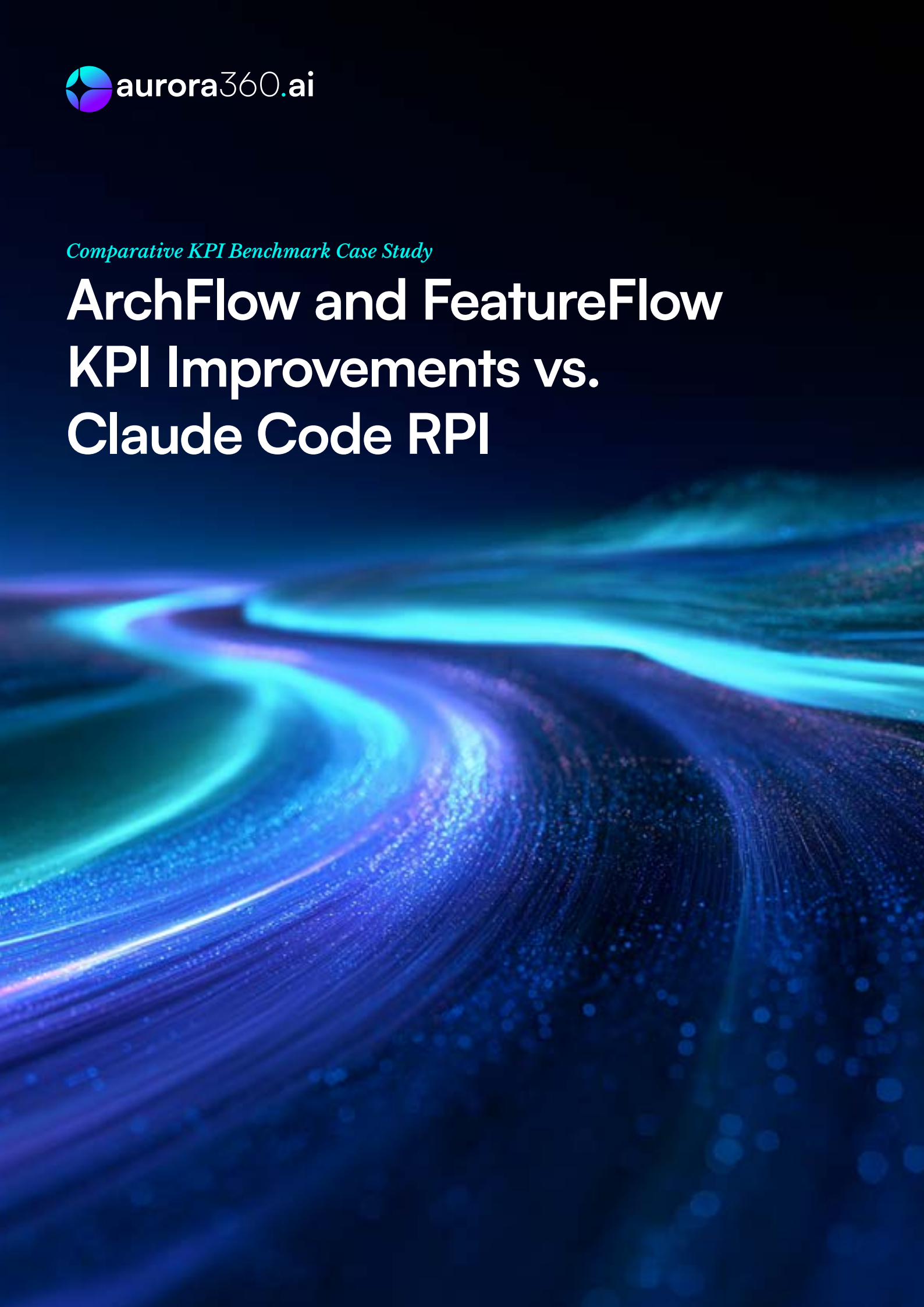
aurora360.ai

Comparative KPI Benchmark Case Study

ArchFlow and FeatureFlow KPI Improvements vs. Claude Code RPI



Technology

Aurora360.ai ArchFlow and FeatureFlow, Claude Code, MCP servers, AI coding agents, RPI (Research, Plan, Implement), ADRs (Architecture Decision Records), quality gates, deterministic validators, traceability artifacts.

