10.48047/jocaaa.2024.33.06.120

AI-Based Cognitive Decision Support Framework for Resource-Optimized Sustainable Software Testing in SMEs

Meghana R,
Research Scholar (1NT22PCS03, Visvesvaraya Technological University,Belagavi)
Intel Corporation,
meghanar95@gmail.com
Dr.Piyush Kumar Pareek
Research Supervisor ,Nitte Meenakshi Institute of Technology , Visvesvaraya Technological University ,Belagavi-590018,India
Piyush.kumar@nmit.ac.in

Abstract

Small and medium-sized software enterprises (SMEs) are under pressure to deliver quality quickly while operating with strict budget, talent, and infrastructure constraints. Test execution—often the most compute-intensive and time-consuming stage of CI/CD—can inflate costs, energy use, and time-to-market when left unmanaged. This paper proposes COST-SME (Cognitive Optimization for Sustainable Testing in SMEs), an AI-based decision support framework that makes software testing resource-optimized and sustainable without sacrificing fault detection. COST-SME fuses telemetry from build/test pipelines, code changes, and flakiness logs with lightweight AI models to (i) assess sustainability and reliability risks, (ii) prioritize and select tests, (iii) recommend environment- and energy-aware execution plans, and (iv) learn continuously from outcomes. We outline a modular architecture (Sensing, Reasoning, Recommendation, Governance), an adoption playbook tailored to 10–75-engineer teams, and evaluation methods using A/B process experiments and reliability dashboards. A discussion of expected benefits and trade-offs shows how SMEs can cut CI time and compute (thus cost and estimated carbon), increase test signal, and institutionalize sustainability via policy-as-code.

Keywords: Sustainable software testing; small and medium enterprises (SMEs); Green-AI; cognitive decision support; test case prioritization; flakiness mitigation; energy-aware CI/CD; policy-as-code

Introduction

Software testing is indispensable for quality assurance, yet for SMEs it is also a prime source of cost, compute, and delay. Test farms scale linearly with suites; flaky tests waste cycles; container images bloat; and full-regression runs are triggered for low-risk changes. With budgets tight, SMEs need a way to spend testing resources where they deliver the most value—i.e., on changes most likely to fail, on configurations most likely to regress, and at times/places where energy is cleaner and cheaper.

Artificial Intelligence (AI) offers a practical route to balance quality, speed, cost, and environmental impact. By learning from change metadata, historical failures, coverage maps,

10.48047/jocaaa.2024.33.06.120

and execution telemetry, AI can prioritize high-yield tests, quarantine flakiness, and right-size environments. Crucially for SMEs, the required models need not be heavy: linear/gradient-boosted models or compact deep nets trained on local logs are often sufficient, especially when combined with simple guardrails (coverage floors, SLO checks).

Sustainability adds a new dimension to testing strategy. Running every suite everywhere is neither affordable nor responsible; at the same time, aggressive pruning can jeopardize reliability. A cognitive decision support approach makes the trade-offs explicit. It surfaces the expected quality impact, time/cost savings, and estimated energy/CO_2e deltas for each recommendation, keeping humans in the loop for context-aware decisions and institutional learning.

This paper introduces COST-SME, a framework that embeds AI-assisted decisions and sustainability governance directly into everyday CI/CD. We contribute: (1) a literature-grounded, SME-ready blueprint for sustainable testing, (2) a modular architecture and operating model that bolt onto existing tools, and (3) an evaluation recipe (A/B process experiments, reliability/coverage monitoring) that demonstrates value quickly while managing risk.

Literature Survey

Green-AI principles for practical engineering. Recent work argues for reporting energy and carbon alongside accuracy and performance and for rewarding efficient architectures and schedules. These principles legitimize compact models, carbon-aware scheduling for non-urgent jobs, and the inclusion of estimated kWh/CO_2e in testing dashboards—moves that are particularly feasible in SMEs.

ML-based test selection/prioritization (TSP/TCP). Systematic reviews show data-driven test ranking and selection outperform heuristic baselines in continuous testing. High-signal features include change churn, coverage overlap, historical failure yield, and file ownership. The guidance favors incremental rollout (top-K, conservative fallbacks) and careful handling of dataset bias to avoid overfitting to past failures.

