sections 6.11 and 6.12
RQ3 (design lessons across deployments)	Cross-deployment quantitative pattern summaries from CS386 and CS499/599 (no single composite)	Cross-study synthesis of themes and documented refinement actions between deployments	Cross-deployment interpretation in Sections 6.14, 6.17, and 6.20

that map directly to OSS contribution mechanics (e.g., branching and merging, opening pull requests, responding to review feedback, and configuring automated checks through GitHub Actions). OSS-Doorway enforced progression through quest sequencing and prerequisite logic, so that later tasks depended on successful completion of earlier workflow steps. This design operationalizes “progressive onboarding” by breaking complex workflows into manageable units while maintaining authenticity through GitHub-based artifacts.

Instrument and measures

The survey instrument was designed to capture students’ perceptions of: (i) OSS readiness, (ii) GitHub self-efficacy, (iii) task scaffolding and tool experience, (iv) validator feedback quality, (v) perceived value of gamification, and (vi) overall recommendation. Items were measured on Likert-type scales (with constructs using either 5-point or 7-point scales depending on the battery), and we included open-ended prompts to elicit concrete descriptions of what students found most helpful, most frustrating, and what they would change. The open-ended questions were particularly important for diagnosing whether the improvements motivated by CS386 addressed the highest-leverage sources of friction (especially validator

interpretability and onboarding clarity).

Analysis approach

We analyze Likert-style items using descriptive statistics (means and standard deviations) and compute construct-level composite scores by averaging items within each construct. For multi-item constructs, we report internal consistency using Cronbach’s α to summarize the coherence of each battery. For the overall recommendation item, we report both the raw distribution summary (mean and standard deviation) and the Net Promoter Score breakdown (Promoters, Passives, Detractors) to align with standard interpretation conventions.

Open-ended responses are analyzed using an inductive thematic analysis. We identify recurring themes that reflect perceived strengths, pain points, and improvement requests, and we triangulate these themes with the quantitative results (e.g., using qualitative feedback to explain why a construct scored lower or to contextualize a high score). Because this is a real-course deployment rather than a controlled experiment, we avoid causal claims and instead frame results as evidence about perceived usefulness, workflow experience, and the fit of OSS-Doorway as a scaffolding mechanism for GitHub-centered OSS education.

Link to thesis arc

Finally, the CS499/599 deployment is explicitly informed by CS386. Rather than treating the deployments as independent evaluations, this chapter positions CS499/599 as the next iteration in an evidence-driven cycle: CS386 surfaced concrete weaknesses (especially around validator actionability and early interaction conventions), which motivated tool refinements, and CS499/599 evaluates the revised system in a course context that more directly targets OSS contribution workflows. This structure supports the main thesis narrative of using results from the first deployment to improve the system before the second deployment.

6.4 Course Context and Timeline (CS499/599)

The second deployment and integration of OSS-Doorway took place in CS499/599 during the next semester; Spring 2026, in a mixed undergraduate/graduate open source software development course. In contrast with OSS-Doorway being used for assignments throughout the semester, as it was in CS 386, OSS-Doorway was deployed for a 6-week stint where the bot delivered course activities as a sequence of quests and tasks every week. Students interacted with OSS-Doorway entirely through GitHub, using the README as the entry point for task discovery and using GitHub artifacts (branches, commits, issues, pull requests, and workflows) as the primary submission and validation surface.

Coursework deployed via OSS-Doorway for the course were organized as seven quests (Q1–Q7) that progressed from Git fundamentals to OSS project setup. Specifically, students completed tasks on branching and merging (e.g., interpreting commit trees, resolving merges, and using merge-abort workflows), explored GitHub project affordances (issue tracker, pull request interface, forking, repository documentation and contributor views), practiced contribution workflows (forking the course repository, committing work, and opening pull requests), and completed a pull request and code review sequence involving implementation work followed by review and revision. Students also completed a CI-oriented quest in which they created a small Python module, wrote tests, and configured a GitHub Actions workflow. Finally, students set up their own OSS-style repository by authoring core community-facing artifacts, including a license rationale, a comprehensive README, contributing guidelines, and a code of conduct. Overall, the course structure ensured that students repeatedly practiced OSS-relevant workflows using the same tools and interaction patterns found in real OSS projects.

Each new quest was deployed at the end of the school week, set to be completed in a week by students outside the classroom and individually, with the exception of Q2 and Q3, which were deployed together and had the same completion window. Following the

deployment, we evaluated OSS-Doorway using a post-course survey designed to capture students’ perceptions of OSS readiness and GitHub self-efficacy, task scaffolding, and overall tool experience, validator feedback quality, and perceived value of the quest structure and gamification layer.

6.5 Participants

The course enrolled 30 students (22 undergraduate and 8 graduate). Of those, **23** students completed the post-course survey, and these 23 responses form the sample used for every quantitative analysis in this chapter (items scored listwise within each composite). In addition to Likert-style items, the survey collected open-ended reflections and background information (e.g., course level and prior GitHub/OSS experience) to contextualize the responses. Survey data were analyzed and reported in aggregate, as well as individually.

6.6 Measures and Instruments

We evaluated OSS-Doorway using a post-course survey designed to capture multiple dimensions of student experience and perceived outcomes. The instrument included multi-item constructs measured on Likert-type scales, along with open-ended questions intended to elicit specific examples of what students found helpful or frustrating.

Construct provenance (adapted vs. author-created). For transparency, we distinguish reused versus project-authored wording. The motivation/value items were adapted from Self-Determination Theory style intrinsic-motivation/value prompts and reworded to OSS-Doorway/quest context (Q44). Confidence items were adapted from GitHub/OSS self-efficacy style wording and contextualized to course tasks (Q36). OSS readiness and contribution-intention items (Q45–Q46) were adapted from common OSS-onboarding readiness formulations and rewritten for this deployment context. Task-scaffolding, tool-experience, validator, and gamification items (Q6, Q40, Q11, Q47), plus recommendation (Q49), were

authored for OSS-Doorway evaluation and aligned to the study goals from CS386 refinement.

The survey measured the following primary constructs:

Perceived OSS readiness. This construct captures whether students felt prepared to engage with OSS contribution workflows beyond the course. Items focus on understanding the typical workflow, feeling capable of making a meaningful contribution, and reduced intimidation. In the CS499/599 context, this construct reflects both conceptual readiness; knowing the steps to contribute to OSS projects, and affective readiness; confidence and reduced barrier perception.

GitHub self-efficacy. This construct captures confidence in performing core GitHub actions required for OSS contribution, such as navigating the issue tracker, understanding pull requests, opening pull requests, and contributing through GitHub more broadly. Because these actions map directly to the skills practiced in quests, this construct provides a focused measure of workflow confidence rather than general attitudes.

Task scaffolding and tool experience. This construct captures students' perceptions of clarity and actionability of tasks, and the usability of OSS-Doorway as a system. Items include whether tasks improved understanding of course requirements, whether they were more actionable than standard assignments, whether OSS-Doorway was easy to use, and whether it was useful for learning OSS contribution.

Validator feedback quality This construct captures whether automated feedback supported revision. Items focus on whether feedback influenced how students adjusted their answers and whether it helped identify specific mistakes. This construct is particularly important given that the first deployment highlighted validator interpretability and format sensitivity as major friction points.

Perceived value of gamification and progress cues. This construct captures whether visible progress through quests motivated continuation, whether gamification made the experience more engaging, and whether students enjoyed the quest workflow. In this study, we interpret these items primarily as reflecting the value of structured progression and progress

visibility, rather than the value of surface rewards alone.

Overall recommendation. We measured overall recommendation using a 0–10 likelihood-to-recommend item. We report both descriptive summary statistics and a Net Promoter Score (NPS) breakdown (Promoters/Passives/Detractors) to align interpretation with common practice.

In addition to these quantitative measures, the survey included open-ended prompts asking students what they found most helpful, most frustrating, and what they would change. These responses serve as an important diagnostic complement to the Likert items, especially when quantitative scores indicate a generally positive experience while qualitative feedback reveals specific breakdowns (e.g., cases where the validator rejected correct work due to formatting).

6.6.1 Survey instrument structure

Students completed the Institutional Review Board-approved post-course survey administered in Qualtrics. The questionnaire opened with consent and contextual framing about OSS-Doorway in GitHub, then asked how many deployed quests students completed, followed by a multipoint matrix whose stems follow Self-Determination Theory practice, GitHub self-efficacy confidence items, agreement-style OSS readiness and intention-to-contribute blocks, an extended OSS-Doorway tool survey, gamification probes, task-scaffolding items, a validator-information block with Likert items targeting feedback influence and diagnostic value, a single 0–10 likelihood-to-recommend item, thematic open prompts, and demographic information.

Table 6.2 summarizes how survey blocks map to the composite scales used in this chapter. The full stems appear in Appendix B.

Table 6.2: CS499/599 analytic constructs, Qualtrics columns used in composites, and instrument-source provenance.

Construct / focus	Qualtrics columns in composite	Source
Perceived OSS readiness (1–7)	Q45 _{1–5} , Q46 ₁	Adapted/contextualized
GitHub self-efficacy (1–5)	Q36 _{1,2,5,7}	Adapted/contextualized
Task scaffolding / tool experience (1–7)	Q6 _{1,3} , Q40 _{1,9}	Author-created
Validator feedback quality (1–5)	Q41 _{1,4}	Author-created
Gamification / contribution intent (1–7)	Q44 _{1,11} , Q47 _{2,3} , Q45 ₆ , Q46 ₄	Mixed (adapted + author-created)
Overall recommendation (NPS; 0–10)	Q49	Author-created
Open responses / demographics	Q16, Q37, Q38, Q17; Q31, Q43, Q33–Q35	Author-created

6.7 Data Sources and Collection

The primary data source for this evaluation is the post-course survey, which includes both Likert-style items and open-ended responses. Survey responses were collected electronically during the semester after the six week period when most students had completed a substantial portion of OSS-Doorway quests. Because the course is mixed undergraduate/graduate, we also collected background items that characterize respondents (e.g., course level and prior experience) to contextualize interpretation.

All collected data were handled in a manner consistent with confidentiality expectations in classroom research. Survey responses were treated as confidential and are reported in aggregate; open-ended responses are presented as short excerpts without identifying information.

6.8 Data Preparation

We computed construct-level composite scores by averaging items within each construct. For constructs with multiple items, we calculated internal consistency using Cronbach’s α to summarize whether items in the same construct behaved coherently. When individual items had missing responses, we used construct-level averaging only when a respondent provided

enough items to yield a stable mean; otherwise, we treated the construct value as missing for that respondent and reported item-level N where relevant.

Secondly, we prepared the overall recommendation item in two forms: (i) a raw descriptive summary (mean and standard deviation), and (ii) an NPS classification where 9–10 are Promoters, 7–8 are Passives, and 0–6 are Detractors. We computed NPS as %Promoters minus %Detractors.

Finally, we prepared open-ended responses for qualitative analysis by removing identifying details (when present), splitting multi-topic answers into analyzable segments, and grouping responses by prompt to preserve context (e.g., separating “most helpful” from “most frustrating” prompts).

6.9 Data Analysis

We analyze quantitative items using descriptive statistics, reporting means and standard deviations for each item and for each construct-level composite. Because the purpose of this deployment study is to characterize student perceptions rather than establish causal impact, descriptive statistics provide an appropriate summary of overall experience while keeping interpretation grounded in the observed responses.

For multi-item constructs, we compute composite means by averaging the items within each construct and report Cronbach’s α as a measure of internal consistency. We interpret composites alongside item-level results rather than treating a single composite score as sufficient, since item-level patterns can reveal specific strengths and weaknesses that composites may obscure (e.g., high overall validator score but persistent complaints about non-actionable rejection messages for open-ended tasks).

