Title: Retrospective report

Student name: Matthew Chung Kai Shing

Student ID: 34102450

Team id: CL_Friday4pm_Team8

Table of Contents

Generative AI was not used to generate this report.

Introduction

This report documents the design and delivery of a custom Learning Management System (LMS), NotMoodle that supports current and future students and teachers in their educational journey .Over three sprints, the team followed the Scrum framework and set of processes with weekly in-person client sessions, providing a working demo to the client during a sprint review at the end of each sprint. For the project, the team used Django web framework, PostgreSQL, Jira and GitLab. The client was happy with the implemented features, noting one minor improvement area in the dashboard text readability.

Project context (brief)

Team. CL_Friday4pm_Team8 (7 members). The Product owner and Scrum Master role was held throughout. In Sprint 3 a team member volunteered to act as SM with support from the original SM.
Product. A web application working on all types of devices designed to help teachers manage courses, lessons, classrooms and grading . Students enrol, view lessons and submit assignments. Delivered scope included student/teacher role access, course/lesson creation and management, classroom tracking, student progress/reporting, and Google Sign-In and an AI chatbot as nice to have features.
Stakeholder & Product Owner. Weekly collaboration with (the TA as) the client; an internal team member acted as Product Owner to maintain the backlog and priorities.

Topic 1: Agile software development practices

Overview of our approach

We delivered three sprints (2w, 2w, 3w: 26 Aug–21 Oct) with **working demos** showcased to the client at the end of each sprint. I was **Scrum Master** throughout (although for S3 another member volunteered to be SM, however I assisted). We met the client **weekly in person on Fridays**, used **Jira** for Product/Sprint Backlogs and **GitLab** for branching, MRs, and reviews.

Where we aligned with Scrum

- We had regular reviews **with shippable increments** (agreed tangible working features). Our retros produced **concrete changes** (e.g., sprint 3 had more frequent merges with summary and reviewer added, stricter Acceptance Criteria usage with checkboxes, an all-team physical working day introduced).
- The Product Backlog lived in Jira, with user stories following INVEST criteria and **prioritising client value**. We added missing flows when discovered (e.g., the “select role” UI user story missing for S1) and **re-ordered user stories** promptly. We added NFR (multi-device) requested from client in S2.
- We used **estimated story points** based on the complexity and the time required for a task and **tracked velocity/burndown (**see appendix Figure 1 & 2); (e.g., grading page re-estimated from 2 to 4 pts in S3). Team spirit and throughput improved as we took on 29 (S1), 50 (S2) and 60 (S3) story points.

Key deviations and why they mattered.

- We **did not run a daily 15-min** event due to only one allocated class per week. Instead, we held **two standups/week** (applied + Sunday 3pm).In S3, we introduced a whole-day co-working session to accelerate integration and unblock questions. This **reduced early visibility of blockers**. We mitigated with proactive SM facilitation and async team updates via Discord.
- Early inconsistency (in S1 and S2, some stories closed without any AC/two reviews). In S3 we **tightened DoD** to include **tests and CI**, added **AC checkboxes**, and enforced reviews.
- We **centralized approvals** ("no one merges to main except SM "). It raised throughput and consistency but slightly reduced the recommended " **self-organising team"** scrum approach.

Customer collaboration and how it evolved

We met the client **weekly in person** and during which we **clarified requirements**, and flowed outcomes directly into Jira (new PBIs, updated AC, changed priorities). **Assignments** originally tied to **classrooms** were moved to **lessons** after clarification; related PBIs were updated and re-estimated. This cadence of collaboration reduced waste (we stopped showing unfinished work) and supported **incremental value delivery** each sprint.

Agile Manifesto principles we followed

- Front-end team trusted back-end with the logic implementation, collaborating through Discord. ("I have completed the UI for student dashboard, pls have a look and let me know if not good**.") valuing individual and interactions** and using tools to support that.
- Each sprint we had a **workable product that provided tangible value** to the client. The prompt feedback received up-front documentation and reduced the risk of investing resources into features that might not be relevant by the time the product is delivered.

