Draft Only, Never Send: Hardening AI-Assisted Workflows in a University Research Lab

Joseph Benjamin R. Ilagan, Gabrielle Anne L. Uy, and Jose Ramon S. Ilagan

Abstract—AI-assisted workflows often start as open-ended prompting and end up as deterministic scripts. This report describes that shift in two workflows we built in a small university lab in early 2026: a geospatial classification pipeline over about 3,000 establishment records, and a reimbursement workflow that once sent an email we did not mean to send. One hardened because of cost, the other because of risk, but both the same way: we used a capable model to write explicit checks and guardrails, then let cheaper models or plain scripts run inside them. In software-engineering terms, the result is an execution harness around an unreliable component. That construct is old news. What we think is worth reporting is how the harness came together after the fact, in a lab that never meant to build one, and which failures are specific to a *multi-model* setup, where knowing which model actually acted matters for forensics and is almost never logged.

Index Terms—agentic workflows, AI hardening, agent harness, multi-model routing, LLM deployment, software engineering practice

Actionable Insights

- **Treat the first irreversible-action failure as a redesign signal, not a patch.** The constraint that emerges from the incident—"never auto-send," "bounding box plus keyword check"—tends to become the spine of the hardened workflow. Patching around it leaves the underlying fragility in place.
- **Log fallback routing state explicitly.** When a workflow cascades through multiple models under quota or credit pressure, knowing which model actually executed a given action is essential for incident forensics. This state is routinely not captured by default, and the gap becomes visible only after an incident, when it is too late to recover.
- **Budget drift detection as a permanent phase, not a cleanup step.** Most failures in hardened workflows come from context-starved models failing to hold structural consistency within constraints, not from the constraints themselves. The runbook grows over time, and that growth is the work.

J. B. R. Ilagan, G. A. L. Uy, and J. R. S. Ilagan are with the Ateneo Business Insights Laboratory for Development (BUILD), John Gokongwei School of Management, Ateneo de Manila University, Quezon City, Philippines.
An earlier, theory-framed version of this work is available as an SSRN preprint [2].
Manuscript received April 2026.

1 The Email We Did Not Mean to Send

Early in 2026, our lab's AI agent produced a reimbursement email addressed to institutional recipients, on a subject with real consequences if mishandled. We never preserved the exact trigger prompt, the payload, or the timestamp in the agent's logs. That gap turned out to be part of the lesson too. What we did preserve, across three separate memory files written in the hours that followed, was the constraint response. All three files said the same thing in slightly different words:

> **NO EMAILS:** Always save as Draft. Never auto-send.
>
> **Draft Only:** Never send directly. Always save to Drafts first.
>
> **NO AUTO-SENDING:** Always save emails as **Draft** first. NEVER call `gcloud gmail drafts create` directly.

That rule, "draft only, never send," is the spine of this report. We did not choose it in advance. We wrote it down, in all caps, at four in the morning, after the workflow did something we could not easily take back.

This article is about what happened next, and about a second workflow running in parallel that hardened for a completely different reason: money. Over one week in early February, a single AI provider billed us $292.82 across seven charges. For a small university research lab, that rate was untenable. It forced a different kind of redesign—one aimed at using cheaper models reliably rather than at keeping more capable models away from dangerous actions.

We wrote this report because both workflows taught us the same lesson by different routes, and because the practitioner literature on agentic systems does not yet cover it. Foundational work describes what capable agents can do [3], [7], but the move from exploratory prompting to a hardened, deterministic harness gets little direct treatment. Almost none of it describes that harness emerging *after* the fact, under pressure, in a lab too small to have built one up front. Small labs building these workflows will hit the same pressures we did. We want to show what ours looked like, what we did about them, and what we would tell another team working on similar budgets.

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. This is the accepted manuscript of an article accepted for publication in IEEE Software. The final version is available at https://doi.org/10.1109/MS.2026.3716868

2 CONTEXT

The Ateneo Business Insights Laboratory for Development (BUILD) is a university-based research laboratory. We run OpenClaw, an open-source AI agent platform, on a single Mac Mini connected to several LLM providers. The two workflows described in this report—a geospatial business classification pipeline (Case A) and a credit-card reimbursement workflow (Case B)—were developed between January and March 2026 by a small practitioner team that includes the authors. Evidence is drawn from the agent's memory files, runbook documents, git commit history, provider billing records, and practitioner recollection where artifacts were incomplete; where reconstruction was needed we used OpenClaw's local retrieval and Codex-assisted log parsing. Incidents where artifact evidence is partial are flagged explicitly in the text.

