

SOFTWARE

Ship work you can stand behind

Alongside People / Advice and hands-on implementation

A practical path to better software delivery with people and AI

Your team may already be using coding assistants and agents, yet still be asking what good looks like beyond a faster first draft. We help you make sense of those experiments and decide what to improve next. You can bring that uncertainty; we can find the right starting point together.

For Melbourne software businesses and internal development teams, the useful question is whether AI helps a worthwhile change reach users with less friction and dependable quality. Writing a first draft of code is one part of that journey. Understanding the request, reviewing the result, releasing it and supporting it are part of the work too.

Our starting point is to start with one bounded delivery problem and measure the whole journey. A tool that generates more changes can increase the review queue. A tool that helps people resolve ambiguity, produce evidence and keep changes small may improve the flow. The trial needs to reveal which is happening in your team.

Our five stages are **Understand, Try, Repeat, Operate, Improve**. Each produces evidence before a larger commitment.

Find the constraint behind the busy team

Consider where a request actually waits. A product manager may know the desired outcome but leave edge cases unstated. An engineer may spend a morning finding the relevant behaviour in an unfamiliar service. A reviewer may need to reconstruct the purpose of a large change. A release may wait because nobody owns the operational check.

These are different problems. A coding assistant could help explore an unfamiliar repository. A clearer acceptance brief could remove repeated clarification. A standard evidence packet could make reviews easier. Some bottlenecks need a changed responsibility or a simpler process rather than an AI tool.

DORA's guidance examines throughput and instability across five delivery measures. It recommends application-level context and warns against targets people game. That supports measuring reliable delivery rather than code volume. [DORA delivery metrics](#).

Before adding anything, trace several recent changes from request to release. Separate active work from waiting. Ask the people doing the work what they repeatedly explain, check or repair. Their answers define the trial.

A fictional workflow: a clearer booking confirmation

This example is fictional and reports no client results.

Imagine a Melbourne business with a small booking product. Support receives questions because the confirmation page gives an unclear next step after a booking changes. The team chooses this one behaviour as a trial. It excludes payments, identity, data migration and any broad redesign.

The product owner writes a brief explaining the customer problem, the intended message, cancellation behaviour and the situations that must remain covered. An engineer gathers the relevant files, existing tests and design conventions. The AI assistant is asked to describe the current flow with file references before proposing a change. Where the explanation conflicts with the actual code, the engineer corrects it.

The assistant then prepares a small change and draft tests in an isolated working area. It must show what changed, why, and which checks actually ran. A successful test cannot become evidence for a different untested scenario. The engineer examines the diff, checks the expected behaviour against the brief and runs the relevant checks independently.

A reviewer receives the brief, the proposed change and the evidence together. They can reproduce the original issue and the corrected behaviour. If the change requires excessive explanation or touches unrelated code, it returns for simplification. The release owner then follows the normal release process and verifies the page after release.

GitHub's application card notes that generated code and explanations can be inaccurate and calls for review and testing. Its code-review documentation describes automated feedback and suggested fixes. We would use those capabilities as inputs to this workflow, while keeping a named person responsible for acceptance. [Copilot application card](#), [Copilot code review](#).

Shared evidence lets everyone assess the purpose and result. The trial would establish whether AI makes that journey easier.

What would make this worth continuing?

The following is a decision aid for the fictional trial, not an observed result. Agree the limits with the people responsible before testing.

- **Accepted result:** An accepted bounded change with reproducible evidence and a named release owner.
- **Who judges it:** The product owner, engineer, independent reviewer and release owner.
- **Hold and return to the manual route when:** A required check is weakened, a material regression is unresolved or the evidence claims a scenario that was not checked.
- **Evidence for another step:** Compare clarification, implementation, review and correction across similar changes; continue observing sparse production outcomes.

A useful fictional evidence packet names the requirement, describes the changed behaviour, lists checks actually run, identifies any remaining gap and names the release owner. Passing tests support the scenarios they exercise; they do not establish every possible product behaviour.

Use the [one-workflow worksheet](#) to record the agreed question, evidence and next decision.

Understand: agree what better means

We begin with the people responsible for the product, engineering, review and operation. Together we map one change type, identify its most costly friction and define an acceptable result. The baseline includes clarification, implementation, review, correction and release work.

The aim is a focused problem everyone recognises. Continue when there is a short workflow map, a written acceptance brief, a baseline sample and a named decision owner. We also record which repositories and information may enter the chosen tool. A tool decision follows that context.