Industry

AI-native software engineering, enterprise software delivery, brownfield modernization, architecture governance, and feature delivery in existing codebases.

Introduction

AI coding agents such as Claude Code can materially accelerate software development, but their value depends on the delivery system around them. Used only with an ad hoc Research - Plan - Implement (RPI) prompting pattern, the agent may produce useful code quickly, while still leaving gaps in architecture governance, traceability, verification, and durable knowledge capture.

This case study evaluates how Aurora360.ai ArchFlow and FeatureFlow can improve the software-delivery process when layered on top of Claude Code. The comparison focuses on a baseline of Claude Code used with a standard Research → Plan → Implement workflow, implemented through Plan mode and user-driven prompting discipline, versus Aurora-supported workflows that introduce structured artifacts, stage gates, validation checks, architecture decision records, and human review points.

Background

The assessment is based on empirical A/B benchmark results from two internal KPI improvement analyses: one for ArchFlow and one for FeatureFlow. The benchmarks compare Claude Code used with a plain RPI (Research - Plan - Implement) approach against Claude Code supported by Aurora360.ai ArchFlow or FeatureFlow on comparable work items. The ranges should therefore be treated as benchmarked results within the tested scenarios and interpreted together with the defined baseline, project complexity, team seniority, and delivery context.

At a Glance

Objective
Present empirical KPI improvements of using ArchFlow and FeatureFlow on top of Claude Code compared with Claude Code using plain RPI approach.

Baseline
Claude Code used with a standard Research → Plan → Implement workflow, implemented through Plan mode and user-driven prompting discipline.

Improvement
Aurora360.ai workflows with structured artifacts, gates, validators, ADRs, traceability, and human review.

Scope
Greenfield architecture, brownfield/legacy modernization, and brownfield feature development.

Evidence type
Empirical A/B benchmark results from internal comparative analyses.

Methodology

The case study follows the same structural logic as a delivery case study: define the goal, establish the baseline, apply the solution, measure expected outcomes, and extract recommendations. Instead of describing a single customer implementation, it documents a comparative delivery model for AI-native software engineering.

Scope and Objectives

1. Define a fair baseline: Claude Code used with plain RPI, without persistent artifact structure, formal gates, or ADR governance.
2. Assess ArchFlow for greenfield architecture work, where the goal is an implementation-ready architecture and a governed implementation plan.
3. Assess ArchFlow for brownfield and legacy modernization, where the goal is validated current-state understanding, technical-debt discovery, and a dependency-aware modernization roadmap.
4. Assess FeatureFlow for brownfield feature development, where the goal is verified implementation rather than raw code-generation speed.
5. Identify the strongest benchmark KPIs, caveats, and conditions under which the measured improvements are most applicable.

Methodology and Caveats

All figures in this document are presented as empirical A/B benchmark results comparing the same underlying coding agent, Claude Code, used with plain RPI against Claude Code supported by ArchFlow or FeatureFlow. The benchmark interpretation assumes comparable tasks or repositories, competent practitioners, the same review standards, and the same Definition of Done across the baseline and Aurora-supported scenarios.

- The results are presented as empirical benchmark outcomes rather than theoretical or mechanism-only estimates.
- Actual ranges will vary by team seniority, codebase size, domain novelty, existing documentation, project complexity, and compliance requirements.
- For external communication, the benchmark scope and baseline conditions should be stated clearly so that the reported ranges are understood in the context of the measured scenarios, repositories, review process, Definition of Done, and quality checklist.
- The ArchFlow greenfield time-to-architecture metric should be presented with its benchmark context: the summary table reports a 55-70% faster outcome, while the detailed section includes a more conservative 10-20% reduction. The applicable figure should be selected based on the comparable project scope and measurement setup.

Phase 1: Determining the baseline and target outcomes

The baseline is Claude Code used with a plain RPI approach. In this setup, the coding agent researches the task, proposes a plan, and then implements. This can work well for scoped tasks, but it often lacks durable artifacts, gate-based validation, formal architecture decision governance, and systematic traceability.

