A PRAGMATIC DISRUPTOR™ FRAMEWORK

The IDE Trust Contract

A framework for scalable, high-assurance AI delivery — so AI speed never outpaces safety.

Version 1.0

July 2026

Thought Leadership / Free Framework

Publisher: Pragmatic Disruptor™ LLC

Pragmatic Speak
by Pragmatic Disruptor, LLC™

The IDE Trust Contract

Publisher	Pragmatic Disruptor™ LLC
Product line	Thought leadership / free framework
Audience	Builders, tech leads, and delivery firms adopting AI-assisted IDEs
Outcome	Name, document, and run a Trust Contract so AI speed does not outpace safety
Version	1.0 · July 2026

> **Independence note:** Pragmatic Disruptor™ LLC is an independent publisher and is not affiliated with, endorsed by, or sponsored by Anysphere, Inc., GitHub, Anthropic, or other tool vendors named for illustration.

Why this exists

In the AI-accelerated delivery era, the bottleneck has shifted from coding speed to **orchestration quality**. Tools generate drafts instantly. Risk rises when trust stays informal — "looks good," "the model said so," or "it worked on my machine."

An **IDE Trust Contract** is how you treat trust as a **system**, not a feeling. It does not ban AI. It defines what AI may draft, what automation must prove, and what a human must authorize before you ship.

This document is a **public framework**. It is useful on its own. It is **not** a substitute for your team's full operator playbooks, prompt libraries, or product-specific rule packs.

Canonical definitions (use these words)

Term	Meaning
Trust Contract	A written agreement of what AI may generate, what automation must verify, and what a human must authorize before release.
Control Pack	A small, reusable set of standards (security, tests, accessibility, secrets, evidence) applied to a project or portfolio.
Trust zones	Labeled areas of the codebase or change types with different human supervision levels.
Release evidence	The minimum artifacts that make "ready" auditable (tests, checks, residual risk notes) — not vibes.
Adapter pattern	Keep governance in the repo/pipeline; map it to whatever IDE or assistant the client or team requires.

If others adopt these terms, the industry conversation gets sharper. That is the point of publishing the framework.

The Triad Operating Model

Every shippable change should be visible against three lanes:

1. AI Speed (Generate)

- Drafts implementation options
- Surfaces risk hypotheses
- Produces documentation stubs and change summaries

Constraint: Generation is not permission to merge.

2. Automation (Verify)

- CI/CD policy gates
- Regression and targeted suites
- Capture of artifacts that later support go/no-go

Constraint: Green CI is necessary; it is not always sufficient for red-zone work.

3. Human (Authorize)

- Must-test journeys that matter to users or clients
- Explicit residual-risk visibility
- Final go/no-go authority with a named owner

Constraint: Speed without a human authorize lane is gambling with brand and liability.

> **Summary:** You do not have a coding-speed problem. You have a **trust-at-scale** problem. Close the triad, or defects scale with throughput.

Trust zones (supervision, not superstition)

Trust is not binary. Label work before a deep session:

Zone	Human posture	Typical examples
Green	Move with light review	Boilerplate, docs, low-risk UI, additive tests
Yellow	Stepwise implement; verify before merge	Business logic, APIs, refactors with contracts
Red	Plan first; smallest change; human approval before merge	Auth, billing, migrations, secrets, compliance-sensitive paths

Principles (not scripts)

- Zone is chosen by **risk and blast radius**, not by how confident the model sounds.
- Red-zone work can still use AI; it must not skip plan → minimal diff → explicit approve.
- Put the zone decision where the team actually starts work (ticket, PR template, or session note) — consistency beats ceremony.

Do not treat this table as a complete control matrix. Teams must map their own paths and domain risks into the three zones.

What belongs in a Control Pack

A Control Pack is deliberately small. If it feels like a constitution, it will rot.

Include at least these **categories** (write your own standards in each):

1. **Secrets & data** — what must never enter chat or commits; where secrets live
2. **Dependency policy** — when upgrades are allowed; when they need a separate change
3. **Definition of done** — what "done" means (behavior, tests, docs) — without dumping your full command cookbook publicly if that is proprietary
4. **Security & accessibility bar** — minimum checks for the type of product you ship
5. **Evidence expectations** — what must appear in the release trail for yellow/red work
6. **Environment notes** — how local, CI, and deploy are expected to relate

Store Control Packs where the **repo and pipeline** can enforce them. IDE settings are adapters, not the source of truth.

The Adapter Pattern (avoid vendor lock-in)

Core (repo-owned): policies, templates, CI gates, and evidence expectations that survive a tool switch.