What we would do differently (opportunities for improvement)

1. Establish a **dedicated weekly backlog refinement session** instead of doing it in class.
   Benefit: Earlier dependency discovery and right-sized stories.
   Measure: Fewer carry-overs and smaller average story size; and blockers.

2. Add **Definition of Ready** to let team know if a backlog item is ready to be added to a sprint and to improve backlog refinement.

<u>Topic 2: Working in teams</u>

Coordination:

We organised around **component teams** with two members on front-end, three on back-end, and two on integration/bug-fixing. Moreover, we shared know-how in **Markdown guides** (e.g., TEST_QUICK_START.md, AI_ASSISTANT_GUIDE.md) so everyone could run, test, and understand the code structure better.

Conflicts

- **No consistency in code structure & naming conventions :** for e.g., student_management vs teachersManagement, and mismatched login URLs. (students/login vs login/teachers)
  Why: parallel work and different personal coding style/experience.
  Resolution: In week 2, SM drafted a team learning plan (:Team learning plan) to get the team started to learn the basics of Django. In Swotvac week, SM ran a short code structure walkthrough (where to place new features, consistent naming, "don't copy-paste blindly from ChatGPT"). This increased shared understanding and reduced rework.
- **Branch/merge friction**. A teammate worked from an outdated branch (we had ~10 feature branches at the time), causing merge conflicts and duplicated UI work.
  Resolution: reinforce "pull before start" and agreed on a minimal branch naming convention (Merged_Everything_0.1 branch was confusing).
- **Dependencies**: grading depended on classroom, so a member had to wait for other member before starting his work. We responded by making user stories as independent as possible and reassigning some free members on more tasks.

Task allocation, distribution & completion

We assigned user stories based on: **"are you OK doing this?",** ensuring every backlog item had an owner. In S1 and S2, some members did more/less work but S3 was more **balanced with clearer splits** (divided story points more equally).

Flow & completion

- The Cumulative Flow Diagram (see appendix Figure 3) across the 3 sprints showed a steadily growing Done band and a thin Review band which indicated controlled Work In Progress and regular throughput throughout.
- A few items bounced from Done to Reopen in Sprint 1 (e.g., "Welcome page UI" lacked agreed standards). We **updated the DoD** and **added acceptance-criteria checkboxes** to prevent premature closure.

Communication methods

We used **Discord + Jira** for about **70% asynchronous** coordination and **30% synchronous** via in-class time and our Sunday check-in. We aimed to **acknowledge messages within 24 hours** (a thumbs-up counted). Across the whole project, members happily abided to that.

Scalability to an industry context

What would **scale** well?	What we would **change** for industry
Async-first comms norms with a 24-hour acknowledgement. This will make sure that team members are not being ignored, and teams are well informed	Move from components team to **cross-functional** team. This would reduce handoffs and dependency waiting.
Living markdowns (e.g README.md) helps for handovers and address any potential confusions in the future	**Daily** 10–15 min standups. This would provide short, outcome-oriented updates unblock within 24h

Tuckman's stages in our team (Tuckman, 1965; Tuckman & Jensen, 1977)

- **Forming (~Week 2):** friendly start, some shyness in the early Lego team activity.
- **Storming (mostly S1–S2):** disagreements on structure/naming and stories dependencies. Addressed via the SM's structured session, stricter PR habits, and clearer splits.
- **Norming:** norms emerged -"only SM merges to main," pull latest before starting, AC checkboxes, and .MD guides.
- **Performing (start of S3):** swarming(working together on a single, high-priority story (classroom/grading feature until it's completed), smoother reviews, steadier flow (as seen in CFD).
- **Adjourning:** we left run/test guides (TEST_QUICK_START.md) and how-to docs for project continuity.

Topic 3: Decision making

Initial choices (what ,why)

We agreed early to build a **responsive web app** using **Django 5 web framework** written in **python 3**,plan in **Jira**, and host the code on **GitLab**. This was because

- everyone was comfortable with Python (Team Technical Interests), (SM had Django experience as well) which allowed for a lower learning curve.
- Django's built in features (auth, forms, admin, migrations) allowed for less boilerplate than split API stacks. We accepted a Django learning curve for members in exchange for increased development speed
- We rejected **MERN** (more layers, dual deploys and learning) and **Laravel/PHP** (new language) under a 3-sprint deadline.

