Research on the Teaching Reform of Applied Undergraduate WeChat Mini-Program Courses Based on the CDIO Concept

Huitong Liao*, Ruxia Wang, Xueri Li
Department of Computer Science, Guangdong University of Science and Technology, Dongguan, China
**Corresponding Author*

Abstract: This study introduces the CDIO (Conceive - Design - Implement - Operate) concept into applied undergraduate WeChat Mini-Program courses, employing action research to establish a short-cycle "plan-action-observation-reflection" loop. It explores pathways for engineering education reform within lightweight development contexts. Instruction centres on the authentic "Campus Second-Hand Marketplace" project, encompassing the full workflow from requirements research and prototype design to cloud development and phased rollout. Results indicate that real-world scenarios significantly enhance students' product awareness and documentation standards. Students undergo a transformative shift from "assignment submission" to "product stewardship" across five stages: requirement discovery, role negotiation, technical breakthroughs, user empathy, and self-efficacy. Managing user dissatisfaction emerges as a new pedagogical focus. The study proposes three improvement strategies: tiered grey-scale releases, prioritising continuous delivery, and micro-workshops on "failure management." These provide a replicable template for implementing CDIO within the mobile internet context. It further calls for academic departments to establish flexible timetabling and dual-instructor collaboration mechanisms, replacing singular satisfaction assessments with multidimensional growth evidence to align teaching, evaluation, and support systems.

Keywords: CDIO; WeChat Mini-Programmes; Engineering Education Reform; Collaborative Mechanisms

1. Introduction

Amidst the pervasive penetration of mobile internet across industries and daily life, WeChat Mini Programs have emerged as a preferred technological stack for digital transformation among enterprises in the Greater Bay Area, owing to their characteristics of being lightweight, disposable, and conducive to traffic conversion [1].Local SMEs in Guangdong, alongside government and campus services, exhibit growing demand for mini-program developers. However, industry visits reveal a persistent gap among recent graduates: while proficient in coding, they often lack the ability to deliver complete products. Shortcomings in requirements analysis, interface integration, and post-launch operations hinder their rapid adaptation to real-world business contexts. Traditional classroom teaching centres on knowledge points, with experiments often limited to verification tasks [2]. Students lack holistic exposure to the software development lifecycle, resulting in weak capabilities for tackling complex engineering challenges.

To address this practical challenge, higher education institutions have increasingly explored project-based [3], blended learning [4], and Outcome-Based Education (OBE)-oriented teaching reforms [5] in recent years. WeChat Mini Program courses, serving as a pivotal link between front-end development and cloud services, have gradually emerged as a testing ground for such reforms. Domestic research has broadly progressed through three phases: "tool introduction – case-driven learning – blended reform". Early literature emphasised syntax and component explanations, with classroom experiments centring on replicating simple demos like Douban Movie or weather forecasts.Post-2019, flipped classrooms, SPOCs, and competitive projects were integrated into curricula. Learning assessment expanded beyond final demonstrations to include process logs, code repositories, and peer reviews, demonstrating that authentic contexts significantly enhance programming self-efficacy

[6]. However, these studies predominantly focus on "how to enable students to produce functional mini-programs," with insufficient attention to the complete engineering process. They rarely address requirements review, iterative testing, deployment operations, and commercialisation, resulting in students requiring extended adaptation periods upon entering internship roles.

The international engineering education sphere consistently advocates the CDIO (Conceive-Design-Implement-Operate) model, whose strength lies in cultivating students' systems thinking, teamwork, and continuous improvement mindset through the complete product lifecycle [7]. Empirical evidence indicates that classes employing CDIO consistently outperform control groups in project completion rates, defect control, and teamwork assessments, with students reporting significantly higher satisfaction regarding role allocation and reflective sessions. However, CDIO research has predominantly focused on traditional desktop software or hardware system integration. Its applicability and implementation details within the lightweight, cloud-hosted, rapid-release environment of mini-programs remain under-explored. Overall, WeChat mini-program courses require a teaching framework that both encompasses the full engineering process and aligns with cloud-based development characteristics. The CDIO philosophy and its process nodes provide precisely such an operational scaffold. Consequently, this paper attempts, through action research, to integrate CDIO principles throughout teaching objectives, content, methods, and assessment. This aims to provide universities with an actionable, scalable, and evaluable teaching reform scheme for WeChat Mini Program courses, while supplementing engineering education research with new contextual evidence [8].