Data availability. The artifacts cited below by file name and line number contain institutional billing data and email recipients and cannot be released in full. We have instead prepared a *redacted* bundle (artifacts 01–07 with a README and a redaction log) that substantiates each load-bearing figure and decision in this report; it is available to reviewers via the accompanying response letter. Each cited path maps to a numbered artifact: `memory/cost-audit-2026-03-02.md` is `03`; `memory/psic_provenance.md` is `04`; `runbooks/geo-cleansing.md` is `05`; the $10.85 baseline (`memory/2026-02-15.md`) is `02`; the $292.82 billing week is `01`; and commits `4de6d17`/`18ac50c` are `06`. For the camera-ready version we will deposit the same redacted bundle on IEEE DataPort, which the call for papers encourages for reproducibility.

We present Case B first, because the email incident is the clearer illustration of risk-driven hardening. Case A, which hardened under cost pressure, follows in Section 4.

3 CASE B: THE REIMBURSEMENT WORKFLOW

3.1 How It Started

The workflow began as an informal convenience. The lead practitioner was accumulating credit-card charges from multiple AI providers that needed periodic reimbursement through the university's administrative process, and the agent was already monitoring Gmail for invoice attachments. It seemed natural to ask it to also log the charges, download the invoices, and eventually draft the reimbursement emails.

No architecture existed. The first prompt was something like *save credit card screenshots somewhere*—and the agent improvised storage, formats, and logging from whatever tools it had. Over the next weeks scope expanded organically: entries went into a notes app, then a spreadsheet, then both. Formatting was inconsistent. Columns drifted. We noticed, corrected, and moved on.

Looking back, this exploratory phase is where the email incident was already taking shape. We were asking the agent to take real actions in external systems—downloading attachments, writing to spreadsheets, drafting emails—with no explicit constraint on what it could do on its own. What it did was bounded only by what we happened to ask next, not by any standing rule about what it should never do.

3.2 The Email Incident and the Constraint Response

We do not have a full record of the email incident itself: the agent's logs were not structured to capture trigger prompts or outgoing-action payloads, and the incident was not tied to a single session we could replay. What we have is the constraint response—written into three separate memory files in the hours afterward, the same rule stated three times in slightly different words.

That the same constraint shows up in three places tells you how it was produced: one of us wrote it in the active workspace memory file, another in the cross-machine context synchronization file, a third in the agent's safeguards file. Nobody coordinated it. That is what happens when three people independently decide a rule is too important to miss.

The constraint was implemented immediately in a dedicated shell script (`scripts/draft_reimbursement_email.sh`) that enforced draft-only creation with hardcoded recipients and argument blocking designed to prevent any pathway to direct sending.

The redesign that followed went further. A later session with a high-capability model produced a structured billing module that replaced the ad-hoc approach with a SQLite-first pattern: a local `billing.db` as the authoritative record, with the spreadsheet as a downstream projection rather than the primary store. The memory log for that day (`memory/2026-03-02.md`) records the transition in one line: *"Architecture Upgrade: moved from Pure Sheets to SQLite-First (billing.db)."* The configuration file (`openclaw.json`, lines 318–326) now lists a primary model and a fallback chain that explicitly excludes the high-capability model used for architectural design—reserving that model for constraint work rather than routine execution.

3.3 The Drift We Kept Catching

A constraint architecture is not a finished workflow; it is a spine that has to keep resisting drift. Over the following weeks we logged five drift incidents in the billing workflow, each fixed by a runbook update, a script change, or both. Three are worth spelling out.

March 2: sheet alignment corruption. A cleanup script that used whitespace-based text parsing destroyed column alignment for rows whose currency prefix was missing. The memory note for that day is specific: *"Root cause: text-based whitespace parsing destroyed column alignment; Fix: Restored from Drive backup."* The runbook (`dollar-bills.md`) was hardened with three new guard rules the same morning, after a human noticed the shifted rows.

March 2: fallback drift. Later the same day, the workflow's fallback model—which the log identifies as Gemini—attempted a logging task without first reading the runbook, creating an OpenRouter entry for a charge that did not exist ("ghost entry") and adding fields the runbook prohibits. This was not a capability failure but a process one: the fallback path did not hydrate the runbook context before acting. The fix was a guard rule requiring runbook reading on every fallback invocation.