The target outcome is not simply faster code generation. The target is faster and more reliable delivery of reviewed, tested, traceable, and implementation-ready software artifacts.

Baseline Limitations

- Architecture decisions are often implicit and discovered incrementally through code.
- Plans may be disposable conversation artifacts rather than durable project records.
- Research can be biased toward the intended solution when the goal is visible too early.
- Testing may be invented after implementation, creating self-verification risk.
- Traceability between requirements, decisions, implementation tasks, and verification artifacts is weak by default.

RPI	ArchFlow	FeatureFlow
Baseline	**Architecture governance**	**Feature delivery**
Claude Code with plain Research - Plan - Implement	Greenfield, brownfield, and modernization	Questions - Research - Design - Plan - Implement - Verify

Phase 2: ArchFlow for greenfield architecture

For greenfield projects, ArchFlow strengthens the architecture work that usually happens implicitly or inconsistently in an RPI-based flow. It front-loads architectural reasoning into structured phases, makes major decisions explicit through ADRs (Architecture Decision Records), applies gate checks, and links architecture to implementation through traceable artifacts.

The strongest value proposition is not only speed. It is governance: fewer architecture violations, stronger decision coverage, lower rework, and better traceability from requirements to implementation.

Benchmark KPI Impact - Greenfield

KPI	Baseline / ArchFlow	Estimated improvement
Time to implementation-ready architecture	Benchmark context includes a conservative 10-20% reduction in the detailed analysis and 55-70% faster in the summary table	Faster path to reviewed architecture; communicate the applicable benchmark range with the tested scenario and baseline definition.
Architecture decision coverage	20-40% baseline -> 85-95% ArchFlow	+50-65 percentage points
Architecture violation rate	N/A	70-85% reduction
Architecture-related rework	25-40% of implementation effort -> 8-15%	50-70% reduction
Architecture-to-implementation traceability	5-15% baseline -> 85-95% ArchFlow	+70-80 percentage points

Why It Matters

In a plain RPI flow, the agent may cover the decisions the developer explicitly asks about, while important cross-cutting concerns such as security posture, NFRs, data classification, deployment topology, and integration patterns remain implicit. ArchFlow makes these concerns visible through structured architecture phases and gate checks.

Phase 3: ArchFlow for brownfield and legacy modernization

For brownfield and legacy modernization, ArchFlow addresses a different problem: incomplete understanding of the current system. Ad hoc code scanning can produce a confident but incomplete architecture snapshot. ArchFlow instead reconstructs the current-state baseline through structured extraction phases and requires validation before modernization work proceeds.

The strongest business KPI is wrong-path rework avoided. In modernization, starting down the wrong path can consume weeks or months before a dependency, transaction boundary, data model, security constraint, or integration consumer is discovered.

Benchmark KPI Impact - Brownfield / Legacy Modernization

KPI	Baseline / ArchFlow	Estimated improvement
Time to validated current-state architecture baseline	2-4 weeks baseline -> 3-7 days ArchFlow	60-75% reduction
Technical-debt discovery coverage	30-50% baseline -> 70-85% ArchFlow	+35-45 percentage points
Architecture reconstruction accuracy	50-65% baseline -> 80-90% ArchFlow	+20-30 percentage points
Modernization roadmap completeness and prioritization quality	35-55% baseline -> 80-90% ArchFlow	+30-45 percentage points
Modernization rework / wrong-path avoidance	30-50% of effort wasted -> 10-15%	50-65% reduction

Why It Matters

ArchFlow is most valuable where current-state understanding is a prerequisite for action: unfamiliar legacy systems, compliance-sensitive platforms, major redesigns, or modernization initiatives with significant integration and dependency risk.

Phase 4: FeatureFlow for brownfield feature development

FeatureFlow is designed for brownfield feature delivery, where the issue is not simply whether the agent can write code, but whether the code is built from clarified requirements, researched facts, approved design, a staged implementation plan, and a pre-defined verification contract.

FeatureFlow turns the RPI discipline into a governed pipeline: Questions - Research - Design - Plan - Implement - Verify. This makes stage boundaries, artifacts, approvals, and validation more explicit than in a plain RPI conversation.

