10.48047/jocaaa.2024.33.05.28

An AI-Enabled Framework for Software Process Optimization in Small Software Firms

Meeravali Shaik,
Research Scholar (USN: 1NT20PCS08)
Nitte Meenakshi Institute of Technology,
Visvesvaraya Technological University,Belagavi-590018,India
meeravali043106@gmail.com

Piyush Kumar Pareek
Research Supervisor ,
Nitte Meenakshi Institute of Technology ,
Visvesvaraya Technological University ,Belagavai-590018,India

Abstract

Small software firms (SSFs) operate with lean budgets, heterogeneous stacks, and tight delivery timelines—yet they face the same expectations for quality, speed, and sustainability as larger organizations. This paper proposes AIPSO-SSF, an AI-enabled framework that helps SSFs measure, optimize, and govern software processes without heavy tooling or specialist teams. The framework integrates (i) a baseline and observability layer that instruments build/test/deploy workflows and lightweight product metrics; (ii) an AI decision layer that recommends changes across test selection, build caching, code review assistance, and model-efficiency tactics; and (iii) a governance plane that encodes guardrails as policy-as-code and closes the loop with experiment-driven validation. We synthesize recent evidence on Green-AI, machine-learning-based test case prioritization, just-in-time defect prediction, effort estimation, and energy-aware CI/CD. We then describe processes, roles, and architectural patterns that fit 5–50-person teams. The discussion outlines expected gains (reduced CI time, fewer regressions, lower compute cost/energy) and trade-offs, with a practical rollout plan. The contribution is a coherent, low-overhead approach that turns scattered AI techniques into a repeatable capability for continuous process improvement in SSFs.

Keywords: software process optimization; small software firms; Green-AI; CI/CD; test case prioritization; defect prediction; effort estimation; policy-as-code

Introduction

Software development has absorbed increasingly compute-intensive practices—continuous testing, containerized builds, and ML-backed features—while delivery cadences have accelerated. For small software firms (SSFs), the result is a persistent tension between speed and discipline: every minute in CI, every flaky test, and every oversized container directly affects developer throughput and cash burn. AI presents a practical way to navigate this tension by learning from routine engineering telemetry—build logs, test outcomes, change metadata—and turning it into targeted, low-risk process interventions.

10.48047/jocaaa.2024.33.05.28

The optimization surface in SSFs is broad but familiar: prioritize the tests most likely to reveal faults; quarantine and deflake brittle tests; reduce container bloat and rebuilds; route non-urgent jobs to greener/cheaper windows; and use predictive models to focus code review and pre-merge checks. The last five years have produced solid building blocks: machine-learning-based test case selection/prioritization (TCP/TSP), just-in-time (JIT) defect prediction, learning-based effort estimation, and emerging practices for energy-aware CI/CD. These are individually useful; the challenge is operationalizing them together—safely, incrementally, and with clear guardrails—for a team of a few dozen engineers. (SpringerLink)

A second motivation is sustainability. "Green AI" argues for efficiency and transparency in computational work: report energy and carbon alongside accuracy and latency; prefer techniques that deliver comparable value at lower cost and footprint. For process optimization, that ethos translates into measuring the energy/time impact of pipeline changes, preferring compact models for AI services, and scheduling background workloads in low-carbon windows. SSFs can embrace this without new platforms by piggybacking on existing CI and observability. (ACM Digital Library)

Finally, improvement must be governed and learnable. AIPSO-SSF wraps recommendations with policy-as-code, canary rollouts, and experiment logging—so teams see the causal impact of changes and avoid single-metric myopia (e.g., faster builds but lower coverage). The result is an everyday capability rather than a one-off "optimization sprint."

Literature Survey

Green-AI fundamentals (2020). Schwartz et al. articulate *Green AI*, advocating efficiency and carbon transparency in AI work. The article frames practical principles—report energy/carbon, prefer efficient architectures, and evaluate cost–benefit—that map naturally to process optimization (e.g., measuring CI energy, preferring compact inference models). They also encourage comparing models by "price of accuracy," nudging teams to justify incremental accuracy against computational cost. For small firms, this legitimizes lightweight models and carbon-aware scheduling as strategic choices rather than compromises. The paper's call for standardized reporting further supports adding energy/CO_2 fields to build and model cards. *(ACM Digital Library)*

ML-based test selection/prioritization (2021–2022). Pan et al. conduct a systematic review of ML-based TSP/TCP, showing data-driven methods outperform heuristics under CI constraints and flagging threats to reproducibility and dataset bias—useful warnings for small orgs. They highlight features like test history, coverage, and change metadata as high-signal inputs for ranking. The review suggests incremental adoption (start with top-K selection and conservative fallbacks) to avoid missed defects. It also recommends public benchmarks and clearer reporting to make results transferable across stacks and languages. *(Orbilu)*