What we'd revise in hindsight.

- Start **on PostgreSQL DB from day one** (we had to switch DB since we were using the default db.sqlite3 which did not allow for pgvector to use for our AI Chatbot). This would have avoided the need to repopulate a new database.
- Create a 1-page **Architecture Decision Record** (ADR) for the stack to have clearer app/URL naming as from start of project.
- Make **tests and CI part** of the Definition of Done **as from Sprint 1**. We would have had less merge conflicts

Requirements management

We used **the weekly client sessions** and **Jira acceptance criteria** to confirm scope. Key changes handled included : adding "core unit" field to Courses, assignments tied to lessons (not classrooms),and adding device compatibility NFR. Our confidence in requirements rose to **medium-high** by Sprint 3. (fewer clarifications questions and smaller rework)

Documentation: Key choices were summarised in **IMPLEMENTATION_SUMMARY.md** (refer to Gitlab)

Impact on product quality

Good calls: Django's stack let us focus on domain logic. The **protected main (only SM can merge to main), reviews and AC checklists** reduced the "done-but-not-done" reopens. Retros led to **swarming** on S3 sprint backlog.

Trade-offs: **Late CI/testing** created early merge difficulties. **Django learning curve** slowed S1and **vague S1 stories** caused re-discussion. We also had timing mistakes (some UI had to be reworked to match the website theme; **AI** feature integrated a bit late led to some team members not really understanding how it worked).

Outcome: Product meets client's needs with nice-to-have features of **Google Sign-In** and an **AI chatbot**.

Future decision process: The decision for choices is **formalised** so that choices are faster, better evidenced, and easier to revisit. We'll classify choices as Amazon's **two-way doors** (reversible choices which we can decided quickly) vs **one-way doors** (hard to reverse that is run a half-day spike first, then decide) (*Narrative: Slater (n.d.*)). All significant decision will get assigned an owner and a timebox (e.g., 48 hours in-sprint) to avoid drift. When consensus stalls, we'll vote once and "disagree & commit," with the SM as the tie-break.

Topic 4: Ethical aspects

If deploy in industry, our project faces material ethical risks around **student data privacy & security** and **AI chatbot governance**.

1) Student data privacy & security (PII, OAuth, grades)

Unauthorised access and misuse of grades or identities can happen when roles are over-permissive (e.g., a teacher can view/alter **any and all** students marks from **any and all** classrooms) or when **shared accounts** are used (for e.g a generic admin account are being reused by **multiple staffs),** enabling viewing/changing grades or seeing sensitive notes such as disability/health conditions of students. This could cause **academic harm** (lost scholarships/progression from incorrect marks), **reputational harm** within cohorts, and **phishing risk** from exposed contact details. Such outcomes **breach the ACS Code of Ethics 1.2.1 Primacy of the Public Interest** which has the duty to prevent harm to individuals and **1.2.6 Professionalism** which refers to the obligation to apply appropriate access controls and accountability of data. Moreover, **gathering more PII than necessary** or hiding third-party use (e.g., login using Google permissions) **violate 1.2.3 Honesty** and erode trust, since students are not aware of how their data is used.

Mitigation:

- Reinforce access control and auditability by having MFA and unique accounts (shared logins if really necessary).
- Add **least-privilege Role Based Access Control** so tutors can access only their classes ,log every grade read/write and **run quarterly access recertification** to reduce insider misuse.

- Make data handling transparent and minimal by **showing a plain-English privacy notice at sign-in.** Mention what is collected and why and any third-party/OAuth scopes are to collect only necessary fields and separate identity from grades.

2) AI chatbot ethics (accuracy, bias, academic integrity)

AI responses could be **biased** and even hallucinations, misleading students about certain content or policies. The chatbot could also potentially **encourage cheating** by revealing solutions to graded tasks, or even expose **private information** if students paste in sensitive data. Such outcomes breach the **ACS 1.2.2 Enhancement of Quality of Life** and **ACS 1.2.3 Honesty** because it could negatively impact students' learning outcomes and not being fair towards other students.