We analyze open-ended responses using inductive thematic analysis. We first performed an initial pass to identify recurring topics and recurring types of friction (e.g., submission conventions, validator format sensitivity, clarity of instructions, pacing through quests). We

then grouped responses into themes and refined theme definitions iteratively, ensuring that themes captured the most frequent and most consequential issues as described by students. Finally, we triangulated qualitative themes with quantitative findings to provide explanation and nuance. For example, when an item or construct scored lower, we used qualitative responses to clarify whether the cause was usability friction, unclear expectations, or validator feedback that did not provide a clear next step.

Throughout, we avoid overclaiming. Results are interpreted as evidence about perceived readiness, confidence, and experience in a specific course context, and as input into an evidence-driven iteration cycle where findings inform further improvements to OSS-Doorway.

6.10 Ethical Considerations

This study involves minimal risk to participants and is conducted in an educational setting where OSS-Doorway is part of the normal coursework experience. Participation in the post-course survey was given as an extra credit assignment, with students being able to decline participation without penalty, and survey responses were treated as confidential. Results are reported only in aggregate, and any excerpts from open-ended responses are presented without identifying information.

OSS-Doorway’s automated validation is used as a formative mechanism to support learning and revision during task completion, not as an opaque grading authority. Similarly, survey responses are used to understand experience and guide system improvement rather than to evaluate individual performance.

Finally, we acknowledge that students may have used external tools during the course (including external LLMs) and that this behavior is difficult to control in realistic settings. Rather than attempting to prohibit such use, we treat it as part of the natural classroom environment and interpret findings accordingly. The goal of this evaluation is not to police tool use, but to assess whether OSS-Doorway provides clear scaffolding and actionable

Table 6.3: Construct-level survey results for CS 499/599.

Construct	Mean	SD	α	N
Perceived OSS readiness (1–7)	5.78	0.79	.887	23
GitHub self-efficacy (1–5)	4.63	0.57	.860	23
Task scaffolding and tool experience (1–7)	5.79	1.31	.926	23
Validator feedback quality (1–5)	4.17	0.77	.844	20
Gamification and contribution intent (1–7)	5.40	0.89	.735	23
NPS (0–10), single item	7.43	2.19	—	23

Note. All composites use the $N = 23$ Finished subsample. The validator composite shows $N = 20$ because two respondents left both validator items, and one additional respondent left just one validator blank; those three cases are dropped only from the validator composite. Every other construct in this table has no missing values and is computed on the full $N = 23$.

feedback that supports learning within authentic GitHub workflows.

6.11 Results

This section reports results from the CS499/599 OSS-Doorway post-course survey and summarizes the open-ended feedback. Quantitative composites are based on the Finished subsample ($N=23$), except in cases where complete responses were not available in the Qualtrics export. The validator feedback composite averages the quest-usefulness items; three respondents were missing one or both valid responses in the archival spreadsheet, resulting in $n=20$ rows for Table 6.3 and Figures 6.1–6.5. No respondents were excluded from the other constructs. The sections below present results for OSS readiness, task scaffolding and tool experience, gamification, GitHub self-efficacy, and the validator feedback composite, followed by a thematic synthesis of the open-ended responses.

Overall, the composite results were positive across all five constructs. The strongest signals were in perceived OSS readiness ($M = 5.78$, $SD = 0.79$, 1–7) and task scaffolding and tool experience ($M = 5.79$, $SD = 1.31$, 1–7), suggesting that students generally felt prepared to engage in OSS workflows and found the quest-based structure clear and usable. GitHub self-efficacy was also high ($M = 4.63$, $SD = 0.57$, 1–5), indicating strong confidence in performing core GitHub actions after the course. Ratings of the validator-labeled composite

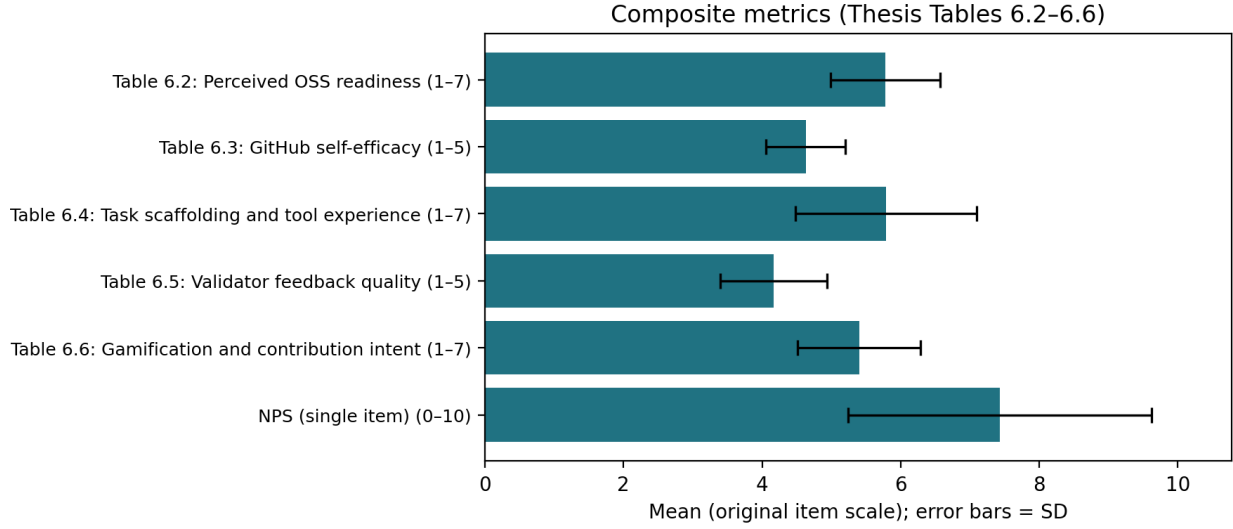


Figure 6.1: Composite metrics (Thesis Tables 6.2–6.6 + NPS).

(average usefulness of Quests 1 & 4 drawn from Qualtrics Q41) were favorable ($M = 4.17$, $SD = 0.77$, scaled 1–5 when mapping ordinal labels; $N = 20$ after dropping rows missing either cell), aligning with Figures 6.1 and 6.5, while thematic evidence still underscores friction with the validator; I revisit that nuance alongside open-ended critiques. Perceived value of gamification and contribution intent was positive ($M = 5.40$, $SD = 0.89$, 1–7), with the lowest Cronbach’s α among multi-item scales (.735), consistent with qualitative comments that quest structure mattered more than reward mechanics.

6.11.1 Perceived OSS Readiness

For this construct, we use survey composites and items (Figures 6.1 and 6.2) together with thematic analysis of open-ended responses (Section 6.12). Respondents in our second integration of OSS-Doorway reported high perceived readiness for OSS participation and confidence with core open-source contribution concepts. Overall composite ratings were positive (composite mean = 5.78 out of 7, $SD = 0.79$; $N = 23$), indicating that most students left the quest sequence feeling capable of executing the main mechanics of contribution workflows and understanding what participation in OSS communities typically entails.

Table 6.2: Perceived OSS readiness (1-7) (N=23)

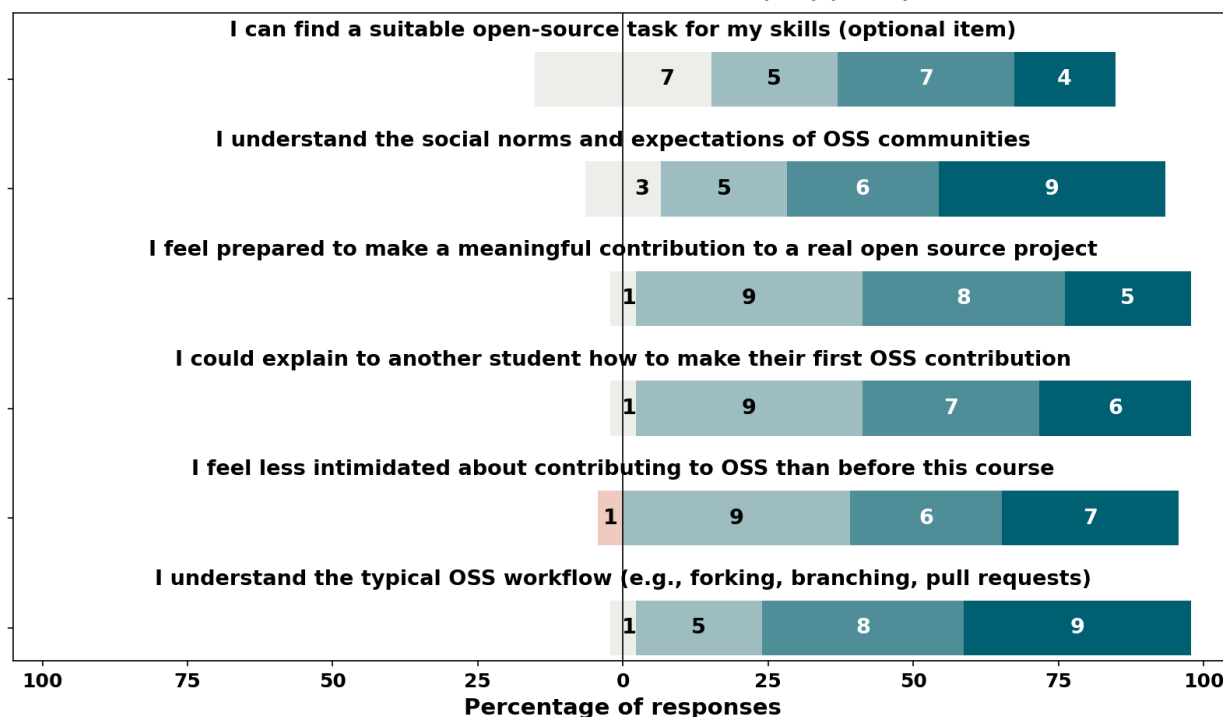


Figure 6.2: Perceived OSS readiness (1–7) ($N = 23$ Finished).

As shown in Figure 6.2, agreement was strongest on understanding the typical OSS workflow (22/23 in the top-two agreement bins). Ratings were similarly high for understanding OSS social norms and expectations (20/23) and being able to explain the first-contribution process to a peer (22/23). These results suggest that the combination of structured quests and repeated exposure to GitHub-based tasks supported both procedural knowledge (what steps to take) and situational knowledge (how OSS communities function).

Furthermore, in students’ comments, perceptions of readiness line up closely with themes about hands-on, GitHub-based practice and progressively guided quests: many described OSS-Doorway as making contribution steps tangible and easier to relate to OSS work compared with abstract assignments (Themes 1–2 in Section 6.12).