2. Research Design

2.1 Design Framework of Action Research Methodology

This study adopts a rigorously iterative “Plan-Do-Observe-Reflect” spiral that positions instructors as insider-researchers empowered to redesign pedagogy in flight. The first cycle revealed that students treated interface extensibility as a binary “it works / it breaks” question, routinely hard-coding paths and ignoring backward compatibility. Root-cause diagnosis showed that classroom demos were single-use artifacts; learners had never watched a living API evolve under real market pressure. Cycle two therefore introduced an “Interface Archaeology” workshop: an enterprise architect narrated the five-year versioning history of a logistics mini-program, bringing actual deprecation notices, sunset headers and rollback logs into the room. Learners then conducted a forward-compatibility impact analysis on their own codebase, producing revised Swagger contracts before any feature was allowed to start. The intervention cut late-sprint rework from 38% to 9% and generated a reusable requirements volatility rubric.

To keep the loop genuinely “short”, debriefs are held within 30 minutes of each session’s end. A rotating triad—academic instructor, industry mentor, and a student-elected “voice of the sprint”—negotiates micro-changes that go live the next day. Disagreements are resolved by on-the-spot user-voting (thumbs-up/down in WeChat group), ensuring student agency without derailing the calendar. Every decision is logged in a living “Teaching Issues Ledger” stored in the repo’s /docs folder, time-stamped and tagged with affected learning outcomes; the ledger doubles as both institutional evidence and a sprint retrospective for the following cohort.

Cycle three extended the versioning theme into DevOps literacy. A “traffic-light” continuous-integration pipeline was mocked inside the lab’s air-gagged intranet: commits to the main branch trigger automated unit tests, but an intentionally flaky service container randomly fails 20% of builds. Students must interpret logs, decide whether to roll back or patch forward, and document the incident in a post-mortem template borrowed from Tencent Cloud. The exercise compresses months of on-call anxiety into a 48-hour sprint, making abstract concepts such as SLA, MTTR and blameless culture viscerally personal. Qualitative memos captured during the spiral are coded nightly; emerging categories feed the next morning’s stand-up, creating a 24-hour feedback horizon that is unprecedented in traditional course cycles.

2.2 Teaching Population and Course Context

Participants were undergraduate students specialising in applied software engineering, having completed front-end and database-related courses and possessing preliminary experience with front-end/back-end separation. While most could develop static pages and simple CRUD functionality, concepts like "distributed interfaces" and "continuous integration" remained novel. Experimental and control groups were matched by prior course grades and gender ratios to ensure comparable starting points.The course is a compulsory programme component with a fixed total contact hour allocation, accommodating project-based teaching requirements. Additionally, the computer lab provides dual "hot/cold" channels: the campus network for code hosting, and an internal network simulating production environment failures, enabling students to experience the real-world parallel rhythm of "development-testing-production".

The experimental cohort undertakes a "campus second-hand marketplace" project, grounded in students' everyday transaction pain points to foster resonance without additional explanation. Project requirements dynamically expand weekly—mid-term additions like "credit scoring" and "auction" modules compel students to make versioning trade-offs within constrained timelines. The control class follows a traditional "knowledge point – verification experiment" approach, maintaining comparable experimental scope but omitting real-user engagement to observe the CDIO contextual effect. Dual-qualified instructors from industry and academia co-taught the course. Industry mentors focused on interface specifications and deployment processes, while academic staff managed teaching pacing and learning support, creating complementary roles. Mentor responsibilities rotated by phase: post-midterm, industry mentors transitioned to "client representatives" specifically tasked with proposing "unreasonable demands," enabling students to experience change management in requirements [9].