March 5: ghost row, missed invoices. A later incident produced a ghost row and a set of missed invoices that

should have been logged. The memory record captures the incident and its resolution (a watcher-based direct-write pattern that bypassed the intermediate logging step), but it does not record which model was actually executing when the incident occurred. We had been running credit-constrained that week, cycling through fallbacks under quota pressure, and the agent's routing state during those cascades was not captured in its own logs. We can describe what the workflow did, but not which model did it.

This observability gap is itself a finding, and the one we would most want other practitioners to take seriously. When a workflow cascades through several models under quota or credit pressure, which model actually ran at the moment of an incident is the forensic detail you need most—and we did not have it. The lesson: log fallback routing state explicitly, before you need it.

3.4 What It Cost to Run, After

Once the constraint architecture and the SQLite-first pattern were in place, routine operations stopped requiring high-capability models at all. A cost audit conducted on March 2 (`memory/cost-audit-2026-03-02.md`; redacted excerpt in the bundle, see Data availability) estimated the recurring execution cost of all scheduled billing jobs at $0.0016/month, excluding quota-based credits and excluding the one-time design-time sessions that produced the architecture (see Table 1). We do not compare this figure directly to Case A's costs: the two workflows process different workloads on different time bases, and a numeric ratio across them would not be meaningful. Qualitatively, both cases show the same shape—once the expensive design work is done once and committed to static artifacts, recurring execution runs on cheap models or scripts. Hardening did not make the agent cheaper by making it worse; it moved the expensive thinking to design time. That computation did not disappear—it was displaced to a one-time cost we report separately rather than amortize silently into the per-run figures.

4 Case A: The Geospatial Pipeline

4.1 What the Billing Week Looked Like

On February 2, 2026, our Anthropic account was charged three times in a single day: $40.28, $50.00, and $40.30—$130.58 before the day ended. By February 8, the week's total had reached $292.82 across seven charges.

The pipeline was OpenClaw's geospatial business classification system: an agent working through approximately 3,000 establishment records from a business permit dataset, resolving each to a barangay-level geocoordinate and a Philippine Standard Industrial Classification (PSIC) code. A *barangay* is the smallest administrative unit in the Philippines—a village or city district. The agent called a frontier model for every classification decision, not by design but because no constraint architecture existed to tell it otherwise; when the problem scope is large and the execution model is unconstrained, the cost follows directly. We treated the billing week not as an anomaly but as a design signal.

4.2 The Constraint Architecture

The redesign was done in a high-capability model session that produced a generation-2 geocoding architecture (commit `4de6d17`; redacted excerpts in the bundle, see Data availability). The resulting constraints addressed three failure modes already observed in generation 1: coordinates that fell outside Metro Manila's bounding box, stacked or averaged results that diluted the best individual geocode, and an inconsistent source hierarchy that allowed low-confidence sources to override high-confidence ones.

The generation-2 architecture enforced four structural constraints. **Bounding box enforcement:** every resolved coordinate is validated against a hard bounding box for the National Capital Region; results outside the box are rejected and rerouted, not averaged in. **Anti-stacking:** the pipeline resolves each record against a single source at a time, in priority order, stopping at the first source that meets the confidence threshold—averaging across geocoders was prohibited because it contaminated high-quality dictionary matches with weaker fallbacks. **Precision gating:** the source hierarchy is strict—hardcoded landmark dictionary first, then OpenStreetMap results at rank 26–30, then Mapbox, then street-level fallback—codified in `runbooks/geo-cleansing.md` lines 10–15. Here *rank* is OpenStreetMap's `place_rank`, an integer encoding how administratively specific a result is: lower values denote coarse areas (country, region), higher values finer ones (street, building), so a rank 26–30 cutoff admits results at least municipality-level in precision. **Barangay cross-check:** final coordinates are verified against the barangay in the source record; a result placing an establishment in the wrong boundary is flagged for audit regardless of confidence.

The hardcoded landmark dictionary (`scripts/geocode_bplo_dictionary.py`, commit `4de6d17`) was itself a hardening artifact: rather than asking the model to recall landmark coordinates at execution time, we had it generate a static dictionary once, verified the entries, and committed them [5]. Execution no longer required model reasoning for known landmarks.