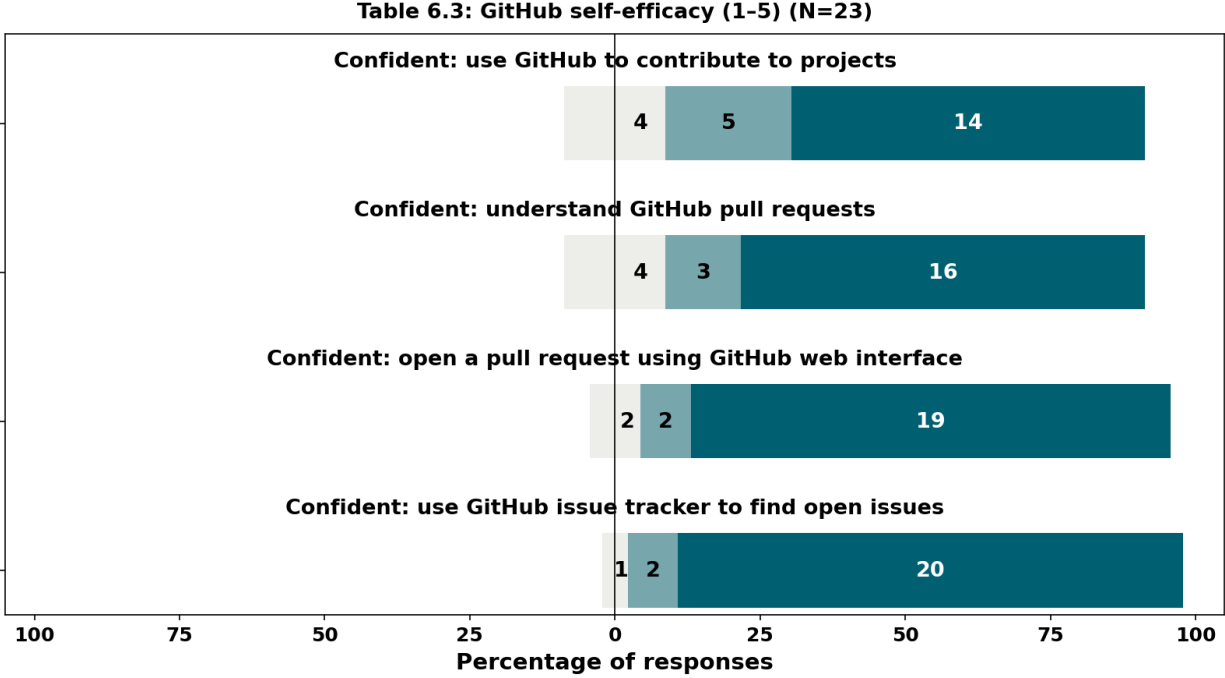


Figure 6.3: GitHub self-efficacy (1–5) ($N = 23$ Finished).

6.11.2 GitHub Self-Efficacy

We use the GitHub self-efficacy composite and item distributions (Figures 6.1 and 6.3) together with thematic analysis (Section 6.12). Students also reported high confidence in executing GitHub-specific tasks required for OSS contribution. Across the four self-efficacy items (1–5 scale), the overall composite was strong (composite mean = 4.63 out of 5, $SD = 0.57$; $N = 23$), indicating that most respondents felt capable of using core features such as the issue tracker and pull requests after completing the quests.

As shown in Figure 6.3, confidence in using the issue tracker to find open issues was highest (22/23 in the top-two agreement bins). Confidence was also high for opening a pull request through the web interface (21/23). Understanding pull requests (19/23) and contributing to projects more broadly (19/23) were comparatively lower, but still well above the midpoint. This pattern suggests that concrete, repeatable actions (e.g., locating issues, opening PRs) were the easiest confidence gains, while more conceptual understanding (e.g., what a PR represents across workflows) remained a slightly weaker area.

Qualitative comments help explain why self-efficacy was highest for narrow, repeatable actions yet still solid for broader items: repeated practice inside issues, pull requests, and repository tasks supported confidence through doing rather than conceptual coverage alone (Theme 1, Section 6.12).

6.11.3 Task Scaffolding and Tool Experience

For the third construct, we use the task scaffolding and tool experience composite and items (Figures 6.1 and 6.4) together with thematic analysis (Section 6.12). Students rated task scaffolding and overall tool experience positively. Across the four items (1–7 scale), the composite mean was 5.79 out of 7 ($SD = 1.31$; $N = 23$), indicating favorable perceptions of instructional clarity and workflow usability in the second deployment.

As shown in Figure 6.4, students agreed that the tasks improved their understanding of class requirements (21/23 in the top-two agreement bins) and provided more actionable guidance than standard assignments (20/23). Ease of use was rated highly (22/23), reinforcing that students generally experienced the GitHub-based workflow as usable and not overly burdensome. The overall usefulness of OSS-Doorway for learning OSS contribution was also rated positively (18/23).

Themes about progressive structure and pacing match the high scaffolding scores while also narrowing where tension remains: comments still pointed to onboarding and submission cues as something that needed to click early before the scaffolding felt effortless (Themes 2 and 5, Section 6.12).

6.11.4 Validator Feedback Quality and Actionability

We pair the exploratory composite plotted as validator feedback (perceived usefulness tied to iterative Git-centric quests most exposed to OSS-Doorway checking) with themes about checker format sensitivity, rejection clarity, and shot-gunned revisions (Section 6.12). Those ordinal averages remain positive ($M = 4.17$, $SD = 0.77$; $n = 20$ after three respondents

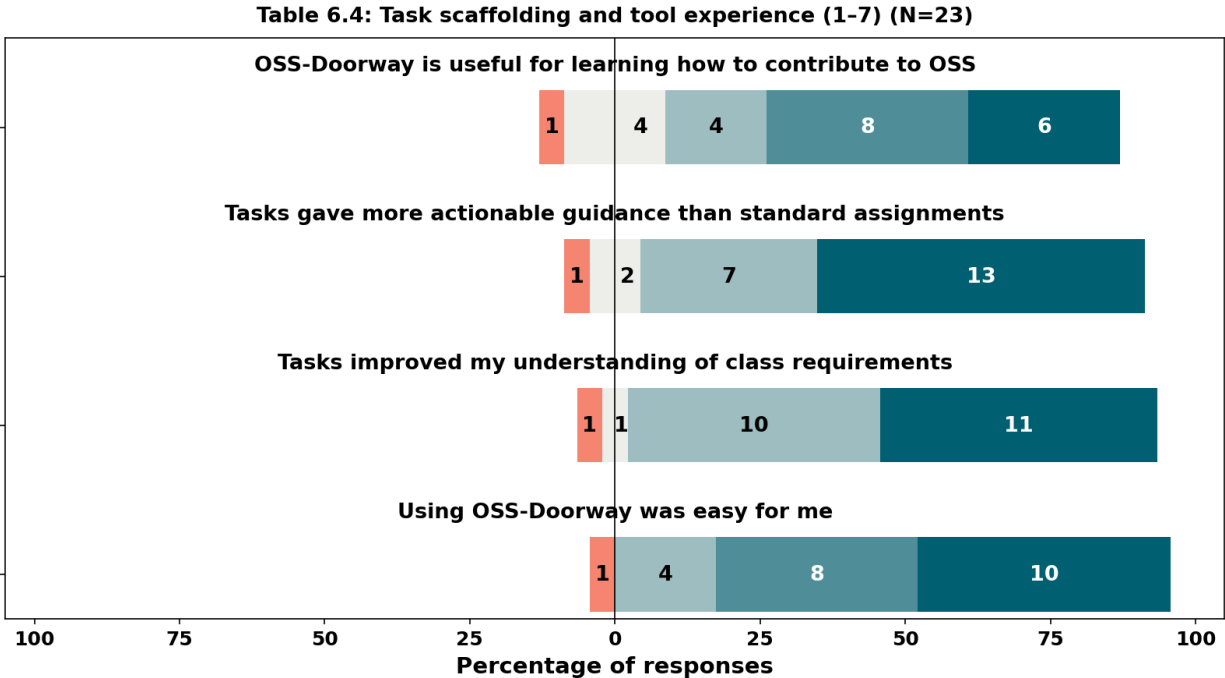


Figure 6.4: Task scaffolding and tool experience (1–7) ($N = 23$ Finished).

lacked fully coded pairs in the archival export), illustrating that pedagogical usefulness lingered despite validator gripes articulated elsewhere. Dedicated checker Likert wording appears in Appendix B.

Figure 6.5 visualizes the ordinal distribution underlying the averaged composite; overall sentiment remains favorable even while open answers document validator-specific friction. Interpretation therefore triangulates these indices with thematic complaints about brittle automated feedback whenever students describe “writing for the bot.”

6.11.5 Gamification and Overall Recommendation

Lastly, we use the gamification-related survey composite and items (Figures 6.1 and 6.6), the willingness-to-recommend item summarized as NPS-style scores (Section 6.13 and Table 6.4), and thematic analysis (Section 6.12, Theme 6). Students generally viewed the gamified structure positively, particularly when it functioned as a progress-and-structure layer rather than as a competitive reward system. Across six items (1–7 scale), the gamification composite

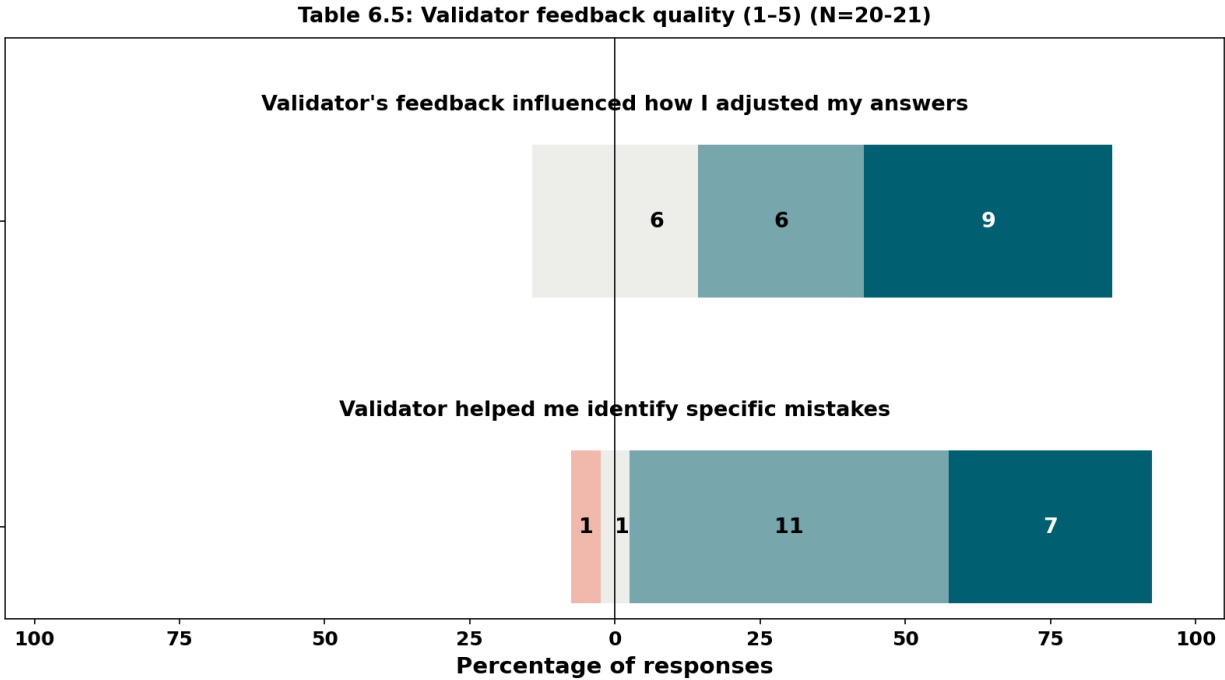


Figure 6.5: Validator-labeled composite (1–5) after removing rows with missing entries or “N/A,” $N = 20$.

was 5.40 out of 7 ($SD = 0.89$; $N = 23$), suggesting that the quest-based progression and progress visibility supported engagement and persistence for many students.

Figure 6.6 illustrates that strong signals included being more likely to contribute to OSS after the experience (22/23) and intending to contribute outside class (20/23). Lower agreement appeared on whether gamification made the learning experience more engaging (16/23), suggesting that the structural mechanics (quests/progression) may matter more than the explicit “game” framing for some students.

Theme 6 aligns with Figure 6.6’s tilt toward contribution intent versus excitement about superficial rewards: many students framed progress indicators as a checklist or dashboard rather than gameplay, while points or badges attracted ambivalence unless they felt credible (Theme 6).