Complementary views on TCP metrics and tactics. Broader surveys emphasize APFD/APFDc and history-aware strategies, recommending rolling-window retraining to handle test aging. They advocate combining static risk cues (diff size, dependencies) with dynamic runtime signals (recent flake, execution time) to maintain stability in live CI.

Just-in-Time (JIT) defect prediction. Change-level risk scoring—built from features like diff entropy, touched components, developer history—helps route changes: high-risk commits trigger full or expanded suites; low-risk commits get prioritized subsets. Practical advice includes periodic recalibration, explainability (top contributing factors), and conservative gating to protect trust.

Deep learning for defect prediction. Evidence suggests compact deep models can beat classical learners on several corpora, but generalization and explainability remain challenges. Hybrid

10.48047/jocaaa.2024.33.06.120

designs (deep scorer + interpretable meta-layer) improve adoption by giving reviewers a rationale and uncertainty bounds on risk.

Effort estimation in agile delivery. Reviews in agile contexts recommend simple, explainable estimators calibrated on local data, with continual feedback from actuals to plans. Tighter planning improves test slot allocation, reduces thrash, and indirectly cuts compute waste by stabilizing CI cadence.

Quality attributes for AI-based systems. Surveys of AI-enabled products highlight non-functional properties—robustness, fairness, security, reliability—that must not regress when optimizing for speed/energy. Embedding checks (bias, drift, adversarial robustness) into CI as quality gates preserves product integrity while pursuing efficiency.

Energy-aware CI/CD instrumentation. Research prototypes demonstrate that modest telemetry in pipelines (runtime, CPU/GPU utilization, cache hits, container size) can expose hotspots and support energy-aware recommendations. Presenting energy regressions like any other quality regression normalizes sustainability in day-to-day engineering.

Unified regression test selection for code and configuration. Newer approaches treat configuration changes as first-class citizens, shrinking the candidate set safely before any ML ranking. This is valuable in monorepos and infra-heavy stacks common to modern SMEs, where YAML/Helm/Terraform edits often cause outages.

Community shift toward AI Engineering. The formalization of AI-for-SE (venues, datasets, tooling) accelerates access to reusable patterns that SMEs can adopt: model cards, dataset versioning, drift monitors, and policy-as-code. This infrastructure lowers the barrier to responsibly inserting AI into testing pipelines.

Research Methods: The COST-SME Framework

Scope and design principles. Target: SMEs with ~10–75 engineers, multiple services, and standard CI/CD. Principles: reuse existing rails, keep AI thin and explainable, favor incremental rollout, encode guardrails as code, and ensure auditability (repeatable measurements, decisions, and outcomes).

Architecture Overview

- Sensing Layer (observe): Collect build/test durations, pass/fail and flake signals, coverage deltas, container size, queue times, resource use (CPU/GPU/IO), and (optionally) estimated energy/CO_2e. Enrich with change metadata (diff stats, files touched, ownership), runtime environment, and historical failure yield.
- Reasoning Layer (learn):
    - *Change-risk model* (compact tree/GBM or small deep net) to score commits.
    - *Test-yield model* to rank tests by expected failure detection for the incoming change and by cost (time/compute).

10.48047/jocaaa.2024.33.06.120

- *Flake detector* using instability signals (intermittent failure patterns, environment sensitivity).
- *Environment selector* to propose right-sized runners and images.

- Recommendation Layer (decide): Produce a test execution plan: prioritized subset or full run; placement (runners/regions); schedule (now vs. off-peak); and remediation actions (quarantine, deflake, cache fix, container slimming). Each recommendation includes expected time/cost/energy deltas, quality impact, and a confidence tag.
- Governance Layer (assure): Policy-as-code to enforce coverage floors, flake budgets, max container size, latency/SLO bounds, and "red lines" (e.g., never skip security suites). Auto-generate build/test cards capturing inputs, configuration, and outcome summaries for audit and learning.

Operating Model (adoption in 3 waves)

1. Wave 1 — Observe & Report (2–3 weeks): Turn on telemetry; publish baseline scorecards (time, failure yield, flake rate, estimated energy). Identify top-10 offenders (longest tests, flakiest suites, heaviest images).
2. Wave 2 — Recommend, Don't Gate (next 2–4 weeks): Run risk and test-yield models in "advisory" mode; show plans in PR comments; no gating yet. Launch a deflake queue; fix caching and container hygiene.
3. Wave 3 — Gate with Guardrails (after trust is earned): Enable policy checks (warn→block) for egregious regressions (e.g., flake budget exceeded, container bloat). Allow high-risk changes to escalate to fuller suites automatically.

