ChatGPT

State of AI-Assisted Software Development When AI Implements Most of the Code

Executive summary

As of 2026-02-21 (Europe/Kyiv), developer workflows are rapidly converging on a new division of labor: AI systems generate and modify large amounts of code quickly, while humans increasingly act as **specifiers, constraint-setters, reviewers, integrators, and risk managers**. Practitioners repeatedly describe the same macro-shift: coding time shrinks, but **review, context management, and safety work expand**—often becoming the bottleneck. [1]

A key structural change in the last 12 months is that **"agentic" coding is no longer one workflow**; it's a spectrum with distinct failure modes and human factors at each autonomy level. A practical mental model used by practitioners is: (1) suggestion-level copilots, (2) multi-file "accelerated" editing under steering, (3) interactive agents that plan + execute tool calls, and (4) **asynchronous background agents** that deliver draft PRs for audit/approval. [2]

What's happening now can be summarized as five "state of affairs" claims, each strongly supported by recent practitioner reports and platform changes:

Humans are becoming "reviewers on an assembly line," and this is cognitively expensive. A recurring practitioner observation is that generative work feels energizing while evaluative work becomes draining—especially under higher PR throughput and nondeterministic outputs. This manifests as decision fatigue, reduced attention, and weaker trust calibration over time. [3]

Context engineering is becoming a first-class engineering discipline. Teams are externalizing "how we build software here" into repo-scoped instruction files (and increasingly modular rule packs), plus persistent "memory" artifacts that survive model/session resets. This responds to two realities: (a) agents are bottlenecked by what they can reliably retrieve, and (b) "garbage context → garbage code" is now a common failure explanation. [4]

PRs are becoming the unit of autonomy. The highest-adopted "autopilot" pattern is: assign a well-scoped issue/task to an asynchronous coding agent, receive a draft PR, iterate via comments, merge with stronger guardrails. This pattern is being productized directly into mainstream platforms, making PR review the primary human-in-the-loop control surface. [5]

Tool ecosystems are standardizing around "connectors" and "procedures." Two forms show up repeatedly in real workflows: (a) protocol-driven tool access (notably MCP servers) and (b) reusable procedural bundles ("skills") that package instructions + scripts + resources and can be loaded on demand. Both aim to reduce improvisational prompting and stabilize outcomes across tasks and team members. [6]

Security risks have shifted from "bad code" to "agentic attack surfaces." The last 12 months saw multiple disclosed vulnerability classes where prompt injection plus agent tool access (or base IDE features) can yield data exfiltration or code execution. Practitioners are responding with "least privilege" for agents, stricter approvals, sandbox/network allowlists, and strong distrust of unvetted context sources (e.g., rule files, MCP servers, filenames). [7]

What changed in the last 12 months

The timeline below focuses on changes between 2025-02-21 and 2026-02-21, emphasizing adoption-relevant product shifts and ecosystem standardization.

- **2025-03-18 — "Rules file backdoor" supply-chain vector publicized.** Security researchers described how instruction/rules files used by coding assistants can be abused to inject hidden malicious instructions into the generation process. [8]
- **2025-03-26 — Repo-scoped "custom instructions" mainstreamed in IDE workflows.** The .github/copilot-instructions.md pattern is documented as a way to provide project-specific instructions automatically. [9]
- **2025-04-08 — "Slopsquatting" (hallucinated dependency names weaponized) enters common security vocabulary.** Supply-chain defenders describe the pattern and mitigations for AI-assisted dependency hallucinations. [10]
- **2025-05-19 — Background PR agents become first-class in a major platform.** A coding agent workflow is announced in public preview: cloud environment, repo exploration, changes, tests/linters, draft PR, iterate via PR comments. [11]
- **2025-08-28 — "Instruction layering" expands: AGENTS.md supported alongside multiple instruction file conventions.** This normalizes a multi-file hierarchy of agent instruction sources across tools. [12]
- **2025-09-25 — Background coding agent reaches GA for paid users.** The "delegate → draft PR → review/comments" loop is formalized as a mainstream workflow. [13]
- **2025-10-29 — Multi-agent IDE workflows are productized (parallel agents + worktrees/remote + "best-of-N").** This codifies parallelism and selection as a first-class strategy for reliability and speed. [14]
- **2025-12-06 — "IDEsaster" vulnerability class published, reframing the threat model for AI IDEs.** Research reports 30+ vulnerabilities and an attack chain that weaponizes base IDE features once agents have autonomous actions. [15]
- **2026-02-02 — A dedicated "Codex app" is introduced, emphasizing skills and multi-step agentic work.** This reinforces "procedures as artifacts" (skills) and agent-first UX patterns. [16]
- **2026-02-04 — Multi-agent "hub" direction accelerates: third-party coding agents integrated into a major dev platform (public preview).** This pushes workflows toward orchestration and comparative evaluation of different agents/models in the same task flow. [17]
- **2026-02-17 — Delegation into background PR agents expands into desktop IDE flows.** This reduces friction to "send task → get PR," accelerating adoption of autopilot-with-guardrails for routine work. [18]