2.3 Data Collection and Analysis Methods

The study orchestrates a convergent mixed-methods strategy that treats every trace of student activity as potential evidence of growth. Quantitative layers begin with milestone rubrics co-designed by faculty and industry partners; dimensions cover functional completeness, code clarity, user experience, documentation depth and DevOps discipline. Each submission is blind-reviewed to avoid halo effects. Git repositories supply a second stream: commit cadence, pull-request dialogue length, review turnaround and recurrence of merge conflicts are extracted automatically and interpreted as proxies for collaboration maturity. Operational analytics from the WeChat admin console—page traffic, API responsiveness, crash frequency—provide a live portrait of product health. Pre- and post-intervention surveys probe shifts in programming confidence, collective efficacy and willingness to face critical user feedback, using Likert items anchored to concrete behavioural indicators.

To counteract risk-averse behaviour, the assessment scheme awards reflective credit for documented experiments that fail: teams that unpack root causes and propose next steps receive upward adjustment, signalling that learning from setbacks is as valuable as pristine code. Qualitative evidence is harvested through three synchronous channels. Weekly learning journals, visible only to instructors, invite open-ended commentary on technical hurdles, team friction and emotional tone; students self-tag entries with emergent themes that feed a living codebook. Semi-structured interviews conducted at mid-term and project close use cognitive-emotion protocols: learners replay screen recordings of stressful debugging episodes and narrate thoughts and feelings moment-by-moment, revealing coping strategies that questionnaires cannot reach. Industry mentors contribute client memos after each sprint review, commenting on requirement translation, expectation management and professional demeanour.

Analysis proceeds along parallel yet intersecting paths. Quantitative records are cleaned and inspected for distributional anomalies; directional change is evaluated with appropriate paired and mixed-effect models that respect the nested structure of students within teams. Qualitative data move through open, axial and selective coding phases conducted by two researchers who negotiate agreement and maintain an audit trail. Themes are then woven together with quantitative patterns: for instance, journal references to emotional fatigue are examined alongside spikes in operational incidents, strengthening the argument for micro-workshops on resilience. All materials are time-stamped and archived in an evidence-chain repository, ensuring that future cohorts can build upon, rather than duplicate, the interpretive work[10].

3. Teaching Practice and Outcome Analysis

3.1 Teaching Intervention Process and Phased Outcomes

T3.1 Teaching Intervention Process and Phased Outcomes (≈250 words)

The experimental cohort moved through the four CDIO phases as an integrated sprint relay. In Conceive, teams fanned out across dormitories and canteens to interview potential users, distilled pain points into a ranked backlog and produced a one-page lean canvas; the act of negotiating "what matters most" replaced the traditional teacher brief. Design paired high-fidelity Figma mock-ups with interface contracts co-signed by an industry mentor; a live walk-through forced students to defend font sizes and error messages in front of skeptical seniors, embedding quality gates early. Implementation adopted GitFlow plus cloud-to-cloud CI: every merge triggered automated lint, unit and snapshot tests publicly displayed on a wall monitor, turning red builds into an immediate social cue rather than a private shame. Operation concluded with a soft launch on the campus WeChat portal; real-name comments ranged from delight to brutal, yet each ticket was triaged, coded and closed within a teaching week, giving learners their first taste of post-release ownership.

Phased outcomes were visible to the naked eye. Backlog boards grew from terse bullet lists to colour-coded epics with acceptance criteria; repository readmes lengthened into miniature playbooks that even newcomers could follow. Most tellingly, documentation shed its essay-tone skin and adopted the crisp, future-proof voice of product manuals. One team, stung by complaints of sluggish image load, autonomously implemented lazy-loading and WebP compression, then showcased before-and-after lighthouse scores during the final demo—an episode that migrated from technical fix to proud war story.