Benchmark KPI Impact - Brownfield Feature Delivery

KPI	Baseline / FeatureFlow	Estimated improvement
Time to verified implementation	Can be 10-25% slower on trivial tasks; 15-30% faster on complex/cross-boundary features	Scope-dependent; U-shaped impact
Post-implementation rework rate	Assumptions often emerge during plan or implementation in RPI	25-45% relative reduction
Defect escape rate / change failure rate	Tests may be invented by the same agent after code exists	30-50% relative reduction
Acceptance-criteria verification coverage	Inconsistent baseline: ~40-70%	Near-complete coverage: ~90-100%
Codebase pattern adherence	Pattern discovery is opportunistic in RPI	25-35% relative improvement

Why It Matters

FeatureFlow is strongest on quality and completeness KPIs. It is less suitable for trivial edits because the ceremony of multiple stages and human gates can exceed the value of prevented rework. It is most appropriate for user-facing, cross-boundary, risky, or long-lived behavior in brownfield systems.

Summary of Benchmark Outcomes

The following summary consolidates the strongest KPI signals from both A/B benchmark analyses. The values should be presented together with their defined baselines and scope conditions, because measured improvement varies by project complexity, codebase size, existing documentation, team seniority, and risk profile.

70-85%
fewer architecture violations
ArchFlow greenfield

50-65%
less wrong-path rework
ArchFlow brownfield

30-50%
fewer escaped defects
FeatureFlow

Solution	Headline KPI	Estimated impact	Why to lead with it
ArchFlow - Greenfield	Architecture violation rate	70-85% reduction	Most concrete governance KPI.
ArchFlow - Greenfield	Traceability coverage	5-15% -> 85-95%	Strong auditability and future-change value.
ArchFlow - Brownfield	Time to validated baseline	2-4 weeks -> 3-7 days	Accelerates current-state understanding.
ArchFlow - Brownfield	Wrong-path rework avoided	50-65% reduction	Direct cost and schedule impact.
FeatureFlow	AC verification coverage	~40-70% -> ~90-100%	Most mechanically enforceable FeatureFlow gain.
FeatureFlow	Defect escape / change failure	30-50% reduction	Independent verification and pre-defined test contract.

Key Recommendations

1. Use ArchFlow for architecture-level work, not routine implementation.

ArchFlow is most valuable for new systems, major redesigns, compliance/audit needs, or unfamiliar legacy systems where durable architecture artifacts and traceability matter.

2. Use ArchFlow brownfield before modernization execution.

A validated current-state architecture baseline and dependency-aware roadmap reduce the risk of wrong-path modernization effort.

3. Use FeatureFlow for risky brownfield features.

FeatureFlow is best suited to user-facing, cross-boundary, complex, or long-lived functionality where rework and escaped defects are expensive.

4. Do not force the full FeatureFlow pipeline onto trivial edits.

For small, low-risk changes, plain RPI or direct coding may be faster and sufficient.

5. Measure verified delivery, not code-generation speed.

The primary KPI should be time to reviewed, tested, accepted implementation, not the time required for an agent to produce a first draft.

6. Preserve artifacts and decisions.

Persistent requirements, ADRs, plans, test contracts, verification results, and traceability files are essential for auditability, onboarding, and future AI-assisted evolution.

Conclusion

The expected benefit of Aurora360.ai is not that it replaces Claude Code or other AI coding agents. The benefit is that it improves the delivery system around the agent. ArchFlow adds architectural discipline, decision governance, traceability, and modernization structure. FeatureFlow adds staged feature delivery, approval gates, durable artifacts, and verification discipline.

The case for ArchFlow is strongest where architecture quality, modernization risk, compliance, or long-term maintainability matter. The case for FeatureFlow is strongest where the cost of rework and escaped defects is higher than the overhead of a structured workflow.

Ready to introduce governed AI-native software delivery?

Use Aurora360.ai to move from ad hoc AI coding-agent usage to structured, traceable, and measurable SDLC workflows.

aurora360.ai

Letališka Cesta 29b
1000 Ljubljana, Slovenia
info@comtrade360.com

aurora360.ai