UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Software Engineering Document

COS750 Exam Assignment
Factory Method

Group 7

Charl Pieter Pretorius: u22519042
Ivan Gareth Horak: u21456552
James Alexander Fawkes Fitzsimons: u21516741

November 16, 2025

Contents

1 Purpose & Scope

Build a post-lecture practice and assessment web app that helps COS214 students recognise, diagram, and implement the Factory Method (FM) pattern, and produces intervention reports for lecturers on FM misunderstandings and prerequisite C++ gaps (constructors/destructors (ctors/dtors), virtual dtor, ownership). Students are assumed to have attended the lecture; the app supplements it.

Primary goals

- Deliver micro-quizzes + dynamic formative assessments tied to specific LOs (Learning Outcomes) from the ID doc.
- Provide a coding practical environment with automated tests and feedback.
- Create an environment where a UML diagram can be created or edited to be assessed against a provided context.
- Generate actionable analytics per LO and per C++ prerequisite to support intervention.

2 Development Approach (SE)

Use a lightweight Agile / Iterative approach (2x1-week sprints) with an MVP cutline:

Sprint 1 (MVP): micro-quiz engine, UML label/build interactions, code runner for the FM practical, LO-tagged analytics, intervention notification.

Sprint 2 (Polish): micro-lessons, AF vs FM discrimination items, accessibility passes, lecturer dashboard.

3 Users & Key Scenarios

Student: completes micro-lessons (Theory T* and Coding C*); after each set of three micro-lessons, the system serves the next Micro-Quiz (MQ); runs the FM coding practical; views mastery heatmap and feedback.

COS214 Lecturing Team: views cohort dashboards, LO mastery, top error classes, and C++ prereq gaps flagged by items.

4 Functional Requirements

An asterisk (*) implies it does not form part of the MVP.

- Module Home (FM): shows goals, time budget, and entry points (UML strand, code strand, recognition strand).
- Micro-Lessons*: short pages with dual-coding (UML + text) and quick “try-it” bits (e.g., identify Creator vs Product).

- Micro-Quizzes (MQ): triggered after each **three micro-lessons (T*/C*)**, 5±1 marks, mixed question types (UML-label, code-read, trace, refactor), immediate feedback, LO-tagged. MQs themselves do not count toward the three.
- FM Coding Practical: scaffolded C++ task (Creator/Product with make()), unit test execution with structure checks (e.g., no new Concrete* in client).
- UML Workbench: drag-to-label roles, add abstract markers, set return types; code→UML and UML→code-signature tasks.
- Pattern Discrimination*: triage FM vs Abstract Factory vs Simple Factory with one decisive cue.
- Analytics: store attempt-level events {user/session, item_id, lo_ids[], pass_fail, time_ms, error_class} and compute mastery bands per LO.
- Reports: per-student and cohort roll-ups; "intervention reasons" by error class (e.g., wrapper-around-new, wrong return type, no virtual dtor).
- Accessibility & Preferences*: captions, transcripts, keyboard-first navigation, high-contrast theme; student can pick preferred modality.
- Formative policy (from ID): total MQ time $\leq$ 60 minutes and $\leq$ 30 marks across MQ1-MQ6; unlimited retries for learning, but only the first graded attempt counts; variants are parameterised.

5 Non-Functional Requirements

- Usability: first correct attempt rate on easy items $\geq 80\%$; keyboard-only path for all actions.
- Performance: item load < 3.0 s (p95); code tests return < 10 s (p95) per run.
- Reliability: autosave state; graceful retry on network hiccups.
- Maintainability: content in YAML/JSON with LO tags; CI on content schema.
- Security & Privacy: minimal PII (anonymous or student number); store only necessary analytics for intervention.
- Accessibility: WCAG-aware (alt text, focus states, ARIA live regions for result panels). High Contrast, text-to-speech.

Target platforms

Desktop browsers: Chromium.
Minimum viewport: 1366×768. Keyboard-only navigation supported. High-contrast theme available.