Mitigation:

- Constrain the bot to **answer with only approved sources** (provided course material and official guidelines) and if question is not covered there, politely refuse or send user to human teacher.
- Display a **"AI may not be accurate please double-check with official sources" reminder message** clearly to students.
- **Block assessment-like prompts** during assessment and log/notify teachers if any students try to cheat.

Appendix

4.1 Figures screenshots

1) Velocity report (Figure 1)

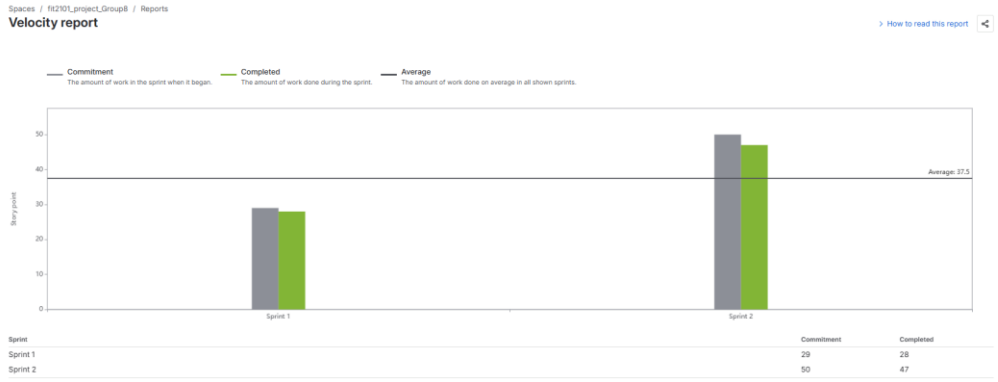

Figure 1 (* does not show S3 since we wanted to keep the sprint backlogs in the “Board” for S3)

2) Burndown chart (Figure 2)

(Figure 2)

3) Cumulative flow diagram(Figure 3)

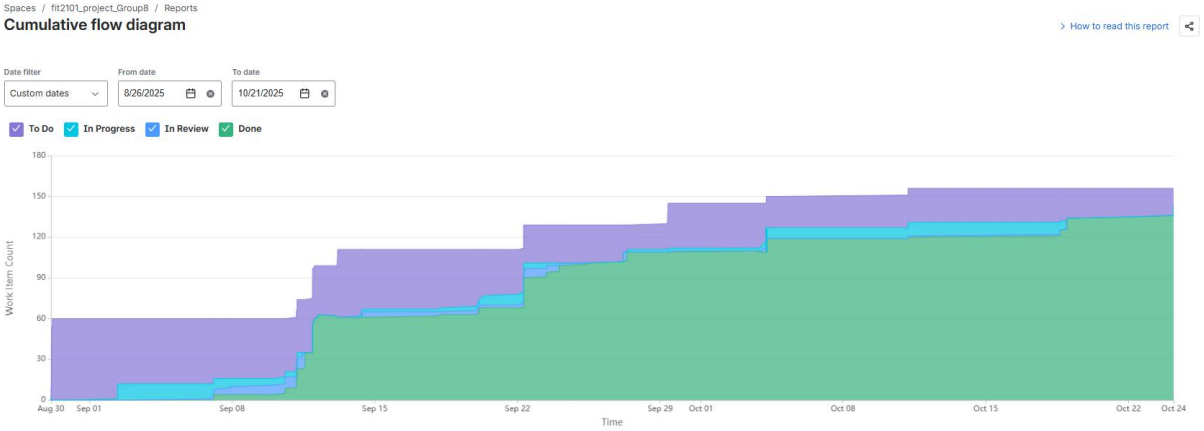

Figure 3

ACS Code of Ethics

4.2 References

Tuckman, B. W. (1965). Developmental sequence in small groups. *Psychological Bulletin, 63*(6), 384–399.

Tuckman, B. W., & Jensen, M. A. C. (1977). Stages of small-group development revisited. *Group & Organization Studies, 2*(4), 419–427.

Slater, D. (n.d.). *Elements of Amazon's Day 1 culture*. AWS Executive Insights. Retrieved November 1, 2025, from https://aws.amazon.com/executive-insights/content/how-amazon-defines-and-operationalizes-a-day-1-culture/