What remains hard and unsolved

Trust calibration does not scale linearly with throughput. Even experienced developers report that AI success trains them into higher acceptance and lower scrutiny, but failure does not always "debias" them

proportionally; this creates a drift toward over-trust (or cynical under-trust) rather than stable calibration. Practitioner accounts describe this as review fatigue + automation bias dynamics. [19]

Review and integration are still largely human-limited. Multiple accounts describe that AI can generate 2–6× more code for the same functional change, shifting review from "does it work?" to "is this necessary/maintainable?"—and making senior reviewers the limiting reagent. [20]

Security "guardrails" are lagging behind autonomy capabilities. The disclosed IDEsaster-style chains and prompt injection vectors show that "human approval" UX can be bypassed if autonomy controls are weak or context sources are untrusted; OS- and IDE-level features become exploitable primitives once an agent can edit settings or trigger background fetch/exec behaviors. [21]

Long-horizon work still breaks on context management and task decomposition. Tooling is evolving toward memory banks, modular rules, subagents, and "skills," but practitioners still report "context rot," prompt spirals, and big-bang diffs as dominant failure modes. [22]

Empirical pattern catalog

Patterns below are organized by degree of AI autonomy: Copilot mode → Accelerated mode → Agent mode → Autopilot-with-guardrails. Each pattern is grounded in repeated practitioner descriptions, platform docs/release notes, community bug threads, or security writeups.

Copilot mode

Pattern: "Tight autocomplete loop" (suggest → accept → run → fix)
What people do (workflow): - Keep AI suggestions on for boilerplate/glue code. - Accept partial completions quickly, then compile/run tests frequently to surface errors. - Use errors as the next prompt/suggestion seed ("here's the stack trace; patch it"). - Prefer this for code that is easy to validate locally (types, lint, unit tests).
Where it works: repetitive edits, boilerplate, familiar stacks, codebases with fast feedback loops. [23]
Human factors: "momentum bias" (keep accepting because it's fast), attention narrowing to compiler/test feedback, risk of shallow semantic review when code looks clean. [24]
Failure modes: silent logic bugs; drift toward accepting patterns the team doesn't actually use; reduced willingness to step back and redesign. [24]
Mitigations used: enforce "run tests before commit"; require explicit intent comments for non-obvious changes; stop using copilot suggestions when doing architecture-heavy work. [24]
Artifacts: "pre-commit checklist: tests/lint pass"; "definition of done" includes local validation.
Evidence strength: repeated reports (practitioner synthesis). [23]
Sources (with dates): - "Learnings from two years of using AI tools for software engineering" (2025-06-24). [23]
- "AI fatigue is real and nobody talks about it" (2026-02-08). [25]

Pattern: "Chat-to-navigate the codebase" (semantic Q&A as replacement for grep/browser search)
What people do: - Ask "where is X validated?" / "what calls this endpoint?" / "what's the pattern for Y?" - Use results to jump to files/functions rather than trusting generated patches immediately. - Treat answers as a search accelerator, then confirm by reading source. [23]

Where it works: unfamiliar codebases, large repos, systems with consistent patterns. [23]
Human factors: anchoring on the first plausible location; "authority halo" when answers are confident; reduced deep reading if explanations feel sufficient. [24]
Failure modes: wrong file targets (especially with partial indexing); false confidence; missing cross-cutting constraints not represented in retrieved context. [23]
Mitigations used: require links/citations to exact symbols/files; ask for multiple hypotheses; confirm by running tests or tracing.
Artifacts: "codebase Q&A prompt template" (see playbook).
Evidence strength: repeated reports (practitioner synthesis). [23]
Sources (with dates): - "Learnings from two years..." (2025-06-24). [23]

Pattern: "AI-assisted PR comprehension" (summaries, review maps, and 'what changed?' explainers before human review)
What people do: - Ask AI to summarize a diff/PR into "intent + files + risk areas." - Use AI to generate a reviewer checklist tailored to the change type. - Review the checklist first, then inspect the diff in that order ("review map"). [26]
Where it works: medium-size PRs; teams already overloaded with reviews; onboarding reviewers unfamiliar with the touched area. [27]
Human factors: reduces cognitive load by chunking; but risks "rubber-stamp via narrative" if the summary is persuasive and wrong. [28]
Failure modes: summary omits the critical issue; mismatched incentives (generator wants merge; reviewer wants safety). [29]
Mitigations used: require "unknowns / assumptions" section; mandate spot-check of high-risk files even if summary claims low risk.
Artifacts: PR review rubric ("risk list" + "assumptions").
Evidence strength: repeated reports (review-fatigue mitigation strategies). [30]
Sources (with dates): - "Tackling Review Fatigue by Document Driven Agentic Coding" (2025-09-19). [29]
- "AI fatigue is real..." (2026-02-08). [25]
- "Why AI coding tools shift the real bottleneck to review" (2026-01-20). [31]