6 Architecture (high level)

Front-end (SPA): React + TypeScript, Monaco editor, custom UML canvas. Feature slices: Lessons*, MQ Engine, UML Workbench, Code Practical, Analytics views.

API: FastAPI/Node for content delivery, item evaluation, and analytics ingestion.

Grading Service: containerised C++ runner with unit tests + static checks (grep/AST) for FM invariants. Gemini-2.5-Pro API for formative feedback.

Store: PostgreSQL (content, attempts, mastery), object storage for static lesson assets.

Auth: simple session; prototype may run anonymous sessions if allowed.

MQ Scheduler*: Emits a Micro-Quiz (MQ) after each **three** completed micro-lessons (T*/C*) on a per-user path; maintains a rolling counter across reloads. Feature-flagged off until micro-lessons ship.

Key integration

- **Code practical flow:** upload or inline edit → build in sandbox → run tests → return structured results (per-check verdicts + hints).
- **UML workbench:** client-side checks for labels/markers; server verifies on submit for consistency.

Environments & deployment MUST STILL BE UPDATED

Mono-repo with Docker Compose. Services:

- `web` (SPA) — React build served by a lightweight web server.
- `api` (FastAPI/Node) — content delivery, item evaluation, analytics ingestion.
- `grader` (C++) — containerised runner with unit tests and static checks.
- `db` (PostgreSQL) — content, attempts, mastery.
- `obj` (object storage or local volume) — lesson assets and fixtures.

7 Data Model

- `learning_outcome(lo_id, name, bloom, strand)`
- `item(item_id, type, prompt, lo_ids[], difficulty, error_classes[])`
- `attempt(attempt_id, session_id, item_id, mq_id, outcome, attempts_n, time_ms, error_class, remedial_clicked)`
- `mastery(session_id, lo_id, band, updated_at)`
- `practical_run(run_id, checks_json, tests_passed, time_ms)`
- `user` (optional for named cohorts)

8 Content Authoring & Pipeline

- Content format: YAML/JSON; each item lists lo_ids[], error_classes[], and render_spec (e.g., UML nodes/edges).
- Validation: schema check in CI; preview tool for authors.
- Randomisation: parameterised names and class variants to deter memorisation.
- Traceability: every item → LO(s) (reuses your ID mapping table).

9 Item Types (engine support)

`UML_LABEL, UML_BUILD_FROM_CODE, UML_SCAN, UML_DESIGN_FROM_CONTEXT,`
`CODE_READ_ROLE, CODE_TRACE_OVERRIDE, CODE_FIX_LIFECYCLE,`
`REFACTOR_TO_FM, EXTEND_NEW_PRODUCT,`
`MCQ_INTENT, SCENARIO_DECISION, PATTERN_TRIAGE.`

10 Testing Strategy

- Unit: item validators, LO mapping, hint selection.
- Integration: code-runner path (compile errors, timeouts), UML canvas → evaluator.
- E2E: happy path (student completes a module), network-loss recovery.
- Accessibility: axe-core scans + manual keyboard audits.
- Content QA: rubric consistency; AF vs FM discrimination sanity checks; General assessment through Gemini-2.5-Pro.
- Logging smoke test: API emits JSON on success and error paths; client error hook posts redacted payload.
- Health checks: liveness/readiness endpoints return 200 within 100 ms on warm services.
- Scheduler cadence (unit)*: given a fresh session, emit no MQ after 1–2 micro-lessons; emit exactly one MQ after micro-lesson #3; then again after #6, #9, … MQ completions do not affect the counter.
- Scheduler LO-selection (integration)*: with mixed T*/C* sequences, verify the emitted MQ targets the last three micro-lessons' LOs and respects completion/persistence across reloads.
- UML design validator (unit)*: enforce role constraints (Creator abstract, factory op return = Product, no client→Concrete deps) on the canvas graph.
- UML design flow (integration)*: given a short brief, user can place participants, add generalisation and dependencies, and pass validator checks; persisted and restored on reload.