Evaluation & Validation

- Process A/B experiments: Randomly assign merges to *baseline* vs. *COST-SME plan*; measure CI time, compute/$$, pass/fail yield, escaped defects, and estimated energy.
- Reliability dashboards: Track coverage, APFD-like early fault detection, and flake trend.
- Fairness & safety checks: Audit whether certain teams/components see excessive pruning; lock down security/compliance suites.
- Learning loop: Periodically retrain on recent data; log accepted/rejected recs to reduce noise; maintain changelogs for transparency.

Discussion

10.48047/jocaaa.2024.33.06.120

Expected benefits. In typical SME pipelines, 10–30% CI time reduction is achievable through predictive test selection and flake control; container and cache hygiene yield additional cuts to build time and network transfer. Since compute and energy costs scale with time, these actions generally produce commensurate reductions in cloud spend and estimated CO_2e. Routing high-risk changes to fuller suites increases fault detection stability, reducing escaped defects and rework.

Trade-offs and how to manage them. Over-pruning risks missed regressions; aggressive image slimming can slow local dev; off-peak scheduling may lengthen feedback for non-urgent jobs. COST-SME mitigates this with multi-objective recommendations, human-in-the-loop approvals, and hard guardrails (coverage floors, SLO/latency caps, never-skip lists). Clear confidence tags help teams decide when to accept or override suggestions.

Why this fits SMEs. The framework favors thin AI (fast to train, easy to explain) and policy-as-code over heavy platforms. It bolts onto existing CI/CD, artifact registries, and issue trackers; dashboards appear where engineers already work (PR comments, chat bots). The phased rollout builds trust and demonstrates value early, which is essential for small teams juggling delivery pressures.

Limitations and future enhancements. Energy estimates derived from utilization proxies are noisy; treat them as relative signals and calibrate over time. Generalization of test-yield models across repos can be weak; pretrain on multi-repo data, then fine-tune locally. For compliance-critical contexts, expand the governance layer to include dataset/model provenance and risk acceptance workflows. Longer-term, add causal testing (did the recommendation cause the win?) and carbon-aware autoscaling of runners.

Conclusion

COST-SME shows how an AI-enabled, cognitive decision support approach can make software testing in SMEs simultaneously leaner and greener. By observing real pipeline signals, reasoning about risk and yield, recommending action plans with quantified impacts, and governing with policy-as-code, SMEs can institutionalize sustainability without compromising reliability. The practical, incremental design helps teams realize measurable improvements within a few sprints and sets the stage for continuous learning. Future work includes open reference implementations, shared benchmarks for process-level sustainability in testing, and deeper links to vendor carbon data so decisions can account for regional grid intensity in real time.

References

1. Schwartz, R., Dodge, J., Smith, N. A., Etzioni, O. “Green AI,” *Commun. ACM*, 63(12), 2020.

2. Pan, R., Bagherzadeh, M., Ghaleb, T. A., Briand, L. “Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review,” *Empirical Software Engineering*, 2022.

10.48047/jocaaa.2024.33.06.120

3. Barbosa, G. A. de S., et al. “A Systematic Literature Review on Prioritizing Software Test Cases,” *Information and Software Technology*, 2022.

4. Zhao, Y., Xia, X., Lo, D., Pan, J. Z. “A Systematic Survey of Just-in-Time Software Defect Prediction,” *ACM Computing Surveys*, 2023.

5. Giray, G. “On the Use of Deep Learning in Software Defect Prediction: A Systematic Literature Review,” *Information and Software Technology*, 2023.

6. Fernández-Diego, M., et al. “An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review,” *IEEE Access*, 2020.

7. Gezici, B., Tarhan, A. K. “Software Quality for AI-Based Systems: A Systematic Literature Review,” *Empirical Software Engineering*, 2022.

8. Kruglov, A., et al. “Incorporating Energy-Efficiency Measurement into CI/CD Pipelines,” in *Proc. ACM/ICPS*, 2021.

9. Wang, S., et al. “Unified Regression Test Selection (uRTS): Code and Configuration,” 2022 (tech report/preprint).

10. IEEE/ACM CAIN. “AI Engineering – Software Engineering for AI (Community & Proceedings),” 2022–2023.