Accelerated mode

Pattern: "Multi-file edit session with checkpoints/rollback"
(Examples: multi-file edit UIs with inline diffs, accept/reject chunks, and rollback points.)
What people do: - Start an "edit session" targeting a bounded change across multiple files. - Review proposed diffs inline; accept/reject in chunks. - If the session goes off-track, revert to checkpoints and re-run with tighter instructions. - Keep scope to "one commit's worth" to preserve reviewer comprehension. [32]
Where it works: refactors, mechanical migrations, repetitive edits touching many files. [33]
Human factors: lower friction increases willingness to try risky edits; but "accept-all temptation" rises when the diff UI feels authoritative. [34]
Failure modes: concurrent edits scramble files; partial application corrupts scripts; "looks clean but wrong" semantics. [35]
Mitigations used: constrain scope (single feature flag / single module); enforce checkpoints; require full test run after "accept all." [36]
Artifacts: "edit session prompt skeleton"; "rollback/run-tests rule."
Evidence strength: repeated reports + documented failure cases. [36]

Sources (with dates): - “What is GitHub Copilot Edits?” (last updated 2025-09-22). [32]
- Bug report: multi-edits scrambling scripts (2025-03-16). [37]

Pattern: “Plan-first, code-second” (explicit plan mode + clarification questions)
What people do: - Ask AI for a plan before making changes (files to touch, approach, tests to add). - Force clarifying questions early (requirements, constraints, acceptance criteria). - Approve the plan, then execute in small steps (“implement step 1 only”). [38]
Where it works: brittle codebases, cross-module changes, tasks with hidden requirements, team environments where review risk is high. [39]
Human factors: reduces impulsive “generate now” behavior; combats anchoring by making assumptions explicit; improves reviewer trust when intent is documented. [28]
Failure modes: plan-quality illusion (great plan, wrong implementation); plan drift when context changes mid-task. [25]
Mitigations used: require plan links to specific files/symbols; “three attempts then stop” to prevent prompt spirals. [25]
Artifacts: PLAN.md / “steps + acceptance checks” template (see playbook).
Evidence strength: repeated reports + shipped plan modes. [40]
Sources (with dates): - Cursor changelog: plan modes, clarification tooling, and long-running agents (2026-01 to 2026-02). [41]
- “Tackling Review Fatigue…” (2025-09-19) (explicit requirements → design → plan flow). [29]

Pattern: “Instruction layering” (repo-scoped guidance files as the real ‘process’) What people do: - Add repo-level instruction files to encode conventions, test commands, and constraints. - Use nested instructions for subdirs (monorepos) and avoid conflicts by scoping. [42]
- Split monolithic instruction files into modular “rule packs” to reduce context noise. [43]
Where it works: teams with many contributors; large repos; regulated or security-sensitive environments; high AI throughput. [44]
Human factors: reduces dependence on “prompt craft”; shifts expertise to maintaining shared artifacts; can create over-trust if instructions are treated as perfect. [28]
Failure modes: conflicting instructions; outdated rules; hidden malicious instructions (security). [45]
Mitigations used: version-control + code review for instruction files; “no network by default” for agent sandboxes; trusted-source policies. [46]
Artifacts: AGENTS.md + .github/copilot-instructions.md + modular rules directory.
Evidence strength: repeated reports + platform support. [47]
Sources (with dates): - “Context is all you need: Better AI results with custom instructions” (2025-03-26). [9]
- “Copilot coding agent now supports AGENTS.md custom instructions” (2025-08-28). [12]
- Cursor community discussion of modular rules to reduce “garbage context” (2026-02-10). [43]

Pattern: “Memory Bank + intentional resets” (persist state, then clear context to avoid drift)
What people do: - Maintain a small set of markdown “memory” files: project context, conventions, current plan, decisions. - When a session grows messy, reset the agent and rehydrate from memory files. - Keep memory curated to prevent “context rot.” [48]
Where it works: multi-day tasks; long refactors; teams switching between tools/agents; any workflow prone to drifting prompts. [26]
Human factors: offloads working memory; reduces re-explaining; but adds maintenance overhead and “meta-work.” [28]

Failure modes: stale memory creates repeated wrong assumptions; memory becomes too large and reintroduces noise. [49]
Mitigations used: keep memory “thin”; store only invariants and current milestone; timebox memory updates.
Artifacts: memory/ directory; “decision log” file.
Evidence strength: repeated reports + widely adopted templates. [50]
Sources (with dates): - “Memory Bank: How to Make Cline an AI Agent That Never Forgets” (2025-02-06). [51]
- “Tackling Review Fatigue…” (2025-09-19) (explicit plan docs + resets + milestone folders). [29]