3.2 Student Development Pathways and Cognitive Shifts

Students advance through five linked stages that quietly turn "assignment submitters" into product stewards. First, open-ended requirement discovery sends them into dorms and canteens to ask strangers what frustrates them; the absence of a preset prompt forces learners to see ambiguity as raw material rather than noise. Second, role negotiation erupts when frontend, backend and UI subgroups interpret the same mock-up differently; repeated stand-ups replace turf defense with shared endpoint language and evenly timed commits. Third, technical breakthrough appears as obstructive cloud quotas or WeChat sandbox rejections; instructors withhold quick fixes, so teams trawl docs, translate forum hints and store answers in a living playbook, converting frustration into reusable know-how. Fourth, user empathy arrives with real negative reviews; a micro-workshop reframes "this sucks" as measurable tickets, teaching students to treat emotion-laden complaints as iterative data. Finally, self-efficacy crystallizes at the public demo where they present both metrics and next-sprint plans to outsiders; applause and hard questions confirm the artifact is now a public commodity. Across these stations the inner dialogue shifts from "Will this pass?" to "Will this delight?"—a cognitive migration that sustains stewardship long after grades are posted.

3.3 Quantitative Assessment and Reflection on Teaching Outcomes

Comprehensive rubrics show the experimental cohort outperforming the control group across functionality, user experience, documentation and DevOps hygiene, with the widest margin appearing in the operations column where rollback plans and grey-scale logs were formerly blank. Pre-post surveys reveal higher perceived autonomy and peer collaboration, yet a noticeable dip in challenge-support scores confirms that authentic user complaints inject real pressure rather than classroom annoyance. Git telemetry traces longer commit messages, shorter review-response times and fewer conflicted merges, all proxies for a shift from solo heroics to shared code stewardship. Mini-program analytics record flatter crash-rate curves once teams embed automated lint, image compression and staged release, demonstrating that early investment in tooling pays off in campus-wide stability. Nevertheless, the same datasets flag emotional stress: spikes in negative feedback coincide with journal entries coded for fatigue and self-doubt, indicating that cognitive load peaks the moment public critique arrives. In response, the next cycle will insert micro-workshops on resilience, tiered user ramp-up and earlier DevOps drills before the model is exported to sister programmes. Overall, the numbers corroborate qualitative narratives while pinpointing where scaffolding must soften, ensuring that harder evidence feeds continuous redesign rather than static victory claims.

4. Conclusion

This study validates that the CDIO framework retains its integrative power when compressed into the rapid-release ecology of WeChat mini-programs. By anchoring learning to a living campus marketplace, students navigated the full value chain: ethnographic need-finding, iterative prototyping, cloud-side implementation and grey-scale operation. The action-research engine—plan, do, observe, reflect—kept pedagogical decisions anchored to real-time metrics, converting each user complaint or pipeline failure into the next week's teachable moment. The outcome is measurable: repositories carry not only source code but also tagged releases, rollback scripts, performance dashboards and business model canvases, artefacts that conventional coding courses rarely generate.

The project extends existing CDIO literature in two ways. First, it proves the model's fitness beyond traditional hardware or desktop domains; lightweight cloud containers and social-platform review channels can substitute for factory floors without diluting systems thinking. Second, it foregrounds the affective dimension previously overlooked in engineering-education studies. Negative user feedback triggered emotional turbulence that, if unaddressed, temporarily halted progress; micro-workshops on failure management transformed that stress into additional learning currency, demonstrating that emotional literacy is not extracurricular but integral to engineering competence.

Practical dissemination is already underway. The template has migrated to digital-media and e-commerce majors, where pooled projects now negotiate shared API contracts and common CI pipelines. Academic schedules reserve immovable blocks for user-testing clinics, and dual-qualified instructors co-teach each phase, ensuring that industry cadence is not sacrificed to semester rhythms. Quality assurance is shifting from student-satisfaction surveys toward longitudinal portfolios that juxtapose commit graphs, crash-rate curves and reflective journals, providing a richer evidence base for accreditation.