Adapters (tool-owned): IDE rules, assistant instructions, or editor integrations that point at the core.

Benefit: If a client mandates a different assistant mid-engagement, governance does not restart from zero.

What to evaluate before standardizing on a tool

(decision prompts, not a vendor ranking)

- Does the data boundary allow external LLM processing?
- Does the team's stack and IDE ecosystem fit day-to-day work?
- Can you export or attach enough evidence for release decisions?
- Can standards be expressed in-repo, not only in a proprietary UI?

Tool landscapes change quickly. Prefer **roles** (interaction layer, compliance integration, audit/CLI agent, context continuity, air-gapped assistance) over permanent brand loyalty.

Draft your Trust Contract (fill in)

Copy this outline into your wiki or repo. Complete it in plain language. Keep it one page if you can.

```
IDE TRUST CONTRACT — [Project / Engagement name]
Owner (go/no-go): [name/role]
Effective date: [date]
Review cadence: [e.g. quarterly or after major incidents]

1. PURPOSE
We use AI-assisted IDEs to increase delivery speed while keeping
safety and auditability intact.

2. GENERATE (AI)
Allowed: [draft types the team permits]
Disallowed: [e.g. pasting production secrets; silent production access]
Required planning for: [red-zone domains]

3. VERIFY (AUTOMATION)
Required gates before merge/release: [list categories — tests, lint,
security, a11y, etc.]
Evidence retained: [what must be linkable or attached]

4. AUTHORIZE (HUMAN)
Named authorizer for release: [role]
Must-test journeys: [user-visible paths that always get a human pass]
Red-zone always requires: [plan + minimal change + explicit approve]

5. TRUST ZONES
Green paths/types: [ ... ]
Yellow paths/types: [ ... ]
Red paths/types: [ ... ]

6. CONTROL PACK VERSION
Pack name / version: [ ... ]
Where it lives in the repo: [ ... ]

7. METRICS WE WILL WATCH
[Escaped defects / rework / onboarding time — pick 2-3 and define
how you count]

8. CHANGE LOG
[Date — what changed in this contract]
```

This skeleton is intentionally incomplete. Filling it is the work. That is also where your competitive edge lives — your standards, commands, and evidence schema — not this public outline.

90-day pilot (validate without theater)

Use a pilot when you want the Trust Contract to become delivery IP, not a slide.

Phase	Focus	Exit signal
Phase 1	Define one Control Pack for one project	Pack reviewed and named; zones assigned for hot paths
Phase 2	Encode gates in repo + CI; map thin IDE adapters	Gates fail closed on critical violations; team knows how to unblock
Phase 3	Measure	Baseline vs. pilot period on 2–3 agreed metrics

Suggested metric families

(define counting rules yourselves)

- Escaped defects after merge/release
- Rework rate (bounce-backs, hotfix frequency)
- Onboarding speed for a new engineer to first safe ship

> **Pilot failed if:** zones exist but nothing blocks high-risk merges, or metrics are undefined so "success" cannot be challenged.

Adoption checklist (print-friendly)

Foundation

- [ ] Trust Contract document exists and names a go/no-go owner
- [ ] Triad is understood: Generate ≠ Verify ≠ Authorize
- [ ] Green / Yellow / Red mapped for this project's real hot paths

Control & pipeline

- [ ] Control Pack categories filled with your standards
- [ ] CI (or equivalent) enforces the gates that matter
- [ ] Secrets and data boundaries stated where engineers will see them

Operating habit

- [ ] Multi-file work ships in reviewable chunks with clear scope
- [ ] Local vs. CI drift is treated as a first-class incident, not banter
- [ ] Release conversations can point at evidence, not only opinion

What this guide is — and is not

This guide IS	This guide IS NOT
A shared vocabulary and operating skeleton	A full prompt library or rule dump
Enough to start a pilot and a one-page contract	A complete security or compliance certification
IDE-agnostic by design	An endorsement of any single vendor
Complementary to deeper operator materials	A substitute for those materials

Deeper, implementation-heavy workflows for day-to-day AI-assisted IDE operation are published separately under **Pragmatic Speak** (series and playbook offerings) at www.pragmaticdisruptor.com.

If AI is already in the critical path of delivery, informal trust is a silent process failure.
Write the contract. Own the Control Pack. Keep the human authorize lane.
Measure whether speed and safety moved together.

That is the IDE Trust Contract.

Pragmatic Disruptor™ LLC builds practical frameworks and guides so builders and leaders turn AI-assisted velocity into measurable, shippable outcomes.