PSIC classification followed parallel constraint logic. A cosine-similarity threshold of 0.35, documented in `memory/psic_provenance.md` lines 39–44, separated auto-accepted classifications (1,786 of 3,006 candidates) from records routed to human audit (1,220). The threshold was calibrated on the score distribution—we inspected where the distribution broke between confident and ambiguous results—rather than against a held-out ground-truth set. We therefore report throughput and cost for the auto-accept path, and we do *not* report a validated label-accuracy rate for it. The per-record classification output that would support such a measure was written to ephemeral storage during the exploratory phase and was not preserved—itself an instance of the unobservability this report warns against, the same failure mode as the unlogged email and the uncaptured March 5 routing state. Our claim for the auto-accept path is one of throughput and cost efficiency, not of validated precision.

4.3 The Drift We Kept Catching

Four drift incidents arose in Case A after the generation-2 architecture was deployed.

Out-of-boundary leakage. With the bounding box in place, a class of records still produced out-of-boundary coordinates: establishments whose permit addresses were themselves outside the NCR box—provincial branch addresses in the dataset. The constraint was correct; the assumption about input scope was not. We added a forced-refresh step to re-query the source address before geocoding—a scope correction, not a constraint relaxation.

Coordinate stacking. Despite the anti-stacking rule, the pipeline was still implicitly averaging across sources at barangay resolution when sources agreed on barangay but disagreed on street. Stack-break scripts and a landmark re-extraction step (commit `4de6d17`) enforced single-source resolution at every level.

Geocoder migration: rank enforcement. A targeted OSM sweep in early March revealed that rank-26 enforcement had been applied inconsistently: some records geocoded with OSM results at rank 28 had been accepted without triggering the fallback. The memory note for March 10 (`memory/2026-03-10.md`) records this as a surgical sweep and re-run. Affected records were reprocessed; the runbook was updated with an explicit rank validation step.

Classification label contamination. A cluster of real estate records was receiving generic-activity PSIC labels because the similarity model was matching on common words ("property," "management") rather than sector-specific terms. Contamination was caught during threshold calibration and resolved through threshold tightening to 0.35 and label-specific exclusion rules added to `memory/psic_provenance.md`.

4.4 What It Cost to Run, After

By mid-February, the pipeline had migrated to OpenRouter and was running at $10.85 per processing event—documented in `memory/2026-02-15.md` lines 28–29, with five consecutive daily entries confirming the post-hardening baseline. We define a *processing event* as one scheduled end-to-end run of the pipeline over the working record set for that day; the five mid-February entries are five such daily runs, one per day, which makes the per-event figure also a per-day figure for this workload. On that common daily basis it sits next to the $130.58 charged on February 2, the highest single pre-hardening day: a reduction of roughly ninety-two percent for the same pipeline, doing the same work, on the same dataset. Table 1 states the units and bases explicitly and reports the one-time design-time cost separately from recurring execution.

The reduction came from routing cheaper. The expensive thinking—threshold calibration, hierarchy design, boundary specification—had been done by a frontier model at design time and committed to static artifacts, a one-time cost reported in Table 1 rather than folded into the per-run figure.

TABLE 1
Cost on explicit, comparable bases. Pre- and post-hardening execution costs for Case A are stated on a per-day basis for the same pipeline and dataset. Design-time cost is the one-time frontier-model session that produced the constraints; it is reported separately rather than amortized into the per-run figures. Case B figures are recurring execution cost only and are not directly comparable to Case A (different workload and time base).

Item	Basis (unit)	Cost (USD)
Case A, pre-hardening peak	per day (Feb 2)	130.58
Case A, post-hardening	per day / per event	10.85
Case A design-time	one-time session	see note
Case B, post-hardening	recurring, per month	0.0016

Note: the Case A design-time session is the frontier-model work that produced the generation-2 constraints (commit `4de6d17`); its cost is part of the February Anthropic billing rather than a separately metered line item, so we report it as displaced one-time cost, not as zero. The post-hardening per-day and per-event figures coincide because the workload ran as one scheduled run per day.