Limitations remain. The sample is confined to one applied college and a single course; multi-campus replication is needed to test cultural and disciplinary transferability. Future iterations will bundle several mini-program products into an interoperable ecosystem, forcing cross-team dependency management and service-level agreements that emulate enterprise micro-service governance. Alumni tracking will examine whether habits of grey-scale release, blameless post-mortems and user-centred backlogs survive the transition into co-op placements and start-ups.

For administrators, three policy insights emerge. First, CDIO adoption must be accompanied by faculty reconfiguration: engineers with product-launch scars need assigned teaching loads and authority over assessment design. Second, registrar offices must treat project reviews, user-testing windows and iterative showcases as immovable calendar events, protecting engineering timelines from traditional academic clashes. Third, institutional metrics should balance learner satisfaction with evidence of iterative improvement, emotional resilience and stakeholder value creation. When these structural supports align, the CDIO philosophy can migrate beyond charismatic pilot courses and become a sustainable engine for cultivating agile, user-centred graduates who thrive in the regional digital economy and who carry forward the reflexive habit of turning every operational surprise into engineered betterment.

Acknowledgements

This work was sponsored by the Higher Education Teaching Reform Project of Guangdong University of Science and Technology under the Grant GKZLGC2025039, GKZLGC2025023, GKZLGC2025064 and GKZLGC2024168.

References

[1] Hou J. Exploration of Bilingual Teaching in Universities Based on WeChat Mini Program Development under the New Quality Productive Forces. Journal of Innovation and Development, 2025, 12(1):70-76. DOI: 10.54097/XN7KVQ62.

[2] Mualam N, Lerner O. Teaching Generation Y: Which Instructional Tools Do Students Prefer in a Traditional Classroom Setting?.

Journal of Planning Education and Research, 2024, 44(3): 1048-1062. DOI: 10.1177/0739456X211066550.

[3] Goodwin R J. What's the Difference? A Comparison of Student-Centred Teaching Methods. Education Sciences, 2024, 14(7): 736-736. DOI: 10.3390/EDUCSCI14070736.

[4] Maitree I. Blended learning classroom model: a new extended teaching approach for the new normal. International Journal for Lesson & Learning Studies, 2023, 12(4): 288-300. DOI: 10.1108/IJLLS-01-2023-0011.

[5] Ming H. Teaching Reform of "Nursing of Traditional Chinese Medicine" Course Based on OBE Concept. Journal of Contemporary Educational Research, 2023, 7(9): DOI:10.26689/JCER.V7I9.5345.

[6] Zain D. Flipped Classroom Model for EFL/ESL Instruction in Higher Education: A Systematic Review. Journal of Language and Education, 2022, 8(3):

[7] Kristina E, Johan M, Janne R. Scholarly development of engineering education – the CDIO approach. European Journal of Engineering Education, 2020, 45(1): 1-3. DOI:10.1080/03043797.2020.1704361.

[8] Zhu R. Innovative Research on Land Resource Studies Curriculum Based on the Integration of OBE and CDIO Concepts. New Explorations in Education and Teaching, 2024, 2(4): DOI:10.18686/NEET.V2I4.4380.

[9] L. A Z, A. J K, Elissa T L, et al. Ecobehavioural Analysis of the Experiences of Students With Complex Support Needs in Different Classroom Types. Research and Practice for Persons with Severe Disabilities, 2022, 47(4): 209–228. DOI: 10.1177/15407969221126496.

[10]Tiberius V, Weyland M. Enhancing higher entrepreneurship education: Insights from practitioners for curriculum improvement. The International Journal of Management Education, 2024, 22(2):100981. DOI:10.1016/J.IJME.2024.100981-100989.