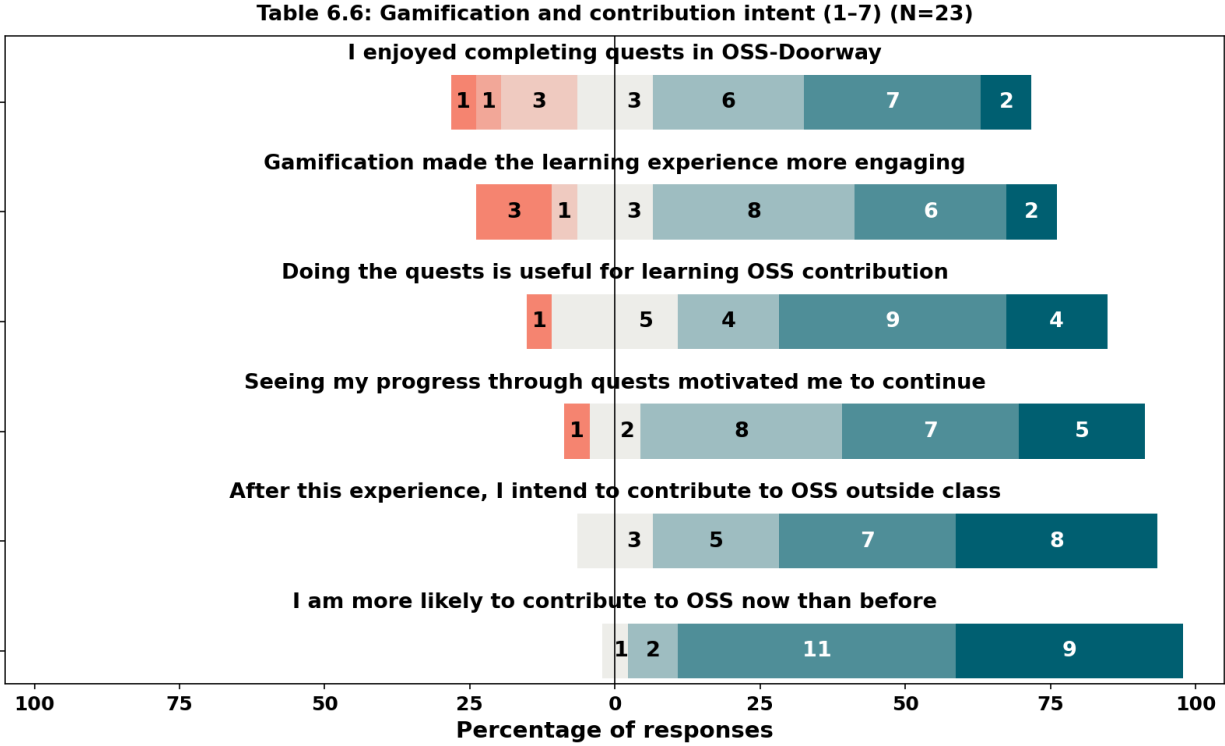


Figure 6.6: Gamification and contribution intent (1-7) ($N = 23$ Finished).

6.12 Open-Ended Responses: Themes from Qualitative Feedback

Open-ended responses reinforced the quantitative results and helped explain, in more detail and in much more clarity, *why* students rated OSS-Doorway positively overall, while also clarifying the main sources of friction. Across themes, students emphasized the value of hands-on GitHub work and structured progression, while frustrations centered on validator reliability/format sensitivity and early onboarding clarity. Below, we summarize six themes and include representative excerpts.

Theme 1: GitHub-Based, Hands-On Learning

Students frequently described OSS-Doorway as more authentic and engaging than traditional assignments, largely because the workflow kept them inside GitHub and required them to

complete real contribution-style steps.

When asked questions about learning and overall use, students overwhelmingly acknowledged the sheer value of OSS-Doorway’s structure of doling out tasks on GitHub and the iterativeness and instant feedback mechanism of the bot.

“I think this was a lot more fun. I get kind of sick with Canvas assignments and needing to tab back and forth, so having everything in one place was great.”

“The activity as a whole was well structured in a way that actually helped me learn. Without the bot fixing the file issues and learning how actions work would’ve taken me far longer.”

“I think this speeds up the process and also encourages learning, before it would be more similar to typical homework structure being that we would turn in HW and have to await a response about being right or wrong. I would also say that this helps us to use AI in a helpful way vs using it as a crutch ...”

In line with spotlighting the importance of OSS-Doorway’s learning style, responses also underscore the importance of OSS-Doorway as an entry point or *doorway* to GitHub and collaborative software engineering.

“ ... A fair amount of CS topics can be rather text heavy, and I feel gamification can help ease newer students in. That being said– I’ve had struggles in the past with my class mates not understanding GitHub, I don’t mean to be hyperbolic, but I think OSS-Doorway is ESSENTIAL in order to help new students understand GitHub (which is essential to a CS career). As such, it helped it a lot, and I think we should move it into an introductory class to teach students GitHub early, allowing them to save their code and get started in OSS maybe in Freshman year instead of the Senior one”

Likewise, an individual who was highlighted being instructed by the bot to work through

the actual coding process was an aspect of OSS-Doorway that was the most helpful for your learning, another participant said that they found learning of GitHub Actions the most useful.

“The general path that the Doorway bot required me to follow benefited my learning the most, as I was forced to physically work through the process. This helps me to remember and understand how to do things better than learning about it from a high level.”

“I like the quests and how the bot gives hints toward assignments. I think its like giving the nudge without giving all the answers.”

Overall, these responses suggest that OSS-Doorway’s main value in CS499/599 was not simply that it delivered content, but that it *forced practice* in an authentic GitHub environment. Students repeatedly framed the experience as more practical than Canvas-style homework because the bot and quests turned course concepts into concrete contribution steps (e.g., working through pull requests, code review, and GitHub Actions) and supported iteration through immediate feedback. Taken together, these indicates that hands-on, GitHub-based execution (with a clear task path and fast feedback loop) was a primary driver of perceived learning and engagement, and it helps explain why students consistently described OSS-Doorway as both more enjoyable and more transferable to real OSS workflows. This naturally connects to the next theme, where students commented on how the quest structure reduced ambiguity and made the “next step” clear throughout the workflow.

Theme 2: Progressive Quest Structure and Clarity of Next Steps

A recurring theme across the open-ended responses was that OSS-Doorway’s quest-to-task progression made the course feel guided and reduced ambiguity about what to do next. Rather than navigating a set of disconnected instructions, students repeatedly framed the experience as moving through a clear path where each step built toward the next. This was especially valued in a GitHub-centered workflow, where unfamiliar menus, repository

conventions, and multi-step processes can otherwise feel overwhelming or easy to approach in the wrong order.

Students explicitly described the structure as progressive and focused, emphasizing that it kept them on one concept at a time before moving forward. Others described the same idea more bluntly as relief from having to self-navigate course material without a guide.

“The progressive structure guides step by step, focusing strictly on an area before moving on to more content.”

“It felt like it was a very guided exercise, which is good for beginners getting their first introduction ...”

“The structure. I hate having to navigate course material without a guide. ”

A related sub-theme was pacing. Students often tied engagement to quest size and actionability. Shorter, “fast and actionable” quests were described as more motivating, while longer quests risked losing attention. This suggests that the same structural design that improves clarity also shapes perceived workload and momentum.

“I liked the fast and actionable quests. Those made it more engaging and interesting. ”

“The activity as a whole was well structured in a way that actually helped me learn. Without the bot fixing the file issues and learning how actions work would’ve taken me far longer. ...”

At the same time, students also pointed to a practical friction point that is closely tied to progression: recoverability. When a student made a mistake, progression could become harder because they needed to backtrack across earlier steps or reconstruct a working state. One student suggested that each quest step should provide an updated “current version” of required files to make it easier to resume after errors. This is less a critique of the quest concept itself and more a request to make the progressive path easier to re-enter when the

workflow breaks.

“I wish that each quest step had an updated version of all the required files so that way I wouldn’t have to go back through previous steps in my file to get back up to speed if I made a mistake.”

Overall, these responses reinforce that OSS-Doorway’s progressive quest structure functioned as an organizing mechanism that reduced “where do I start?” and “what do I do next?” friction. Students valued the clarity and pacing benefits of a guided path, and the main improvement request within this theme was to make progression more robust to mistakes (for example, by supporting easier recovery and reducing the need to backtrack). This naturally connects to the next theme, where students discuss how immediate validation and iteration interacted with progression, shaping how quickly they could move through tasks and correct errors.

Theme 4: Validator Format Sensitivity and Reliability

Across the open-ended responses, the validator was the most consistent source of friction. Students did not mainly criticize the idea of automated validation. Instead, they described specific failure modes where the validator felt overly format-sensitive, gave feedback that was hard to interpret, and made it difficult to understand what needed to change. In effect, students sometimes experienced the validator less as a learning aid and more as an opaque gatekeeper.

A common complaint was that answers could be rejected even when the student believed they were conceptually correct, because the validator expected a very specific phrasing or structure. Students described the system as struggling to recognize correctness when it was expressed differently than the “preferred” format.

“The validator is just not that good, does not give the best feedback, and seems to struggle in validating things that are actually correct but not in the format it wants.”

This format sensitivity often pushed students into trial-and-error rewriting rather than reflective revision. Instead of using feedback to improve an answer, some students described “playing around” with wording until acceptance, even when the underlying idea had not changed.

“I didn’t really ever get too much feedback, but sometimes when I did it was mildly helpful and sometimes I was a bit confused by what it was trying to have me do, because my answer seemed to already have what it was implying it wanted. Then I would just need to play around with restructuring my answer until it finally accepted it.”

Several responses also highlighted that feedback was not always actionable. Students described moments where they could not pinpoint what was wrong, which led them to “shotgun” responses rather than make targeted improvements.

“The code review was a little bit confusing in the feedback it gave. I couldn’t really pinpoint what I should change in my answers and eventually just shotgunned my answer.”

Relatedly, students reported confusion about where feedback would appear and whether it was even being given in some cases. This came up most strongly in the Code Review quest, where students were unsure how to interpret validator behavior and how to verify whether they were meeting expectations.

“Sometimes the feedback could be difficult to interpret. Also, sometimes the instructions for what to submit/do felt a little bit vague. The most confusion for me came from the Code Review quest, as it was unclear where the feedback would be given, if feedback was even given (I never found any in my submissions), if what was being submitted was correct, etc.”

“Having the bot have better descriptions of things that are wrong.”

Overall, Theme 4 shows a tension between the benefits of automated feedback (Theme 3) and the costs of unreliable or non-actionable validation. Students valued quick iteration, but when the validator’s acceptance criteria were unclear or overly rigid, iteration shifted from learning-oriented revision to guessing what formatting the system wanted. This theme naturally leads into onboarding and submission clarity (Theme 5), because several complaints about the validator were intertwined with unclear instructions about what to submit and where to look for feedback.

Theme 5: Onboarding Clarity and Submission Conventions

While many students described OSS-Doorway as well-structured once they understood the workflow, multiple responses show that the very first interaction still produced avoidable confusion. The issue was not a difficulty with GitHub itself. Rather, students sometimes did not immediately understand the expected “interaction model” of the system: where to begin, what action counts as a submission, and how to complete a task in a way the bot would recognize.

A clear pattern was that some students initially did not realize that their answers were meant to be posted directly in the GitHub issue thread as comments. In these cases, students described spending time trying to figure out how to start, until they discovered the correct action through trial-and-error.

“When i first started, I wasn’t really sure what to do and so I sat there for a while trying to figure iut how to start the quest. Eventually, I realized I just sent my submission as a comment.”

Importantly, students also proposed a simple and low-cost fix: make the submission convention explicit in the very first task. The suggestion was not to redesign the system, but to add one line of instruction that removes ambiguity at the entry point.