Complementary TCP review (2022). Barbosa et al. survey prioritization techniques and metrics (APFD variants), emphasizing incremental and history-aware approaches in continuous testing—motivating "learning-on-logs" designs in AIPSO-SSF. They note that aging effects (test usefulness shifts over time) require models that refresh on rolling windows. The paper

10.48047/jocaaa.2024.33.05.28

stresses balancing speedups with guardrails like minimum coverage and flaky-test quarantine. It proposes combining static risk cues with dynamic runtime signals for more stable gains. *(ScienceDirect)*

JIT defect prediction – survey (2023). Zhao et al. synthesize advances in Just-in-Time defect prediction (change-level risk scoring), clarifying features (diff size, entropy, developer history) and modern learners, and discussing CI deployment—direct input to pre-merge gates. They caution that labeling noise and domain drift can erode precision, advocating periodic recalibration and explainability (top features per score). Practical guidance includes routing high-risk changes through deeper test suites while leaving low-risk paths fast. This selective intensification preserves throughput without sacrificing quality. *(ACM Digital Library)*

Deep learning for defect prediction – SLR (2023). Giray's review finds deep models often outperform classic learners across datasets, but generalization and explainability remain challenges—supporting a policy of pairing risk scores with rationale and conservative gating. The review recommends hybrid pipelines (deep model + interpretable meta-layer) to aid reviewer trust. It also underscores dataset curation and leakage prevention as preconditions for credible results. For SSFs, the takeaway is "simple first, prove value, then scale." *(ScienceDirect)*

Effort estimation in agile – SLR (2020). Fernández-Diego et al. review agile effort estimation, noting progress but persistent dataset/feature issues. They recommend combining simple, explainable estimators with local calibration to handle team-specific norms. Continuous feedback loops—closing the gap between planned and actual—improve estimates over time and stabilize sprints. Embedding uncertainty bands in estimates helps product owners manage scope risk transparently. *(SciELO México)*

Software quality for AI-based systems – SLR (2022). Gezici & Tarhan catalog quality attributes (robustness, fairness, security, reliability) and practices for AI-based software, reinforcing that process changes must preserve product qualities—not just throughput—via policy-as-code guardrails. They argue for traceable datasets, versioned models, and test oracles tailored to ML failure modes. The survey suggests integrating non-functional checks (bias, drift, robustness) into CI like any other quality gate. This is particularly actionable for small teams adopting AI both in pipeline automation and product features. *(ACM Digital Library)*

Energy-aware CI/CD (2021). Kruglov et al. present a method/tool linking development-stage analysis to energy outcomes, arguing for pipeline-level telemetry and recommendations—an early blueprint for energy/time scorecards. They demonstrate that modest instrumentation can expose "hot spots" (e.g., a handful of tests dominating compute). The paper recommends surfacing energy regressions alongside performance regressions to normalize sustainability as a quality attribute. For SSFs, this validates starting with estimates, then refining measurement as maturity grows. *(ACM Digital Library)*

Regression test selection beyond code (2022). Wang et al. introduce uRTS, unifying selection for code and configuration tests with safety guarantees while reducing end-to-end time—useful for monorepos and infra-heavy stacks. The approach treats configuration changes as first-class

10.48047/jocaaa.2024.33.05.28

citizens, catching failures that traditional code-only selection misses. Their safety constraints and fallback logic make phased rollout practical in small teams. It also complements ML-based prioritization by shrinking the candidate set before ranking. *(mir.cs.illinois.edu)*

SE for AI – community shift (2022). The IEEE/ACM CAIN venue signals institutionalization at the SE↔AI boundary—relevant as SSFs adopt AI inside pipelines (test selection, JIT prediction) and in products (model efficiency). The community emphasizes tooling, governance, and education, not just algorithms. This shift provides venues, datasets, and patterns that small firms can reuse rather than build from scratch. It also accelerates convergence on shared terminology and metrics for responsible, efficient AI engineering. *(IEEE Computer Society)*

Research Methods: The AIPSO-SSF Framework

Scope and principles. Designed for 5–50-engineer teams shipping web/mobile services and occasional ML features. Principles: reuse existing rails (CI/CD, issue tracker, observability), incremental adoption, human-in-the-loop decisions, policy-as-code guardrails, and auditable outcomes.

Phase 1 — Baseline & Observability.
Inventory. Map repositories, pipelines, and critical paths (commit→merge→build→test→deploy).
Telemetry. Add low-overhead collectors for build/test durations, cache hit rates, container size, CPU/GPU/IO, and—where feasible—energy proxies (consistent method beats perfect accuracy).
Scorecards. Define a balanced set of indicators: (i) environmental/compute (kWh proxies, $kgCO_2e$, container bytes moved), (ii) operational (lead time, failure rate, MTTR, flake rate), (iii) product (p95 latency, error budget), and (iv) human factors (after-hours deploys, review wait time).
Hotspot analysis. Run 2–4 weeks to expose top offenders (e.g., a handful of tests dominating CPU time, redundant container layers, long queues before slow jobs).