5 What We Built, Named Plainly: A Reactive Agent Harness

Described in standard software-engineering terms, what we have been calling a "constraint architecture" is an execution harness: scaffolding wrapped around a nondeterministic component to make its behavior testable and safe. The mapping is close to one-to-one. The hardcoded landmark dictionary that replaces model reasoning for known landmarks is a *stub*. The bounding-box validation, the confidence thresholds, and the barangay cross-check are *assertions and oracles*. The fallback chain in `openclaw.json` and the draft-only script with argument blocking are the *driver and guardrails*. The growing runbook is a *regression checklist* that accumulates as new failure modes emerge. Stubs, drivers, oracles, and harnesses are long-standing testing constructs [8]; wrapping an unreliable component in a deterministic harness with validations and a growing set of checks is established practice, and we do not claim it as novel.

What we think is worth reporting is not the construct but how it came to exist. We did not design a harness up front and then run workflows inside it. It emerged *reactively*, one incident at a time, in a small lab that had not planned for one, shaped by cost and risk pressure rather than foresight. Three contributions follow. First, the reactive-emergence account itself: which constraints a small team actually reaches for, and in what order, when an unconstrained agent first does something expensive or irreversible. Second, and more specific, a class of failure modes particular to a *multi-model* harness. When execution cascades through several models under quota or credit pressure, which model actually acted becomes the piece of forensic information you need most, and it is almost never captured by default. The fallback-routing lesson below survives precisely because it is a harness lesson of this multi-model kind: which backend acted has to be recorded, and standard single-component harness practice does not tell you to record it.

Third, a point about who these platforms now put at

the controls. Standard harness practice assumes an engineer who knows a harness is needed and has the vocabulary—stub, oracle, regression check—to build one. An agent platform needs no such background to run: it lets a workflow expand into irreversible or expensive action before anyone decides hardening is needed. We recognized the need, but we put that down to a software-engineering background the operators these platforms increasingly serve may not have. When an irreversible action is involved, a stochastic skill layer cannot *guarantee* the constraint the situation needs—a draft-only rule that holds every time, not most of the time—and the platform gives no signal about where that line falls. We offer this as an observation from a single lab with one technical operator, not a measured finding; it is the part of our experience we would most want tested in settings less technical than ours.

The sequence we now follow is deliberate, though we describe it as things we do, not a framework we propose [1]. A new workflow begins with open-ended, deliberately expensive high-capability sessions that surface edge cases and draft constraint candidates. After that exploratory phase we audit what the agent has been doing on its own and where behavior diverged from intent—the audit that drove both the SQLite-first redesign and the generation-2 geocoding constraints. We then ask the best available model for an exhaustive constraint specification (valid inputs, permitted outputs, fallback triggers, what fallback models may and may not do), and that spec becomes the runbook. Where the spec is complete, we replace the model with a script: the landmark dictionary and the draft-only shell script cost nothing per invocation and cannot drift, and deterministic execution at this layer is a production-readiness requirement, not a performance preference [4]. Remaining execution runs on cheaper models or scripts inside the constraints [6], with the expensive model confined to design time—the mechanism behind both the $0.0016/month billing baseline and the $10.85/event geocoding baseline. Throughout, the runbook keeps growing: the real constraint space is larger than anything specifiable in advance, and each incident surfaces a case the original spec missed. A runbook that is not growing is either a finished pipeline or an unmonitored one.

One acknowledged limitation: our commit discipline during exploratory phases was inconsistent. The agent committed opportunistically rather than on milestones, so we cite commits `18ac50c` and `4de6d17` as markers of hardening bursts rather than a continuous development trail. This affects how artifact evidence aligns with narrative sequence, and is worth naming for practitioners building on similar agent platforms.

6 What We Would Tell Other Teams

Four things we would do earlier, based on what we got wrong.

Design for the irreversible action before you deploy it. The email incident happened because we had not thought explicitly about which actions were irreversible: auto-sending is, drafting is not. For any action your agent can take in an external system—send, delete, post, submit—decide in advance whether it may act autonomously, and encode that decision as a structural constraint, not a prompt instruction.

Log fallback routing state explicitly. When your workflow cascades through several models under quota or credit pressure, which model actually ran at a given step is the thing you most need afterward, and it is almost never captured by default. Add explicit routing-state logging before your first incident. The March 5 incident is permanently unresolvable because we did not.

Calibrate thresholds deliberately, and keep the evidence that justified them. We set the PSIC threshold of 0.35 by looking at the score distribution rather than guessing, which was the right call. But we did not keep the per-record output that would let anyone check the resulting label quality later. The calibration was defensible at the time and impossible to verify afterward, because the output was transient. The lesson runs both ways: calibrate against data, and make that data durable. A threshold you cannot audit later is a threshold you cannot defend.