Agent mode

Pattern: “Fix-until-green” (agent runs tests/linters and iterates)
What people do: - Give the agent a task plus the exact test/lint commands to run. - Let the agent attempt changes, run checks, interpret failures, patch, repeat. - Human intervenes mainly when agent gets stuck or proposes risky redesigns. [52]
Where it works: well-tested codebases; mechanical refactors; medium complexity tasks. [53]
Human factors: strong automation bias once tests are green; reviewers may treat CI success as semantic correctness. [54]
Failure modes: “tests pass but wrong”; brittle tests masking issues; over-defensive code bloat that passes checks but harms maintainability. [55]
Mitigations used: require targeted behavioral tests; add “risk-based manual testing” steps; enforce small PRs for reviewability. [56]
Artifacts: “test requirements” in instructions; CI gates; PR checklist.
Evidence strength: repeated reports + platform implementation. [52]
Sources (with dates): - “GitHub Copilot coding agent in public preview” (2025-05-19) (tests/linters before push). [11]
- Cursor 2.0 blog (2025-10-29) (native testing tool, iterate until correct). [14]

Pattern: “Tool gating + sandboxing as guardrail, not afterthought”
What people do: - Configure agents to require approval for risky actions (shell exec, network, credential access). - Restrict network egress via allowlists; isolate work in sandbox/VM environments. - Treat “agent authorization” and audit trails as part of sustainable human workflow. [57]
Where it works: enterprise environments; security-sensitive repos; when using MCP/tool plugins. [58]
Human factors: reduces reviewer anxiety and cognitive burden; combats “I can’t review everything” reality; introduces friction that can be bypassed under schedule pressure. [59]
Failure modes: YOLO/autonomous settings changed by the agent; prompt injection chains that exploit IDE features; complacency because “the sandbox exists.” [60]
Mitigations used: “trusted projects only” policy; lock down settings files; treat instruction/rules files as sensitive; organization-enforced allowlists. [58]
Artifacts: sandbox.json allowlists; policy doc; “no untrusted MCP servers.”
Evidence strength: repeated reports + major disclosed vulnerability classes. [58]
Sources (with dates): - “IDEsaster: A Novel Vulnerability Class in AI IDEs” (2025-12-06) (mitigations + threat model). [61]
- Cursor changelog: sandbox network access controls (2026-02-17). [41]

Pattern: “MCP toolbus” (use standard connectors for live context + actions)
What people do: - Connect agents to MCP servers for repo operations, docs lookup, ticket context, etc. - Use MCP to reduce manual copy/paste and improve tool reliability (structured tool calls). - Treat MCP servers like dependencies: pin versions, trust selectively, monitor changes. [62]
Where it works: workflows needing external context (issues, accessibility audits, API docs), multi-system automation. [63]
Human factors: shifts attention from code writing to “tool orchestration”; increases blast radius when trust is miscalibrated. [59]
Failure modes: malicious/compromised MCP server; prompt injection via tool outputs; unsafe data flows hidden behind “legitimate tools.” [64]
Mitigations used: “trusted MCP only,” least-privilege tokens, human-in-loop for writes, and monitoring tool payloads. [65]
Artifacts: approved MCP registry; tool allowlist; security review checklist for MCP servers.
Evidence strength: repeated reports + official tutorials + security warnings. [66]
Sources (with dates): - MCP specification (2025-11-25). [67]
- GitHub MCP server repo (n.d.; active in 2025–2026). [68]
- “Enhancing agent mode with MCP” tutorial (n.d.). [69]

Pattern: “Skills as procedural guardrails” (SKILL.md packages repeatable workflows)
What people do: - Encode repeatable procedures (e.g., “write migration,” “incident runbook,” “dependency audit”) as skills. - Let the agent load skills on demand to reduce context bloat and variability. - Share skills across teams to standardize outcomes and reduce prompt folklore. [70]
Where it works: recurring workflows; regulated environments; SRE/DevOps runbooks; migrations. [71]
Human factors: reduces cognitive load and onboarding cost; creates new governance surface (skills can embed risky scripts). [72]
Failure modes: poisoned skills; stealthy prompt injection via skill files/scripts; overconfidence in “standardized” procedures. [73]
Mitigations used: treat skills as code; code review; signed provenance; restrict executable scripts; least privilege. [74]
Artifacts: skills directory; skill authoring checklist; allowed-scripts policy.
Evidence strength: repeated reports + official docs + emerging security research. [75]
Sources (with dates): - “Agent Skills” documentation (n.d.). [76]
- “Introducing the Codex app” (2026-02-02) (skills as core feature). [16]
- “Agent Skills Enable… Prompt Injections” (2025-10-30). [77]

Autopilot-with-guardrails

Pattern: “Delegate → draft PR → review/comments → merge” (background PR agent loop)
What people do: - Assign a small, well-defined issue (bug, test coverage, refactor, docs). - Agent works in a cloud/dev environment, opens a draft PR, requests review. - Humans steer via PR comments; agent revises; human merges. [78]
Where it works: low-to-medium complexity tasks in well-tested repos; maintenance work; mechanical changes. [79]
Human factors: displacement of “ownership” feelings; review fatigue if PR volume spikes; temptation to treat agent as accountable teammate when it’s not. [80]
Failure modes: PR fails to run in correct environment; agent can’t create PR reliably; tasks too ambiguous; hidden security risks. [81]