“On quest 1, task 1, maybe just add one line saying “Send your answer as a comment here.” And thats about it. It would have stopped my confusion at the very start about where exactly I was suppost to answer.”

Beyond where to submit, students also pointed to the clarity and consistency of task directions, especially early on. Some described the first issue directions as too general, and noted that tasks sometimes felt structured differently from one another, which made it harder to form a stable expectation of “how tasks work” in OSS-Doorway.

“I found some of the beginning issue directions to be confusing and general. Maybe more specific instructions and making sure each task is structured similar. I thought that some of the tasks seemed a bit different than others.”

Relatedly, one student suggested a scaffolding progression that starts with highly guided tasks and gradually reduces guidance. This comment reinforces the idea that early onboarding should explicitly teach how to use the system (submission conventions and patterns), before expecting students to operate more independently.

“It felt like it was a very guided exercise, which is good for beginners getting their first introduction. I think a cool addition would be to do a guided exercise where there are step-by-step instructions, and there is another exercise that has fewer step-by-step instructions, practicing the same Git/GitHub activity but at a higher level of understanding.”

Overall, Theme 5 highlights that onboarding friction is concentrated at the beginning of the experience: students want an explicit “how to interact with OSS-Doorway” cue at the entry point, plus more consistent task formatting. These are small changes, but they matter because early confusion can shape the first impression of the system and can amplify later frustrations (for example, if students cannot tell whether a task was “submitted correctly” or whether the validator is simply rejecting it). Notably, unlike CS386, where students completed early OSS-Doorway assignments in class with a built-in onboarding exercise, CS499/599 did not include the same guided onboarding step, which likely contributed to why this pain point surfaced more clearly here (we return to this in the discussion). This sets up Theme 6, where students discuss how they interpreted gamification elements, and how progress tracking was often more valuable than rewards.

Theme 6: Gamification as Progress Tracking vs. Rewards

Across responses, students consistently separated the value of the quest structure itself from the value of surface reward mechanics such as points, badges, and medals. In many comments, the “gamified” layer was not described as motivating in a competitive or game-like sense. Instead, students framed it as a practical progress-tracking dashboard that helped them understand where they were in the course workflow and what remained.

A recurring idea was that the experience “didn’t feel like a game,” and that the main utility of gamification elements was informational: tracking completion, pace, and status. This framing aligns with how students repeatedly emphasized structure, sequencing, and

clarity of next steps throughout the prior themes.

“It never really felt like a game, I used the statistics more as a dashboard to gauge my own progress. . . . the gamification was not of huge value to me.”

Relatedly, several responses explicitly downplayed or dismissed points and badges. Students often said that the quest split itself was the useful part, while the rewards felt irrelevant or unconnected to anything they cared about.

“I like the split into quests, but that’s about it – medals badges and points meant nothing to me, especially because the AI response bot was inconsistent in point assigning”

This comment also connects gamification perceptions to validator behavior. When students perceived the validator as inconsistent, it did not only create frustration during task completion. It also undermined the meaning of the reward system itself, since points (and therefore badges/streaks) felt unreliable or unfair. In other words, reward mechanics depend on trust in the scoring logic, and inconsistent validation weakened that trust.

At the same time, not all comments rejected rewards entirely. One response suggests that badges/points could become motivating if they carried stronger meaning or social value, implying that the current implementation felt too low-stakes or too detached from any visible outcome.

“I didnt find badges or points very helpful – Maybe make them something more worth showing off or bragging about?”

Another strand of feedback focused less on motivation and more on legibility: students did not always understand what points were for, or how the system was intended to be interpreted. This reflects a design gap where rewards exist, but their purpose is not obvious from the student perspective.

“At this point in the course, I do not really understand the points system, and that part seems a little useless at the moment, unless there is something you could do with the points.”

Overall, Theme 6 suggests that gamification in CS499/599 functioned primarily as progress scaffolding rather than as a reward-driven motivational system. Students valued the quest breakdown and visibility of progress, but frequently treated points and badges as secondary or meaningless, especially when validator behavior made scoring feel inconsistent. This reinforces the broader pattern across themes: structure and clarity drove engagement more reliably than competitive reward mechanics, and the perceived credibility of the validator shaped whether “gamified” metrics felt fair enough to matter.

6.13 Overall Recommendation

To capture an overall, global judgment of the experience, we asked students how likely they would be to recommend OSS-Doorway to a peer on a 0–10 scale (NPS-style). Using standard NPS conventions, ratings of 9–10 are classified as *Promoters*, 7–8 as *Passives*, and 0–6 as *Detractors*. In our sample ($N = 23$), 26.1% were Promoters (6/23), 56.5% were Passives (13/23), and 17.4% were Detractors (4/23), yielding a Net Promoter Score of +8.7 (%Promoters – %Detractors). As a descriptive complement, the raw recommendation ratings had $M = 7.43$ and $SD = 2.19$, indicating that students were generally willing to recommend the tool, but with meaningful variation in sentiment.

Table 6.4: Overall recommendation (NPS; 0–10).

Metric	Value	Notes
Mean recommendation rating	7.43 (SD = 2.19)	$N = 23$
Promoters (9–10)	6/23 (26.1%)	—
Passives (7–8)	13/23 (56.5%)	—
Detractors (0–6)	4/23 (17.4%)	—
Net Promoter Score	+8.7	%Promoters - %Detractors

The NPS breakdown is consistent with the broader quantitative results in Table 6.3. Most students reported strong perceived readiness and confidence with GitHub/OSS workflows (RQ1–RQ2), and many framed the GitHub-based workflow as the most valuable part of the system (qualitative themes in Section 6.12). At the same time, the presence of detractors aligns with persistent friction points that surfaced repeatedly in open-ended feedback, particularly cases where validator behavior felt overly format-sensitive or where early interaction norms (e.g., how to submit) were not immediately obvious to every student.

6.14 Discussion

This deployment represents our second integration of OSS-Doorway, and the results provide a clearer view of what the system reliably delivers (and at what cost). Overall, the findings reinforce that embedding coursework inside GitHub can make OSS workflows feel more authentic and transferable, while also showing that the quality and interpretability of automated validation remains the most consequential bottleneck for learner experience and perceived fairness. Below, we summarize what worked well, where friction persisted, and how the CS386 deployment shaped the changes implemented in CS499/599.

6.15 What Worked Well in CS499/599

Students left with high perceived readiness for OSS contribution. Across the OSS readiness items (RQ1), respondents reported consistently strong agreement, with an overall composite mean of 5.78/7 ($SD = 0.79$); Figure 6.1. The strongest signal was students’ understanding of the typical OSS workflow (22 positive, out of 23 in the top-two agreement bins; Figure 6.2), suggesting that the quest-to-task structure succeeded at making the end-to-end flow concrete (e.g., forking, branching, pull requests).

Readiness also extended beyond mechanics. Items about social norms and explaining first contributions were high ($M = 5.91$, $SD = 1.08$; and $M = 5.78$, $SD = 0.90$), implying that students not only learn steps, but also feel more capable of participating in an OSS community context.

GitHub self-efficacy was consistently high and aligned with the course tasks. Students reported strong confidence in performing GitHub-specific tasks (RQ2), mirroring the hands-on structure of the course quests (e.g., issue tracking, pull requests, code review, GitHub Actions, and project setup). This alignment matters because it suggests the intervention was not experienced as “extra tooling,” but as a direct enactment of the target workflow.

In the qualitative feedback, students frequently described the experience as “more real” than conventional LMS assignments and highlighted that completing tasks inside GitHub increased transferability to future work. This framing reinforces the idea that self-efficacy gains were tied to repeated practice in authentic GitHub interaction patterns rather than isolated skill drills.

Why the GitHub-Native Format Worked (and Why It Felt More Authentic)

A consistent reason the intervention worked well is that OSS-Doorway did not feel like an additional educational layer on top of the course. Instead, it positioned GitHub itself as the learning environment, so the tasks, submissions, and feedback loop happened in the same

artifact space students would encounter in real OSS work (issues, pull requests, commits, and CI workflows). This reduced context switching and helped students treat coursework as a sequence of concrete contribution actions rather than abstract homework steps. In practice, the quest structure operationalized the workflow into checkable milestones, which likely explains why students rated both OSS readiness and task scaffolding strongly: the system made the intended contribution process visible, repeatable, and harder to misinterpret.

Beyond learning the mechanics, the GitHub-based format also supported a stronger sense of transfer. Because students practiced the same interaction patterns used in real projects (creating and navigating issues, opening pull requests, responding to review feedback, and observing automated checks), the work was easier to interpret as *skill-building* rather than merely assignment completion. This aligns with qualitative feedback where students described the experience as more practical than LMS-based assignments, and with the quantitative pattern where readiness extended beyond workflow steps to higher-level confidence about participating in OSS communities (e.g., social norms and explaining a first contribution).

Onboarding Worked Well Overall, But Early Orientation Still Mattered

Although the CS499/599 deployment produced strong ratings and many positive reflections, the qualitative themes show that a portion of friction was concentrated at the very start of the experience. Students primarily wanted a clearer “how to interact with OSS-Doorway” cue at the entry point (e.g., where exactly to submit, what a correct submission looks like, and how to interpret the bot’s responses). This matters even when the rest of the system is effective, because early ambiguity can shape the first impression of the tool and can amplify later frustrations (for example, if students cannot tell whether a task was “submitted correctly” or whether the validator is simply rejecting it).

Importantly, this onboarding friction was more visible in CS499/599 because the deployment did not include a comparable guided onboarding exercise at the beginning. In CS386, students completed early OSS-Doorway assignments in class, which implicitly supported ori-

entation to the interaction model and allowed immediate clarification when students were unsure how to begin. In CS499/599, the system was introduced in a more self-directed way, and students interacted with the tool asynchronously, which made the initial entry point and instructions do more work. As a result, even small missing cues (e.g., explicitly stating “submit your answer as an issue comment”) become more consequential. We revisit this contrast in more detail in the discussion when we connect Theme 5 (onboarding clarity) to the broader design lessons from CS386 and the remaining friction points in CS499/599.

Task scaffolding and tool experience were rated positively. Students rated task-level scaffolding and overall tool experience positively (RQ3), with a composite mean of 5.79/7 (SD = 1.31; Table 6.3). Items about tasks improving understanding of class requirements and providing more actionable guidance than standard assignments were among the highest-rated, reinforcing the interpretation that OSS-Doorway is functioning primarily as a *structure and pacing mechanism* rather than just a content delivery layer.

In open-ended feedback, this showed up as students emphasizing clarity of next steps, reduced ambiguity, and the value of having everything in one place. Taken together, these responses suggest that students experienced the quests as a practical guide for translating course expectations into concrete actions, which aligns with the goal of reducing “what do I do next?” friction during GitHub-based work.

Immediate feedback supported iteration and flexible scheduling. Across qualitative responses, immediate feedback was consistently framed as a practical advantage. Students reported that the system enabled them to work on tasks at any time, iterate without waiting for grading cycles, and validate progress incrementally, which made it easier to maintain momentum when they got stuck.

This complements the quantitative validator findings (RQ4), which show that students generally agreed that feedback influenced revisions and helped identify mistakes (composite mean = 4.17/5, SD = 0.77; Table 6.3). Even when feedback was imperfect, students often valued the presence of a rapid feedback loop as a learning affordance because it supported

trial, correction, and resubmission in a way that is hard to replicate with delayed, manual grading.

6.16 Where Friction Persisted and Why

Despite the strong overall results, the same categories of friction that emerged in CS386 were still present, and they remain central to the interpretation of our findings.