Treat model migration as a drift event. When a fallback model changes—by your own routing update or a provider change—run your test cases before resuming. The March 2 incident was a process failure: a capable model received a task without the runbook context it needed.

7 Conclusion

Two workflows, two failure modes, one pattern underneath: unconstrained AI action surfaces the constraints you should have built in advance. In both cases scope grew faster than the constraint architecture, and the fix was the same. Use a capable model to design explicit limits, commit them to static artifacts, and route execution through cheaper models or scripts inside those limits. Our lab is smaller than the environments most agentic-systems research describes. The pressures—cost, risk, observability gaps in fallback chains—are not.

The gaps in our own record—an unlogged email, an unrecoverable routing state, a transient classification output—all point to the same requirement: the artifacts that justify a workflow's decisions have to be made durable as the workflow hardens, not left in ephemeral storage. Designing agent harnesses so that decision-supporting evidence is preserved by default, rather than reconstructed after an incident, is where we intend to go next.

References

[1] J. B. R. Ilagan and J. R. S. Ilagan, "A Constraint-Driven Conversational Architecture for Staged Task Interaction Using Large Language Models," *Int. J. Adv. Comput. Sci. Appl.*, 2026.

[2] J. B. R. Ilagan, G. A. L. Uy, and J. R. S. Ilagan, "From Prompt to Pipeline: Why and How AI-Assisted Workflows Harden into Deterministic Systems," SSRN preprint, Mar. 2026. [Online]. Available: https://ssrn.com/abstract=6358278

[3] S. Yao *et al.*, "ReAct: Synergizing Reasoning and Acting in Language Models," in *Proc. 11th Int. Conf. Learning Representations (ICLR)*, 2023. [Online]. Available: https://openreview.net/forum?id=WE_vluYUL-X

[4] P. Drammeh, "Multi-Agent LLM Orchestration Achieves Deterministic, High-Quality Decision Support for Incident Response," arXiv:2511.15755, Jan. 2026. [Online]. Available: https://arxiv.org/abs/2511.15755

[5] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code," arXiv:2107.03374 [cs.LG], Jul. 2021. [Online]. Available: https://arxiv.org/abs/2107.03374
[6] I. Ong *et al.*, "RouteLLM: Learning to Route LLMs with Preference Data," in *Proc. 13th Int. Conf. Learning Representations (ICLR)*, 2025. [Online]. Available: https://arxiv.org/abs/2406.18665
[7] C. Bandi *et al.*, "The Rise of Agentic AI: A Review of Definitions, Frameworks, Architectures, Applications, Evaluation Metrics, and Challenges," *Future Internet*, vol. 17, no. 9, Art. no. 404, 2025.
[8] G. J. Myers, *The Art of Software Testing*. Hoboken, NJ, USA: Wiley, 2006.

ACKNOWLEDGMENT

The authors used Anthropic's Claude (Claude Sonnet 4.6 for the initial preparation and Claude Opus 4.8 for the revision) extensively in the preparation of this manuscript. Specific uses included: drafting and iterative revision of all prose sections, LaTeX typesetting and formatting, verification of reference metadata against public sources, cropping and preparation of author biography photographs, and general editorial assistance throughout the writing process. All technical claims, evidence citations, artifact references, cost figures, and incident descriptions reflect the authors' own documented workflows and were verified by the authors. The underlying work—the workflows, incidents, constraints, and runbooks described—was conducted independently of and prior to the use of any AI writing assistance.



Joseph Benjamin R. Ilagan is an Assistant Professor and Director of the Ateneo Business Insights Laboratory for Development (BUILD) at the John Gokongwei School of Management, Ateneo de Manila University. He recently defended his Ph.D. dissertation in Computer Science. His research spans AI infrastructure, educational technology, and applied civic systems. He has 17 years of prior industry experience in software development.



Gabrielle Anne L. Uy is a Lecturer in the Quantitative Methods and Information Technology Department, John Gokongwei School of Management, Ateneo de Manila University, where she covers advanced courses in the Information Technology Entrepreneurship program. She is also a researcher at the Ateneo Business Insights Laboratory for Development (BUILD).



Jose Ramon S. Ilagan is a Lecturer in the Quantitative Methods and Information Technology Department, John Gokongwei School of Management, Ateneo de Manila University. He is also the Resident Tech Lead of the Ateneo Business Insights Laboratory for Development (BUILD).