Phase 2 — AI Decision Layer.
Test optimization. Train TSP/TCP models on change metadata and test history to rank tests; quarantine flaky tests; enforce coverage floors. Findings from recent studies suggest pretraining across projects and fine-tuning locally improves early-cycle performance. (Orbilu)

JIT risk scoring. Apply JIT defect prediction to flag risky changes and adjust pipeline strategy (e.g., run full suite if risk>τ; otherwise, prioritized subset). Provide explanations (top contributing features) to maintain reviewer trust. (ACM Digital Library)
Build/container hygiene. Recommend multi-stage Dockerfiles, base-image swaps, dependency pruning, and remote cache strategies to cut rebuilds and cold starts.
Effort estimation support. Provide calibrated story-point/size suggestions with uncertainty bands to improve planning accuracy and reduce thrash. (SciELO México)
Energy-aware scheduling. Shift non-urgent jobs (nightly builds, batch training) to greener/cheaper windows; surface the expected impact in scorecards. (ACM Digital Library)

10.48047/jocaaa.2024.33.05.28

Phase 3 — Governance & Learning.

Policy-as-code. Encode guardrails (max container size, flake budgets, coverage floors, SLO bounds) as CI rules; start in "warn" mode; graduate to "gate" once stable. *Experimentation.* Treat improvements as A/B process experiments (e.g., 50% of merges get predictive test selection); analyze time/energy and reliability effects; stop early when benefits are clear.

Provenance & auditing. Auto-generate "build cards" with inputs, parameters, runtime, and computed impacts; store with artifacts.

Feedback loops. Review scorecards monthly; rotate a lightweight "sustainability/process champion" role; capture accepted/rejected recommendations to debias the models.

Discussion

Expected impact. Based on the literature and field reports, SSFs can expect 10–30% cuts in CI time from ML-based test prioritization/selection, noticeable reductions in container sizes and rebuild frequency through hygiene recommendations, and fewer escaped defects via JIT-risk-gated pipelines. Energy proxies typically decline with time savings, aligning AIPSO-SSF with Green-AI goals. (Orbilu)

Trade-offs and safeguards. Aggressive test pruning risks missed regressions; JIT risk scores can create false positives that slow merges; container minimization can hinder local dev if over-optimized. AIPSO-SSF mitigates these with multi-objective recommendations, conservative defaults (never compromise coverage/SLOs), and human-in-the-loop approvals.

Adoption risks. The most common failure modes are noisy recommendations and dashboard sprawl. Constrain scope initially to tests and containers, integrate outputs into existing PR/CI comments, and measure "blast radius" (e.g., change failure rate) alongside speed. Energy estimates are imperfect—use them comparatively with a consistent method rather than as absolutes. (ACM Digital Library)

Fit for small teams. The framework emphasizes *thin AI*: simple models trained on local logs; policy-as-code instead of heavyweight platforms; and progressive rollout. This matches SSF constraints while capturing much of the value documented in recent surveys and systematic reviews. (ScienceDirect)

Conclusion

AIPSO-SSF translates a decade of AI-for-SE advances into a coherent, low-overhead capability for small software firms. By combining baseline observability, targeted AI recommendations, and policy-backed governance, SSFs can shorten feedback loops, reduce compute/energy waste, and improve reliability without sacrificing speed. Future work includes open reference implementations, benchmark datasets for process-level evaluation (e.g., multi-project TSP/JIT corpora), and deeper integrations with vendor emissions data and cost allocation—so teams can make optimization choices with full operational and environmental context.

References

10.48047/jocaaa.2024.33.05.28

[1] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, “Green AI,” *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020. (ACM Digital Library)

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

[3] G. A. de S. Barbosa, et al., “A systematic literature review on prioritizing software test cases,” *Information and Software Technology*, 2022. (ScienceDirect)

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

[5] G. Giray, “On the use of deep learning in software defect prediction: A systematic literature review,” *Information and Software Technology*, 2023. (ScienceDirect)

[6] M. Fernández-Diego, E. Méndez, F. González-Ladrón-de-Guevara, S. Abrahão, and E. Insfran, “An update on effort estimation in agile software development: A systematic literature review,” *IEEE Access*, vol. 8, pp. 166768–166800, 2020. (SciELO México)

[7] B. Gezici and A. K. Tarhan, “Systematic literature review on software quality for AI-based software,” *Empirical Software Engineering*, 2022. (ACM Digital Library)

[8] A. Kruglov, et al., “Incorporating energy efficiency measurement into CI/CD pipeline,” in *Proc. ACM/ICPS*, 2021. (ACM Digital Library)

[9] S. Wang, et al., “Test Selection for Unified Regression Testing,” 2022 (preprint/tech report). (mir.cs.illinois.edu)

[10] IEEE/ACM, “1st International Conference on AI Engineering – Software Engineering for AI (CAIN),” 2022 (community/venue reference). (IEEE Computer Society)