Validator format sensitivity reduced perceived fairness. The clearest recurring frustration concerned situations where correct or reasonable answers were rejected due to formatting, phrasing, or the validator’s interpretation of open-ended responses. Students described cases where the system appeared to prefer a specific surface form (e.g., a bare number versus a sentence containing that number), which made the validator feel less like a supportive tutor and more like an opaque gatekeeper. Importantly, this type of friction has a disproportionate impact because it directly affects progression, streaks/points, and the sense of whether effort is being assessed fairly. In other words, validator imperfections do not merely reduce accuracy; they shape trust.

False acceptance created a different kind of risk. A second, distinct failure mode also surfaced: students noted that the validator could sometimes be “too easy” to satisfy, approving responses that did not reflect genuine completion or understanding. While this is less frustrating in the moment, it undermines the pedagogical role of the validator by weakening the link between completion and learning. Together, false rejection and false acceptance point to the same design tension: LLM-based judgment is flexible but can be inconsistent, and when it is used as a gating mechanism, that inconsistency becomes visible and consequential.

Early interaction norms were not universally obvious. Even in CS499/599, some students reported initial uncertainty about how to start tasks and where to submit responses (especially when the interaction model differs from typical LMS norms), and this friction

was less visible in the CS386 deployment because students completed early OSS-Doorway assignments in-class with live guidance that implicitly established the “submit as an issue comment” norm from the start, whereas CS499/599 relied more on students self-starting outside class.

Gamification helped as a structure, not as a reward system. Across both quantitative and qualitative signals, gamification was not primarily experienced as competition. Students valued the quest organization and visibility of progress, but many treated points, badges, and streaks as secondary (or irrelevant), especially when validator inconsistency affected reward assignment. This suggests that the *structural* components of gamification (progress visibility, chunking, milestones) drive most of the perceived benefit, while the reward layer can become counterproductive if it amplifies the impact of validator errors.

6.17 How CS386 Findings Informed This Deployment

A central arc of this thesis is that the CS386 deployment functioned as a formative, diagnostic study that shaped how OSS-Doorway was prepared for CS499/599. The CS386 results highlighted three practical constraints that directly informed changes in the second deployment:

(1) Make interaction norms explicit. CS386 surfaced that students could lose time early simply figuring out where and how to respond. In CS499/599, we treated this as an onboarding design problem rather than a student misunderstanding, and we emphasized clearer submission cues and task instructions so that the first successful interaction happens quickly.

(2) Reduce avoidable format sensitivity. CS386 feedback made it clear that format sensitivity was not a minor annoyance, but a core usability barrier. For CS499/599, we adjusted validator policies and prompting to better tolerate reasonable paraphrases and minor formatting differences for open-ended tasks, while still requiring task-relevant evidence

for completion.

(3) Treat validator messaging as part of scaffolding. In CS386, students reported that rejection messages were sometimes difficult to act on. For CS499/599, we emphasized more actionable validator responses (what is missing, what to change, and what counts as acceptable evidence), aligning validator behavior with the instructional scaffolding goal rather than only correctness checking.

Although these changes improved the overall experience, the CS499/599 results show that validator interpretability and consistency still dominate perceived friction. This reinforces the conclusion that the validator is not a peripheral feature; it is the primary determinant of whether the tool feels fair and helpful at scale.

6.18 Implications for GitHub-Centered OSS Education

The CS499/599 deployment provides several implications for instructors and researchers designing GitHub-centered OSS education. While our results are grounded in a single course context, the patterns across both the quantitative ratings and open-ended feedback point to design principles that are likely to transfer to other GitHub-based course interventions.

GitHub-based delivery can improve authenticity and transfer. Students repeatedly framed OSS-Doorway as “more real” than LMS-based assignments because the work unfolded inside the same artifacts that define OSS contribution: issues, pull requests, code reviews, and CI workflows. This matters for education because it reduces the gap between “learning about” contribution and actually practicing it. Instead of treating GitHub as a submission portal, the intervention turned it into the learning environment itself. In doing so, it supports workflow socialization: students practice not only the mechanics (forking, branching, PRs, reviews, Actions), but also the interaction norms that accompany them (where work happens, how progress is tracked, what evidence looks like, and what counts as completion). A practical implication is that instructors who want to transfer should con-

sider shifting from GitHub-as-attachment to GitHub-as-interaction surface, where the course intentionally uses the same objects students will encounter in real projects.

This also suggests an advantage for assessment and instructional alignment. When tasks live inside issues and pull requests, the instructor can author prompts that explicitly connect lecture concepts to concrete contribution steps, and students can build a traceable activity record that mirrors professional practice. Over time, this kind of “artifact trail” can make learning outcomes easier to demonstrate (and easier for students to describe later), because the outputs look like real contributions rather than isolated homework files.

Structure and pacing matter more than rewards. Across themes, students consistently separated the value of the quest structure from the value of points and badges. The quest-to-task progression was treated as the main source of engagement because it reduced ambiguity, narrowed attention to one actionable step at a time, and made the next step visible. In other words, the motivational benefit looked less like “competition” and more like *instructional sequencing*: making an unfamiliar workflow feel navigable through clear ordering and milestone visibility. This is an important implication for how we interpret “gamification” in OSS education. If the design goal is onboarding and persistence, then progression mechanics (quests, gating, clear milestones, lightweight progress dashboards) may be the core contribution, while reward layers (points, streaks, badges) should be treated as optional and secondary.

A direct design consequence is that pacing should be treated as part of the curriculum, not a cosmetic interface choice. The timing and size of quests influence students’ ability to sustain momentum, recover from mistakes, and avoid overload. When the system makes the “next step” obvious, students can focus cognitive effort on the task itself rather than on figuring out what the course expects. For instructors, this suggests investing effort in (i) task granularity (small enough to complete with quick feedback loops), (ii) dependency clarity (why a task unlocks the next), and (iii) recovery pathways (how to get unstuck without redoing large portions of work).

Automated validation must prioritize transparency, actionability, and fairness. CS499/599 reinforced a core lesson about automated validation in authentic workflows: if validation gates progress, students will treat it as a judge. When the validator is strict about hidden formatting expectations or when rejection messages do not clearly explain what is missing, students cannot distinguish between a conceptual mistake, a formatting issue, or an evidence mismatch. That uncertainty is what turns iteration into frustration. In educational tooling, correctness is not the only requirement; the validator must be *instructionally legible*. Students should be able to answer: “What was checked?” “What failed?”, and “What do I do next?” without trial-and-error guessing.

This implies a hybrid approach. Deterministic checks should be used wherever feasible (file existence, commit evidence, CI status, specific repository metadata, link presence, test outputs) because they are reproducible and easier to explain. LLM-based validation can add flexibility for open-ended responses, but it should be constrained by explicit rubrics and paired with messaging that lists the acceptance criteria and highlights missing elements. If uncertainty remains, the system should shift to “needs human review” rather than confidently rejecting or accepting. From a fairness perspective, transparency is not optional: if students perceive the gate as arbitrary, it undermines trust not only in the validator, but also in associated mechanics like points, streaks, and progress tracking.

Onboarding should be treated as a first-class design target, not an afterthought. The beginning phase of any learning is the most important, with that, even small confusions at the entry point can shape overall perceptions because the first interaction sets the mental model for how the system works. In CS499/599, the interaction model relied more heavily on students self-starting outside class, which makes implicit norms (e.g., “submit as an issue comment”) easier to miss than in in-class guided settings. This implies that GitHub-centered courseware needs a lightweight “orientation layer” that explicitly teaches the workflow of the tool before students begin content tasks. A short onboarding exercise, a single line on the first task (“Submit your answer as a comment here”), and one or two examples of acceptable

responses can prevent avoidable early friction and reduce downstream misattributions (e.g., students thinking the tool is broken when they simply submitted in the wrong place).

More broadly, onboarding is not only about instructions. It also includes expectation-setting around iteration: how many attempts are typical, what kinds of feedback the bot can and cannot provide, and what to do when feedback is unclear. In GitHub-based learning environments, students are learning two things at once: the course content and the platform interaction norms. Making that dual learning goal explicit can reduce anxiety and increase persistence, especially for students with less prior GitHub exposure.

Taken together, these implications suggest that the strongest educational value of GitHub-centered interventions like OSS-Doorway comes from *guided participation in authentic workflows* supported by a clear progression structure and a feedback loop that is transparent enough to enable learning-oriented iteration.

6.19 Limitations

Several limitations constrain how these results should be interpreted.

Our evaluation is based primarily on a post-course, self-reported survey. This provides direct insight into students' perceived learning, confidence, and experience, but it does not measure objective skill gains, behavioral mastery, or long-term retention. As a result, the findings should be read as evidence about perceived readiness and usability, rather than definitive evidence of performance improvement.

The study reflects a single course offering in a single institutional context, with a specific instructor, curriculum structure, and cohort. This setting improves ecological realism, but it also limits generalizability. Student backgrounds, course incentives, and available support vary widely across programs, so the same intervention may produce different outcomes in other courses or institutions.

Because OSS-Doorway was integrated into a graded course, student engagement and be-

havior were shaped by deadlines, workload constraints, and grade incentives, not only by tool design. In addition, we did not include a controlled comparison group (for example, students completing the same learning objectives without OSS-Doorway or using an alternative workflow such as GitHub Classroom or a traditional LMS). Without a control condition and random assignment, we cannot make causal claims that OSS-Doorway *caused* the reported outcomes.

In practice, students differed in prior GitHub experience, effort, time-on-task, and help-seeking behavior, and these sources of variation cannot be fully separated from the effects of the intervention. Relatedly, OSS-Doorway’s effectiveness depends partly on how tasks are authored and configured (prompt clarity, acceptance criteria, validator strictness, and feedback messaging). Differences in task design can meaningfully shape the student experience, so our results reflect both the tool and how it was instantiated in CS499/599.

Finally, the validator includes LLM-based judgment for open-ended tasks. While this enables flexible feedback, it introduces non-determinism and occasional format sensitivity that are difficult to control fully, and this variability likely influenced both quantitative ratings and qualitative feedback. We also did not control or systematically measure students’ use of external support (e.g., general-purpose LLMs, peers, or online resources), which may have affected perceived difficulty and confidence in ways that cannot be disentangled from OSS-Doorway’s contribution.

6.20 Summary

This chapter reports findings from the CS499/599 deployment of OSS-Doorway, our second integration of the system in a university course context. Students reported strong perceived OSS readiness (RQ1) and high GitHub self-efficacy (RQ2), and they rated task scaffolding and overall tool experience positively (RQ3). Open-ended responses reinforced that the GitHub-based, hands-on workflow and immediate feedback loop were the most valued aspects

of the tool. At the same time, persistent friction centered on validator format sensitivity, reliability, and the clarity of rejection messages, showing that automated validation remains the primary improvement opportunity. Finally, the results support the thesis arc that CS386 served as a formative deployment: it revealed specific usability and validator shortcomings that informed changes for CS499/599, while also demonstrating which challenges persist even after iteration.

CHAPTER 7 Conclusion

In this thesis, I studied how a GitHub-based quest system (OSS-Doorway) can support students as they learn OSS contribution workflows in real course settings, and how students perceive the system’s scaffolding, automated feedback, overall usefulness, and confidence in use afterward. I conducted two classroom deployments across two semesters: an initial formative deployment in CS386 (Fall 2025) and a follow-up evaluation in CS499/599 (Spring 2026). Together, these studies explain not only whether students found OSS-Doorway helpful, but also where friction emerged, how students interpreted automated validation, and how lessons from the first deployment informed improvements for the second.