Mitigations used: strict WIP limits; enforce PR templates + checklists; task scope rubric (“30-minute human equivalent”). [82]
Artifacts: “delegate-ready issue template”; PR checklist; “acceptance criteria required.”
Evidence strength: repeated reports + mainstream productization. [83]
Sources (with dates): - “GitHub Copilot coding agent in public preview” (2025-05-19). [11]
- “Copilot coding agent is now generally available” (2025-09-25). [13]
- “Using Cursor background agents” (2025-07-31). [84]

Pattern: “PR templates + structured summaries as audit surface”
What people do: - Require PR templates (“intent, approach, risks, tests run, rollout plan”). - Ensure background agents follow the PR template automatically. - Use templates to keep reviewers focused on risk and verification, not prose. [85]
Where it works: teams with many PRs/day; regulated orgs; AI-heavy codebases. [86]
Human factors: combats review fatigue by standardizing review questions; can become box-checking if incentives are misaligned. [26]
Failure modes: template filled with plausible nonsense; reviewers trust format over substance. [25]
Mitigations used: require “tests run (exact commands)” and “what I did not verify”; require repo links to logs/CI.
Artifacts: PR template (see playbook).
Evidence strength: platform support + repeated practitioner use. [87]
Sources (with dates): - “Copilot coding agent now supports pull request templates” (2025-11-05). [88]
- “AI fatigue is real…” (2026-02-08) (review as bottleneck). [25]

Pattern: “Document-driven agentic delivery” (requirements → design → plan committed with code)
What people do: - Write lightweight requirements + interface/design docs first. - Generate an explicit TODO plan with checkboxes and commit it. - Use docs as the shared source of truth for both agent guidance and reviewer comprehension. [26]
Where it works: complex domains; ML/data systems; codebases where reviewers lack full context; changes that must be justified. [29]
Human factors: reduces “proxy prompting through other humans” (reviewing AI-written docs/code with no mental model); improves ownership by making intent explicit. [26]
Failure modes: doc drift (docs say one thing, code does another); docs become AI-generated noise. [29]
Mitigations used: require doc review before code review; keep docs short; require acceptance tests tied to requirements. [82]
Artifacts: requirements.md, interface.md, plan.md, acceptance-tests.md.
Evidence strength: repeated practitioner reports with concrete workflow. [26]
Sources (with dates): - “Tackling Review Fatigue…” (2025-09-19) (explicit doc-driven workflow steps). [29]
- “AI fatigue is real…” (2026-02-08) (need durable infra: context + audit). [25]

Pattern: “Parallel agents + best-of-N selection” (worktrees/remote + compare outputs)
What people do: - Run multiple agents (or the same agent with different models) in parallel. - Isolate changes via worktrees or remote environments to avoid conflicts. - Compare PRs/patches and pick the best candidate to merge or to refine. [89]
Where it works: ambiguous tasks; refactors with many possible implementations; when reliability matters more than raw speed. [90]
Human factors: reduces anchoring to the first output; increases coordination overhead; can worsen fatigue if it produces “too many options.” [91]

Failure modes: WIP explosion; inconsistent style/architecture across candidate outputs; review overload. [28]
Mitigations used: cap parallelism; enforce one canonical instruction set; require shared plan + acceptance criteria. [92]
Artifacts: “model comparison rubric”; “one instruction layer rule.”
Evidence strength: repeated reports + tool support (multi-agent UIs). [89]
Sources (with dates): - “Introducing Cursor 2.0 and Composer” (2025-10-29) (parallel agents via worktrees/remote, choose best). [14]
- Cursor changelog: async subagents + long-running agents (2026-01 to 2026-02). [41]
- “GitHub adds Claude and Codex AI coding agents” (2026-02-04). [17]

Loop dynamics

These “stuckness loops” are recurring dynamics practitioners describe when AI is responsible for a large share of implementation. Each loop includes observed warning signals and interventions that map back to patterns above.

Loop: Prompt spiral (a.k.a. “just one more prompt” trap)
Trigger: output is ~70–80% correct, tempting iterative prompt tweaks rather than direct edits. [25]
Escalation dynamics: time sinks into prompt tuning; outputs drift; sunk-cost fallacy increases; fatigue reduces judgment. [25]
Early warning signals: third prompt attempt with no net improvement; increasing diff size; rising frustration. [25]
Loop breakers: “three attempts rule,” switch to direct coding for the core logic, or force plan-first with explicit acceptance criteria. [28]
Preventative patterns: Plan-first, code-second; memory bank + resets; multi-file edit checkpoints. [93]
Sources: “AI fatigue is real…” (2026-02-08). [25]