11 Risks & Mitigations

- C++ compile latency → cache and small test sets; pre-warm containers.
- False positives in static checks → pair static checks with unit tests; keep rules transparent in feedback.
- Scope creep → stick to MVP cutline (below).

12 MVP Cutline

Included

FM strand: MQs, coding practical with tests, UML workbench (label/build), analytics + CSV export.
Note: until micro-lessons ship, MQs are accessible via a direct "Start MQ" flow; the MQ scheduler is feature-flagged off and enabled once micro-lessons are available.

Deferred/Nice-to-have

Micro-lessons, pattern discrimination, rich lecturer dashboards, cohort compare over time, item authoring UI (use files in prototype).

13 Work Plan (example)

Sprint 1 (week 1): content loader, MQ engine, UML label, code runner v1, analytics ingestion, 40 core items.

Sprint 2 (week 2): UML build, AF vs FM triage, refactor/extend tasks, accessibility polish, lecturer CSV export, 20 more items.

Traceability (ID ↔ SE)

Table 1: Goal → Feature → Assessment → LO mapping (MVP unless marked *).

ID	Goal (from ID)	App features (MVP / *polish)	Formative (MQ)	Summative* (from ID)	Core LOs
G1	Creation variability & why creational	MCQ_INTENT items (intent/recognition); Module Home cues; analytics tag for "intent/strategy"	MQ1	Q1 (Intent MCQ/FITB)	LO4, LO6
G2	Canonical FM & "client must not construct concretes"	UML Workbench (UML_LABEL, UML_SCAN); CODE_READ_ROLE; grader invariants (no `#include "Concrete"`, no `new Concrete`, no type switch)	MQ1, MQ5	Q11 (Refactor)	LO1–LO4, LO9
G3	UML structure & notation; produce/read/locate FM in diagrams	UML_LABEL, UML_BUILD_FROM_CODE, UML_SCAN; *UML_DESIGN_FROM_CONTEXT (palette, signatures, constraints)	MQ2, MQ3	Q3 (Label), Q4 (Build from code), Q5 (Scan), Q13 (Outline from UML), Q14 (Translation duet), *Q6 (Design from brief)	LO5, LO7, LO10, LO11*, LO12, LO13, LO17, LO23
G4	Code role cues & conformance	CODE_READ_ROLE, CODE_TRACE_OVERRIDE, CODE_FIX_LIFECYCLE	MQ4	Q7 (Role classify), Q9 (Trace), Q10 (Lifecycle fix)	LO7, LO14, LO19
G5	Pitfalls & misconceptions (wrapper, switch, tight coupling)	Smell-detection + seam-pointer items; error classes feed reports	MQ5	Q15 (Smell + seam)	LO14, LO19, LO21
G6	Correct C++ realisation & UML↔code conformance	Grader invariants; REFACTOR_TO_FM; EXTEND_NEW_PRODUCT; round-trip checks	MQ4–MQ6	Q10 (Lifecycle), Q11 (Refactor), Q12 (Extend), Q14 (Translation duet)	LO10, LO14, LO16–LO18, LO20–LO23

ID	Goal (from ID)	App features (MVP / *polish)	Formative (MQ)	Summative (from ID)	Core LOs
G7	Related patterns (recognition)	Pattern triage (FM vs AF vs Simple) *	MQ6	Q16 (Pattern discrimination)	LO8, LO9, LO12, LO15, LO24*
G8	Guided hands-on culminating in refactor/transfer	FM coding practical with tests + static checks; extend step	Practical (formative use possible)	(Practical rubric)	LO9, LO11, LO16, LO21, LO22

14 Assumptions & Constraints

- Students attended the lectures first; web-app is supplemental.
- Assessment must surface why intervention is needed (mapping to potential LOs).
- Pattern content must follow pure forms (avoid collapsed hierarchies unless teaching why).

15 Deliverables

- Running prototype (SPA + API + grader).
- Item/content bundle (YAML/JSON) and sample analytics export.