Study 1 (CS386) served as the first end-to-end deployment of OSS-Doorway in a classroom workflow. It emphasized formative evaluation: understanding whether students could follow the interaction model (finding tasks, submitting work on GitHub, interpreting feedback, and progressing through quests) and identifying concrete points of confusion or breakdown. The results from this deployment highlighted the importance of clarity in submission conventions and the need for validator feedback that is interpretable and actionable. In particular, student feedback suggested that when the system’s expectations are not explicit, students spend time figuring out where to respond and what counts as evidence, and when automated rejection messages are vague or format-sensitive, students experience unnecessary friction even when they understand the underlying concept.

Study 2 (CS499/599) evaluated an improved version of OSS-Doorway informed by Study 1. This deployment focused on students’ perceived OSS readiness, GitHub self-efficacy, task scaffolding and tool experience, validator feedback quality, and perceptions of the quest structure and gamification layer. Overall, students reported positive perceptions of readiness

and confidence with core GitHub workflows, and they described the quest-to-task structure as useful for making expectations concrete and supporting learning-by-doing. At the same time, the most consistent remaining area for improvement was the validator: students valued immediate feedback, but they still reported cases where feedback was not sufficiently diagnostic, where validation was sensitive to formatting or phrasing, or where acceptance did not always align with the evidence they believed they had provided. The qualitative feedback reinforced this interpretation by consistently distinguishing the value of progressive structure and hands-on GitHub practice from the frustrations tied to validation reliability and clarity.

Across both deployments, a clear pattern emerged. OSS-Doorway’s main strength is that it embeds structured learning into authentic GitHub workflows, reducing context switching and helping students practice contribution-relevant actions in a realistic environment. Students consistently described the quest structure as improving clarity about what to do next and making the learning experience more engaging than conventional assignments delivered through an external LMS. However, the system’s effectiveness depends strongly on whether automated validation behaves like a helpful feedback assistant rather than an opaque judge. When validation logic is legible, and feedback explains what is missing and how to fix it, students can iterate productively. When it is not, the tool risks shifting effort away from learning the workflow and toward guessing hidden acceptance criteria.

This thesis contributes (i) an empirical account of two classroom deployments of a GitHub-based quest system for teaching OSS contribution workflows, (ii) evidence that structured progression and hands-on GitHub integration can support students’ perceived readiness and confidence, and (iii) a set of design implications showing that validator transparency and diagnostic feedback are the highest-leverage improvements for future iterations.

By iterating from CS386 to CS499/599, this work also demonstrates a practical path for refining educational bots through deployment-driven evidence, where early classroom findings guide system changes and later deployments evaluate whether those changes improve

student experience.

RQ1. Across both deployments, students generally perceived OSS-Doorway’s quest-based scaffolding as a useful way to learn GitHub-centered software engineering and OSS workflows. The clearest strength of the system was its progressive structure: quests and tasks made the next step more visible, reduced ambiguity, and supported learning-by-doing inside authentic GitHub artifacts rather than outside the workflow in a separate LMS.

RQ2. Students perceived OSS-Doorway’s AI-assisted feedback as valuable when it was immediate, interpretable, and specific enough to support revision. At the same time, both studies showed that the validator remained the system’s main point of friction whenever feedback became vague, overly format-sensitive, or difficult to connect to a concrete next action. In short, the feedback loop was helpful, but its educational value depended strongly on clarity and legibility.

RQ3. The two-deployment refinement process showed that the most important design lessons were not about adding more gamified features, but about improving onboarding clarity, progression robustness, validator transparency, and reliability. What worked best was the combination of structured GitHub-based practice and visible progression; what needed the most attention was making automated validation behave more like a learning aid than an opaque gatekeeper.

7.1 Implications for OSS and Software Engineering Education

For instructors who want students to learn contribution workflows in authentic tooling, these findings suggest that GitHub-based delivery can be an effective alternative to traditional LMS-centered assignments, but only when onboarding cues and submission conventions are made explicit at every entry point. In addition, structured progression and task chunking appear to be the primary mechanisms driving engagement and clarity, while surface-level

reward elements such as points and badges should remain secondary.

If automated feedback is used at scale, its criteria should be transparent to students, and its rejection messages should consistently specify what evidence was missing and what actionable next step the student should take. More broadly, the findings suggest that the value of GitHub-based educational systems depends less on reward mechanics alone and more on whether they provide clear progression, visible expectations, and feedback that students can use productively.

7.2 Implications for Automated Validation and Feedback Design

Across both deployments, the validator emerged as the most important improvement target. Future systems should prioritize criterion transparency (what is being checked and why), diagnostic specificity (what is missing or incorrect), and robustness to common formatting or phrasing variation. Where possible, acceptance should be anchored in deterministic, machine-checkable evidence (e.g., repository state, issue metadata, PR status, or required artifacts) rather than purely textual judgment. When LLM-based assessment is necessary, it should be constrained by explicit rubrics and should clearly communicate uncertainty, prompting human review when confidence is low.

Building on this thesis, future work should evaluate OSS-Doorway across additional courses and institutions to strengthen external validity and understand how the system performs for students with less GitHub exposure. In addition, future deployments should incorporate stronger behavioral measures beyond self-report, such as linking quest completion patterns to later contribution behavior or assessing workflow proficiency through pre/post performance tasks. Finally, improving the validator remains central: refining prompts and rubrics using real student responses, expanding deterministic checks for artifact-based tasks, and improving feedback templates are likely to provide the largest gains in perceived fairness,

usability, and learning-oriented iteration.

7.2.1 Replication package

Anonymized survey extracts, analysis scripts, and generated tables and figures for this thesis are archived in a self-contained replication package (directory layout summarized in its README.md). Archived release: `doi:10.5281/zenodo.20093389`

Bibliography

- [1] Kirsti M Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005. arXiv:<https://doi.org/10.1080/08993400500150747>, doi:10.1080/08993400500150747.
- [2] Sogol Balali, Igor Steinmacher, Uma Annamalai, Anita Sarma, and Marco Aurelio Gerosa. Newcomers’ barriers. . . is that all? an analysis of mentors’ and newcomers’ barriers in oss projects. *Computer Supported Cooperative Work (CSCW)*, 27, 12 2018. doi:10.1007/s10606-018-9310-8.
- [3] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: defining ”gamification”. In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek ’11, page 9–15, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/2181037.2181040.
- [4] Darina Dicheva, Christo Dichev, Gennady Agre, and Galia Angelova. Gamification in education: A systematic mapping study. *Journal of Educational Technology & Society*, 18(3):75–88, 2015. URL: <http://www.jstor.org/stable/jeductechsoci.18.3.75>.
- [5] Misan Paul Etchie, Hunter Beach, Katia Felizardo, and Igor Steinmacher. Towards a dataset to assess conversational agent’s efficacy in mentoring newcomer developers. In *Proceedings - 2025 IEEE/ACM International Workshop on Bots in Software Engineering, BotSE 2025*. IEEE/ACM, 2025. Also available via NAU Experts page <https://experts.nau.edu/en/publications/towards-a-dataset-to-assess-conversational-agents-efficacy-in-men/>.

URL: https://www.researchgate.net/publication/393081917_Towards_a_Dataset_to_Assess_Conversational_Agent's_Efficacy_in_Mentoring_Newcomer_Developers.

- [6] Joseph Feliciano, Margaret-Anne Storey, and Alexey Zagalsky. Student experiences using github in software engineering courses: a case study. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, page 422–431, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2889160.2889195.
- [7] John Hattie and Helen Timperley. The power of feedback. *Review of Educational Research*, 77(1):81–112, 2007. arXiv:<https://doi.org/10.3102/003465430298487>, doi:10.3102/003465430298487.
- [8] Hao He, Haonan Su, Wenxin Xiao, Runzhi He, and Minghui Zhou. Gfi-bot: automated good first issue recommendation on github. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 1751–1755, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3540250.3558922.
- [9] Courtney Hsing and Vanessa Gennarelli. Using github in the classroom predicts student learning outcomes and classroom experiences: Findings from a survey of students and teachers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 672–678, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3287324.3287460.
- [10] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. page 86–93, 2010. doi:10.1145/1930464.1930480.

- [11] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. A systematic literature review of automated feedback generation for programming exercises. *ACM Trans. Comput. Educ.*, 19(1), September 2018. doi:10.1145/3231711.
- [12] Ramon Lawrence, Sarah Foss, and Tatiana Urazova. Evaluation of submission limits and regression penalties to improve student behavior with automatic assessment systems. *ACM Trans. Comput. Educ.*, 23(3), June 2023. doi:10.1145/3591210.
- [13] Juho Leinonen, Paul Denny, and Jacqueline Whalley. A comparison of immediate and scheduled feedback in introductory programming projects. page 885–891, 2022. doi:10.1145/3478431.3499372.
- [14] Maria Augusta Nelson and Lesandro Ponciano. Experiences and insights from using github classroom to support project-based courses. In *2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG)*, page 31–35. IEEE, May 2021. URL: <http://dx.doi.org/10.1109/SEENG53126.2021.00013>, doi:10.1109/seeng53126.2021.00013.
- [15] Maciej Pankiewicz and Ryan S. Baker. Large language models (gpt) for automating feedback on programming assignments. 2023. URL: <https://arxiv.org/abs/2307.00150>, arXiv:2307.00150.
- [16] Swapneel Sheth, Jonathan Bell, and Gail Kaiser. Increasing student engagement in software engineering with gamification. Technical report. doi:10.7916/D8H99DF0.
- [17] Dan Sholler, Igor Steinmacher, Denae Ford, Mara Averick, Mike Hoye, and Greg Wilson. Ten simple rules for helping newcomers become contributors to open projects. *PLoS Computational Biology*, 15(9), 2019. doi:10.1371/journal.pcbi.1007296.
- [18] Valerie J. Shute. Focus on formative feedback. *Review of Educational Research*, 78(1):153–189, 2008. arXiv:<https://doi.org/10.3102/0034654307313795>, doi:10.3102/0034654307313795.

- [19] Priscylla Silva and Evandro Costa. Assessing large language models for automated feedback generation in learning programming problem solving. 2025. URL: <https://arxiv.org/abs/2503.14630>, arXiv:2503.14630.
- [20] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. page 1379–1392, 2015. doi:10.1145/2675133.2675215.
- [21] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. Overcoming open source project entry barriers with a portal for newcomers. page 273–284, 2016. doi:10.1145/2884781.2884806.
- [22] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. volume 59, page 67–85, USA, March 2015. Butterworth-Heinemann. doi:10.1016/j.infsof.2014.11.001.
- [23] Philipp Straubinger and Gordon Fraser. Improving testing behavior by gamifying IntelliJ, 2023. URL: <https://arxiv.org/abs/2310.11171>, arXiv:2310.11171.
- [24] Sujit Subhash and Elizabeth A. Cudney. Gamified learning in higher education: A systematic review of the literature. *Comput. Hum. Behav.*, 87(C):192–206, 2018. doi:10.1016/j.chb.2018.05.028.
- [25] Armando Toda, Ana Klock, Wilk Oliveira, Paula Toledo Palomino, Luiz Rodrigues, Lei Shi, Ig Bittencourt, Isabela Gasparini, Seiji Isotani, and Alexandra Cristea. Analysing gamification elements in educational environments using an existing gamification taxonomy. *Smart Learning Environments*, 6, 12 2019. doi:10.1186/s40561-019-0106-1.
- [26] Michel Wermelinger. Using github copilot to solve simple programming problems, 2023. doi:10.1145/3545945.3569830.

- [27] Wenxin Xiao, Hao He, Weiwei Xu, Xin Tan, Jinhao Dong, and Minghui Zhou. Recommending good first issues in github oss projects. In *Proceedings of the 44th International Conference on Software Engineering, ICSE '22*, page 1830–1842, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3510003.3510196.

APPENDIX A CS386 Qualtrics instrument

The following appendix reproduces the CS386 post-course instrument content and block flow as administered in Qualtrics.