Loop: Review fatigue → rubber-stamping → downstream incidents
Trigger: AI increases code volume/PR throughput faster than review capacity grows. [94]
Escalation dynamics: reviewers skim; fewer meaningful comments; generator incentives diverge from reviewer incentives; defects escape. [95]
Early warning signals: large PRs with minimal discussion; “LGTM” comments on complex diffs; PR queue growth. [96]
Loop breakers: PR size caps; doc-driven review; AI-assisted PR comprehension (review maps); WIP limits on agent-created PRs. [97]
Preventative patterns: PR templates; delegate → PR model with strict scoping; plan-first. [98]
Sources: Chezo (2025-09-19); Morgan Stanley transcript (2025-10-24). [96]

Loop: Context rot (more context → worse output → more patches → even more context)
Trigger: instructions, logs, and partial plans accumulate; the model starts to contradict earlier constraints. [49]
Escalation dynamics: the human keeps “adding context” rather than pruning; output quality drops; the agent overfits to irrelevant constraints. [99]
Early warning signals: agent repeats previously fixed mistakes; inconsistent style; “why did it do that?” surprises. [100]
Loop breakers: reset session + rehydrate from thin memory; modularize rules; require plan regeneration

from current state only. [101]
Preventative patterns: instruction layering; memory bank + resets; skills for stable procedures. [102]
Sources: Cursor community modular rules discussion (2026-02-10); “AI fatigue is real…” (2026-02-08). [99]

Loop: Big-bang commit → review blockage → rework
Trigger: agent generates a massive diff after trial-and-error, collapsing work into one PR/commit. [82]
Escalation dynamics: nobody can review; feedback becomes vague; generator keeps rewriting; merge stalls; eventually a rewrite happens anyway. [26]
Early warning signals: diff in the thousands of lines; unclear intent; failing tests with many unrelated changes. [82]
Loop breakers: revert to checkpoint; slice into 30-minute chunks; commit docs + plan; enforce “one commit worth” constraints. [103]
Preventative patterns: doc-driven delivery; multi-file checkpoints; PR templates with “scope statement.” [104]
Sources: Chezo (2025-09-19); Copilot Edits guidance (updated 2025-09-22). [82]

Loop: Autonomy overshoot → security incident (or near-miss) → overcorrection
Trigger: agent uses tools (exec/network/settings edits) beyond reviewer’s awareness; prompt injection or unsafe defaults exploited. [60]
Escalation dynamics: trust collapses; teams disable agents entirely; productivity drops; shadow usage emerges. [105]
Early warning signals: unexplained config changes; unexpected outbound requests; “auto-approval” settings toggled; unreviewed instruction files. [106]
Loop breakers: least-privilege + sandbox allowlists; lock down settings; trusted-source policy for repos/rules/tools. [107]
Preventative patterns: tool gating + sandboxing; instruction file governance; MCP trust model. [108]
Sources: IDEsaster (2025-12-06); Tenable filename prompt injection (2025-11-04). [106]

Loop: “Green tests” → false confidence → semantics regressions
Trigger: agent achieves passing tests quickly; reviewers infer correctness from CI signals. [109]
Escalation dynamics: semantic bugs reach production; more defensive code added; maintainability declines; trust oscillates. [55]
Early warning signals: increased code volume per feature; extra validation layers; reviewers asking “is this overkill?” [55]
Loop breakers: strengthen tests around behavior/edge cases; require manual validation steps for risky paths; add “why this abstraction exists” notes. [110]
Preventative patterns: doc-driven delivery; PR templates; AI-assisted PR comprehension with explicit risk list. [111]
Sources: “AI fatigue is real…” (2026-02-08); “Why AI coding tools shift the real bottleneck to review” (2026-01-20). [112]

Debates and fault lines

Speed vs correctness
Pro-speed arguments emphasize that AI agents remove latency in routine work (scaffolding, refactors, tests, docs) and that humans can reserve time for high-leverage decisions—especially when background agents produce PRs asynchronously. [113]
The counter-argument is that raw generation speed can **decrease throughput** if it shifts work into review,

rework, and long-term maintenance, with practitioners describing a “quality tax” paid in cognitive load and integration friction. [114]
Where evidence lands today: teams that see durable wins typically restructure around **plan-first**, **small slices**, and **explicit verification**, rather than “generate more.” [115]

Autonomy vs accountability
Pro-autonomy advocates frame background agents as “async teammates,” with PRs as the accountability boundary: every change is inspectable, discussable, and revertible. [116]
Opponents argue that autonomy increases the blast radius of mistakes and creates ambiguity of responsibility (“who wrote this?”), especially when developers merge code they didn’t truly understand. The strongest recent pushback comes from security disclosures showing that agent autonomy can be weaponized via prompt injection and IDE features. [117]
A pragmatic middle ground emerging in practice is **autopilot-with-guardrails**: constrain what agents can do (sandboxing/approvals), keep tasks small, and keep humans accountable at merge time. [118]

Testing under high AI throughput
Optimists argue AI makes it cheap to expand test coverage and to iterate until CI is green (“fix-until-green”), raising baseline quality—particularly for well-tested systems. [119]
Skeptics argue that AI can produce *more tests* but not necessarily *stronger tests*, and that the human bottleneck shifts to understanding brittle/opaque tests or over-defensive code that passes checks but harms maintainability. [110]
Current practice trend: teams increasingly require “tests run (exact commands)” plus targeted behavioral/edge-case tests tied to requirements, rather than celebrating test volume. [120]