Try: complete one change with evidence

We configure a contained trial around a familiar, reversible change. The team records the tool, model where visible, settings and information supplied. It compares the proposed result with the acceptance brief and keeps a record of failed attempts as well as useful ones.

The aim is learning from a complete task. Continue when there is a reviewable change, meaningful checks, reviewer notes and total human effort. A plausible explanation or a green screenshot alone is insufficient. If the team cannot reproduce the result, the trial is not ready to repeat.

Repeat: see whether the method survives variation

One well-chosen example can conceal a fragile process. We repeat the method across several comparable changes, including an ambiguous request and a case where the assistant should ask for clarification. We keep the scope small enough to inspect every result.

The aim is a reusable way of working. Continue when there is an agreed task brief, examples of accepted and rejected outputs, and evidence about review effort across the sample. A method that only works with one expert continually rescuing it needs more work before others depend on it.

Operate: make ownership and recovery ordinary

Once the method is useful, we connect it to the team's normal work. We establish access, task boundaries, review responsibility, cost visibility and a manual fallback. Release authority stays explicit. The team can stop a task, inspect its changes and return to an earlier state.

The aim is an improvement people can rely on during a normal week. Continue when there is a working runbook, an assigned owner, a demonstrated recovery and evidence from the approved environment. Passing a local trial does not establish that production access or monitoring is ready.

Improve: use results to choose the next change

We review whether the original constraint moved. Perhaps implementation became quicker while review became slower. Perhaps clearer briefs helped more than code generation. Either finding is useful. The next action might be narrowing the task, improving context, changing tools or stopping the approach.

The aim is informed investment. Continue when there is a dated decision supported by the trial record, with one next experiment and its success criteria. Expanding automation is an option when the evidence supports it.

A proposed 30-day trial

This proposed evaluation is not a commercial commitment or delivery guarantee.

During days 1-5, select one application and one recurring change type. Review recent comparable work, agree acceptance criteria and establish the baseline. During days 6-12, complete a contained example with an engineer and independent reviewer. Use days 13-21 to repeat the method, recording exceptions and correction work. During days 22-30, assess whether the method is suitable for routine use and document the decision.

If permissions, suitable examples or review capacity are unavailable, use synthetic material and describe the limits. Do not rush production integration to fit the calendar.

Measure completed value and its full cost

Agree thresholds before the trial. Compare similar changes and show the sample size. Record elapsed request-to-acceptance time, active human effort, clarification rounds, review time, material corrections and reopened work. Separate waiting for a release window from preparing the change.

Use quality guardrails alongside speed. Required checks must remain intact; known regressions and security findings must be resolved under the team's normal standards. Record any escaped defects and recovery work. For a short trial, production failure rates may be too sparse to support a conclusion. Continue observing rather than reporting a precise improvement from a tiny denominator.

Calculate cost per accepted change from tool usage, setup, human review and correction. Show recurring costs separately from the initial investment. Keep model or configuration changes visible so two runs are not treated as equivalent without explanation. Avoid individual rankings based on prompts, tokens or lines written.

The decision is straightforward: continue when the work becomes easier without weakening acceptance; revise when benefit depends on hidden effort; stop when the workflow adds more burden than it removes.

How we help

Alongside People brings advice and implementation together. We can help your team identify the constraint, define the task and its evidence, select an appropriate setup, and build the workflow around existing delivery practices. We then help evaluate what happened and make the operating method understandable to the people who will own it.

Paul Volpato is our founder and CEO. A useful starting conversation is about one recent change that took longer, required more review or created more rework than it should have.

Sources and scope

Understand / Try / Repeat / Operate / Improve is Alongside People's original discussion framework. It is not a certification, validated maturity assessment or official standard.

The workflow and five stages are our proposed approach. The example is fictional. Sources support the specific factual statements above and do not establish results for your business. All sources were checked on 3 October 2026; product features and access should be refreshed before a trial.

- [DORA's software delivery performance metrics](#), DORA; maintained guidance, checked 3 October 2026.
- [Application card: GitHub Copilot Chat](#), GitHub; maintained documentation, checked 3 October 2026.
- [About GitHub Copilot code review](#), GitHub; maintained documentation, checked 3 October 2026.

If you are already trying AI and are unsure what deserves further investment, [tell us what you have tried and what you want to improve](#). We can help find the starting point together.

Read the structured HTML edition: alongsidepeople.com/resources/ai-for-software