Start of Block 1 Q1. You are invited to take part in a research study evaluating OSS-Doorway, the GitHub-integrated learning platform used in this course. Your participation is voluntary and your responses are anonymous. This survey takes 10–12 minutes.

End of Block 1

Start of Block 2 Q2. How would you describe your current software engineering (SWE) skills?
Response options: Beginner/Novice; Intermediate; Advanced; Expert.

End of Block 2

Start of Block 13 (projects inventory) Q3. Approximately how many software projects have you completed in each category throughout your career?

Scale: 0, 1, 2–3, 4+.

Categories: Course projects; Personal projects; Industry/internship projects.

End of Block 13 (projects inventory)

Start of Block 3 Q4. Git/GitHub experience prior to this class.

Response options: None; Basic; Intermediate; Advanced.

End of Block 3

Start of Block 4 Q5. How useful do you think gamification elements (badges, ranks, quests, tasks, and points) are in your coursework?

Scale: Not at all useful to Extremely useful.

End of Block 4

Start of Block 5 Q6. Tasks refer to the individual, structured activities within each biweekly quest.

Scale: Strongly disagree to Strongly agree.

- Tasks improved my understanding of class requirements and learning objectives.
- Tasks reinforced key concepts covered in lectures.
- The system improved my ability to use GitHub effectively.
- Tasks provided timely feedback that deepened my understanding of core course topics.

Q7 Comparison to Traditional Assignments.

Scale: Strongly disagree to Strongly agree.

- Compared to standard assignments, tasks gave me more actionable guidance.
- Compared to standard assignments, tasks provided a more interactive and engaging experience.
- Compared to standard assignments, the task system helped me familiarize myself with GitHub.

Q8 Workload & Clarity.

Scale: Strongly disagree to Strongly agree.

- Tasks were easy to locate from the GitHub repository.
- I understood how to call the professor and teaching assistants for assistance in GitHub issues when I needed help.

Q22. Any comments?

End of Block 5

Start of Block 8 Q9 Experience with LLMs.

Scale: Never, Rarely, Sometimes, Often, Very Often.

Prompt: How often did you rely on tools like ChatGPT, Copilot, or Gemini while completing tasks?

Q10 LLM Use.

Scale: Not at all influential to Extremely influential.

Prompt: To what degree did OSS-Doorway influence your decision to use these AI tools?

End of Block 8

Start of Block 9 Q11. The built-in answer validator is the automated checker inside OSS-Doorway that reviews your work (such as code, issue responses, or diagrams) and gives immediate feedback on whether you completed the quest correctly. It also provides helpful feedback on next steps whenever you do not quite get the answer right.

Scale: Strongly disagree to Strongly agree.

- The validator's feedback influenced how I adjusted my answers.

- Suggestions for next steps were clear and actionable.
- The validator helped me identify specific mistakes.
- Feedback on diagrams, screenshots, or artifacts influenced my progress.

Q23. Any comment?

End of Block 9

Start of Block 10 Q12 System Mechanics.

Scale: Strongly disagree to Strongly agree.

- Having multiple tasks open at once influenced my pacing.
- The README structure influenced how easy it was to find the next task.

Q13. What made it hard to figure out what to do next (if anything)?

End of Block 10

Start of Block 11 Q14. Describe a moment when a quest clarified a concept better than lectures or homework.

Q15. Describe a task where the validation feedback felt wrong or unclear. What would have fixed it?

Q16. How would your learning have been different without the bot/quests?

End of Block 11

Start of Block 12 Q17. Suggest specific technical improvements to OSS-Doorway that would have improved your learning outcomes and ease of use with the system.

End of Block 12

Start of Block 13 (closing) Q18. Thank you for participating! Your feedback directly helps improve OSS-Doorway and future course design.

APPENDIX B CS499/599 Qualtrics instrument

The following appendix reproduces the CS499/599 post-course instrument content and block flow as administered in Qualtrics.

Start of Block: Informed consent Q30 Consent to Participate in Research.

Study Title: HSI Implementation and Evaluation Project: OSS-Doorway: A Learning Environment for Open Source Contribution Model (NSF 2247929).

Summary of the research. You are being asked to participate in a research study. Your participation in this research study is voluntary and you do not have to participate. This document contains important information about this study and what to expect if you decide to participate. Please consider the information carefully. Feel free to ask questions before making your decision whether or not to participate.

Purpose: The purpose of the study is to understand and improve OSSDoorway, a gamified learning environment.

What you will do in the study: We are requesting your permission to use your assignment responses from the open-source development class as part of a research project aimed at improving OSSDoorway. The data we collect will be anonymized by a third party in compliance with FERPA guidelines, ensuring your confidentiality and data security. We have coordinated with the NAU Registrar's Office to reinforce adherence to privacy standards, and an amendment has been made to document this process. Our study will review these documents—specifically, anonymous reports—to gain insights into your reflections and experiences related to the project. This branch of the project is an addition to previously approved studies, with minimal to no risk posed to individuals, even in the event of a malfunction. By consenting, you allow us to analyze your anonymous report solely for research purposes. All reports will remain anonymous, with no identifying information linked to any individual student in our analysis. The collected data will be securely stored, protected by NAU credentials, and accessed only by authorized researchers in line with the original approval package.

Risks or discomforts: There is a risk that your identifiable information could be accidentally disclosed; however, the researchers are taking measures to protect your data.

Benefits of this study: The benefit to the researcher is understanding how to make GitHub available to newcomers (e.g., computer science students), a better understanding of the technological and accessibility issues faced by our stakeholders, and to learn how to make future contributions in OSS projects better in general. There will be no direct benefit for you to participate.

Confidentiality: Your name will not be used in any report. Identifiable research data will be encrypted and password protected. While researchers follow procedures to maintain your confidentiality, as with any internet activity, we cannot guarantee confidentiality of interception of data sent via the Internet by any third parties. The information that you provide in the study will be handled confidentially. However, there may be circumstances where this information must be released or shared as required by law. Northern Arizona University Institutional Review Board

may review the research records for monitoring purposes.

Future Research: Information that may identify you may be used for future research or shared with another researcher for future research studies without additional consent.

Who to Contact: For questions, concerns, or complaints about the study you may contact Dr. Igor Steinmacher by emailing him at igor.steinmacher@nau.edu. For questions about your rights as a participant in this study or to discuss other study-related concerns or complaints with someone who is not part of the research team, you may contact the Human Research Protection Program at 928-523-9551 or online at <http://nau.edu/Research/Compliance/Human-Research/Welcome/>.

AGREEMENT TO PARTICIPATE. I have read (or someone has read to me) this form, and I am aware that I am being asked to participate in a research study. I have had the opportunity to ask questions and have had them answered to my satisfaction. I affirm that I am at least 18 years of age and voluntarily agree to participate in this study.

Response options: I agree to participate; I do not agree to participate.

Q42. Your name.

End of Block: Informed consent

Start of Block: Introduction **Q1.** This survey is about the bot used in the class (OSS-Doorway), which was integrated into the GitHub environment.

Q29. There were 6 quests in the system (Q1 Dealing with Git; Q2 Understanding OSS Projects and GitHub Basics; Q3 Using GitHub; Q4 GitHub Repo; Q5 Pull Request and Code Review; Q6 GitHub Actions; Q7 Set Up Your Open Source Project). *How many quests did you complete?*

Response options: I have never used this system; None; 1–4; 5–6; 7 (all of them).

End of Block: Introduction

Start of Block: Self-Determination Theory (SDT) **Q44.** For each statement, indicate how true it is for you based on your experience with OSS-Doorway. **Scale:** 1 (Not at all true) to 7 (Very true).

- I enjoyed completing quests in OSS-Doorway very much.
- I think that doing the OSS-Doorway quests is useful for learning how to contribute to open source projects.

End of Block: Self-Determination Theory (SDT)

Start of Block: Self-Efficacy / Legitimate Peripheral Participation (LPP) / OSS Readiness **Q36.** After using OSS-Doorway, I am confident that I can... **Scale:** 1 (no confidence) to 5 (High confidence).

- use GitHub issue tracker to find open issues;
- understand GitHub pull-requests;
- open a pull request using GitHub web interface;
- use GitHub to contribute to projects.

Q45. Please indicate how much you agree or disagree with the following statements. **Scale:** Strongly disagree to Strongly agree.

- I understand the typical workflow for contributing to an open source project (e.g., forking, branching, submitting pull requests).
- I feel prepared to make a meaningful contribution to a real open source project.
- I feel less intimidated about contributing to open source projects than before this course.
- Click to write Statement 4.

End of Block: Self-Efficacy / Legitimate Peripheral Participation (LPP) / OSS Readiness

Start of Block: Intention to Contribute to OSS **Q46.** Please indicate how much you agree or disagree with the following statements. **Scale:** Strongly disagree to Strongly agree.

- I am more likely to contribute to open source projects now than I was before taking this course.
- After this experience, I intend to contribute to an open source project outside of this class.

End of Block: Intention to Contribute to OSS

Start of Block: Tool Eval **Q40.** Please rate how much you agree or disagree with the following statements based on your experience interacting with OSS-Doorway. **Scale:** Strongly disagree to Strongly agree.

- Using OSS-Doorway was easy for me.
- OSS-Doorway is useful to learn how to contribute to OSS projects.

Q49. How likely are you to recommend OSS-Doorway to a friend or fellow student who wants to learn about open source contribution? **Scale:** 0 to 10.

End of Block: Tool Eval

Start of Block: Gamification **Q47.** The following questions are about the gamification elements in OSS-Doorway (quests, progression, badges/points, etc.). **Scale:** Strongly disagree to Strongly agree.

- The gamification elements (points, badges, progression) made the learning experience more engaging.
- Seeing my progress through the quests motivated me to continue.

Q48. Any comment about the gamification elements?

End of Block: Gamification

Start of Block: Tasks Q6. Tasks refer to the individual, structured activities within each biweekly quest. **Scale:** Strongly disagree to Strongly agree.

- Tasks improved my understanding of class requirements and learning objectives.
- Compared to standard assignments, tasks gave me more actionable guidance.

End of Block: Tasks

Start of Block: Validator Q11. The built-in answer validator is the automated checker inside OSS-Doorway that reviews your work (such as code, issue responses, or pull requests) and gives immediate feedback on whether you completed the quest correctly. It also provides helpful feedback on next steps whenever you do not quite get the answer right. **Scale:** Strongly disagree to Strongly agree.

- The validator’s feedback influenced how I adjusted my answers.
- The validator helped me identify specific mistakes.

Q23. Any comment?

End of Block: Validator

Start of Block: Open questions

- **Q16.** How would your learning have been different without the bot and quests?
- **Q37.** What aspects of OSS-Doorway were most helpful for your learning? Why?
- **Q38.** What aspects of OSS-Doorway were least helpful or most frustrating? Why?
- **Q17.** What suggestions do you have for improving OSS-Doorway?

End of Block: Open questions

Start of Block: Demographics Q32. Introductory text: demographic responses are reported only in aggregate.

- **Q31.** What is your gender? (Man; Woman; Non-binary; Other; Prefer not to say)
- **Q43.** Are you enrolled in CS499 or CS599? (CS499 Undergraduate; CS599 Graduate)
- **Q33.** How many years of *professional* experience do you have in software development-related roles? (Less than 1 year; 1–2 years; 3–4 years; 5–9 years; 10+ years)
- **Q34.** How would you rate your experience with GitHub previous to this class? (No experience; Beginner; Intermediate; Advanced)
- **Q35.** How would you rate your experience with OSS previous to this class? (No experience; Beginner; Intermediate; Advanced)

End of Block: Demographics

Start of Block: Thank you Q18. Thank you for participating! Your feedback directly helps improve OSS-Doorway and future course design.