Maintainability and code ownership
A recurring pro-AI claim is that abstraction levels always rise (compilers, frameworks, cloud services), and “AI is the next layer,” enabling more software and keeping experienced developers focused on architecture and problem selection. [121]
The counterpoint is a lived experience of **loss of ownership**: when nobody can explain why code exists, systems become unmaintainable, and senior engineers get trapped in review/repair roles instead of design. [27]
An emerging mitigation is to shift “ownership” from writing every line to owning the **instruction layer, plans, and decision logs**—i.e., owning intent and constraints, not keystrokes. [122]

Security/privacy and supply-chain risk
Two dominant risk clusters show up in current controversies.
First, **agentic prompt injection + IDE/tool vulnerabilities**, including IDEsaster-style chains and prompt injection via filenames or instruction files. This supports the argument that AI IDEs must be “secure for AI,” and that human review alone is not a sufficient control at scale. [21]
Second, **dependency hallucination (“slopsquatting”)** and supply-chain manipulation: AI suggests non-existent or wrong packages; attackers can register them; teams install them under time pressure. [123]
Practitioner compromise position: AI can be used, but only with strict dependency verification, provenance tooling, restricted tool permissions, and “trusted context sources only” defaults. [124]

Junior learning vs skill atrophy
Proponents argue juniors can learn faster by asking AI for explanations, examples, and scaffolding, lowering the barrier to experimentation. [125]

Concerns center on “deskilling”: if juniors skip fundamentals and seniors become review bottlenecks, team knowledge sharing drops and long-term capability erodes. Practitioner accounts compare this to “GPS skill atrophy” and describe workflow changes that reduce deep focus and increase decision fatigue. [126]
A practical middle route is emerging: use AI heavily for **exploration and learning**, but require explicit artifacts (plans, tests, decision logs) and human explanations for production merges (“you must be able to explain it”). [111]

Practical playbook

This is a pragmatic “starter kit” for how teams *actually* stabilize AI-heavy implementation within ~4 weeks, based on the strongest patterns above.

High-leverage patterns to adopt in four weeks

Adopt instruction layering as the new source of truth (Pattern: Instruction layering).
Make repo-scoped instructions mandatory and reviewed—because this is where teams encode how the agent should behave. [127]

Enforce plan-first for any multi-file change (Pattern: Plan-first, code-second).
This reduces big-bang diffs and anchors review on intent rather than prose. [39]

Use background PR agents only for “delegate-ready” tasks (Pattern: Delegate → PR loop).
Start with low-to-medium complexity tasks in well-tested repos; expand only when review load is stable. [128]

Make “fix-until-green” real, but don’t worship green.
Require explicit test commands in PR templates plus manual checks for risky areas. [129]

Institutionalize doc-driven delivery for anything that needs explanation later (Pattern: Document-driven agentic delivery).
Treat short docs as the primary review artifact for complex changes. [26]

Put security guardrails in front of autonomy (Pattern: Tool gating + sandbox).
If you cannot review everything, constrain what the agent can do. [58]

Minimal guardrails that pay off quickly

1) **PR scope caps:** “one human-commit worth” per PR unless approved. [103]
2) **Mandatory PR template fields:** intent, files touched, tests run (exact commands), risks/unknowns, rollout plan for behavior changes. [130]
3) **Agent WIP limit:** maximum N open agent-authored PRs per team at once (prevents review collapse). [96]
4) **Trusted context policy:** forbid unreviewed rule/instruction files; connect only trusted MCP servers; avoid “open untrusted repo in agentic IDE.” [131]
5) **Dependency verification rule:** AI may suggest dependencies, but humans (or security tooling) must verify existence/provenance before merge. [123]

What to avoid right now (high-risk patterns)

- **"Accept All always" as a production practice.** It optimizes for throughput and systematically creates review/ownership failures. [132]
- **Big-bang PRs from agents** ("+10,000/-7,000" style diffs) without doc-driven intent artifacts. [82]
- **Running high-autonomy agents on untrusted repos or with untrusted instruction/rules files** (supply-chain/prompt injection risk). [133]
- **Unrestricted network/tool permissions** for agents "because it's faster." This directly conflicts with lessons from recent vulnerability chains. [58]

Concrete templates and example prompts

```
# .github/copilot-instructions.md (or AGENTS.md companion)

## Project intent (keep short)
- What this repo does:
- What we optimize for (performance, safety, simplicity):
- What we do NOT optimize for:

## Hard rules (must-follow)
- Do not add dependencies without explicit approval.
- Do not modify CI/config files unless the task explicitly requires it.
- Do not change public interfaces without updating docs + tests.

## Validation commands (always run before proposing final changes)
- Unit tests: <command>
- Lint: <command>
- Typecheck/build: <command>

## PR expectations
- Keep PR scope to "one human-commit worth".
- Include: intent, files touched, tests run (exact commands), risks/unknowns.
```

```
# PR template snippet for AI-heavy changes

## Intent
(What user-facing or system behavior should change?)

## Approach
(High-level approach + why this design.)

## Files touched (review map)
- <file>: <why>
- <file>: <why>

## Verification
```

```
- Tests run (exact commands):
- Manual checks performed:

## Risks / Unknowns
- Risk:
- What I did NOT verify:

## Rollout / Revert plan
(Feature flag? Migration? How to revert safely?)
```

```
# Prompt: Plan-first, narrow-scope execution

You are working in this repository under its instructions.
1) Ask clarifying questions until acceptance criteria are unambiguous.
2) Produce a plan with: files to edit, tests to add/update, and exact commands
to run.
3) Wait for approval of the plan.
4) Implement ONLY step 1 of the plan, then stop and show a diff summary + tests
run.
```

```
# Prompt: Review map + risk list for a draft PR

Summarize this diff as:
- 1-paragraph intent
- Review map: list files touched with why they changed
- Risk list: top 5 places a subtle bug/security issue could hide
- Verification checklist: exact tests + 2 manual checks
Also list anything you are uncertain about.
```

```
# Prompt: Dependency safety check (slopsquatting defense)

Before suggesting or adding any dependency:
- confirm the package exists in the registry
- confirm it is the intended package (not a lookalike)
- state why the dependency is necessary
If you cannot verify, propose an alternative that uses existing dependencies.
```

Why these templates help (in practice): they convert AI output from “persuasive text + lots of code” into an auditable workflow with explicit intent, explicit verification, bounded scope, and repeatable procedures—exactly the elements practitioners cite as missing when review fatigue and trust failures emerge. [134]

---

1 3 22 25 27 28 49 54 59 80 91 100 109 112 114 126 AI fatigue is real and nobody talks about it | Siddhant Khare
https://siddhantkhare.com/writing/ai-fatigue-is-real

2 19 23 24 125 Learnings from two years of using AI tools for software engineering
https://newsletter.pragmaticengineer.com/p/two-years-of-using-ai

4 12 42 44 45 47 102 Copilot coding agent now supports AGENTS.md custom instructions - GitHub Changelog
https://github.blog/changelog/2025-08-28-copilot-coding-agent-now-supports-agents-md-custom-instructions/

5 11 52 53 78 79 83 113 119 128 129 GitHub Copilot coding agent in public preview - GitHub Changelog
https://github.blog/changelog/2025-05-19-github-copilot-coding-agent-in-public-preview/

6 67 Specification
https://modelcontextprotocol.io/specification/2025-11-25?utm_source=chatgpt.com

7 15 21 57 58 60 61 64 65 74 105 106 108 117 124 131 133 IDEsaster: A Novel Vulnerability Class in AI IDEs | MaccariTA
https://maccarita.com/posts/idesaster/

8 New Vulnerability in GitHub Copilot and Cursor
https://www.pillar.security/blog/new-vulnerability-in-github-copilot-and-cursor-how-hackers-can-weaponize-code-agents?utm_source=chatgpt.com

9 127 Context is all you need: Better AI results with custom ...
https://code.visualstudio.com/blogs/2025/03/26/custom-instructions?utm_source=chatgpt.com

10 123 The Rise of Slopsquatting: How AI Hallucinations Are Fueling...
https://socket.dev/blog/slopsquatting-how-ai-hallucinations-are-fueling-a-new-class-of-supply-chain-attacks?utm_source=chatgpt.com

13 116 Copilot coding agent is now generally available - GitHub Changelog
https://github.blog/changelog/2025-09-25-copilot-coding-agent-is-now-generally-available/

14 89 90 Introducing Cursor 2.0 and Composer · Cursor
https://cursor.com/blog/2-0

16 Introducing the Codex app
https://openai.com/index/introducing-the-codex-app/?utm_source=chatgpt.com

17 GitHub adds Claude and Codex AI coding agents
https://www.theverge.com/news/873665/github-claude-codex-ai-agents?utm_source=chatgpt.com

18 Delegate tasks to Copilot coding agent from Visual Studio - GitHub Changelog
https://github.blog/changelog/2026-02-17-delegate-tasks-to-copilot-coding-agent-from-visual-studio/

20 31 55 56 110 Why AI coding tools shift the real bottleneck to review - LogRocket Blog
https://blog.logrocket.com/ai-coding-tools-shift-bottleneck-to-review/

26 29 30 39 82 92 94 95 96 97 104 111 115 122 134 Tackling Review Fatigue by Document Driven Agentic Coding | Democratizing Data
https://chezo.uno/blog/2025-09-19-review-fatigue/

32 33 34 36 103 GitHub Copilot Edits in Visual Studio - Visual Studio (Windows) | Microsoft Learn
https://learn.microsoft.com/en-us/visualstudio/ide/copilot-edits?view=visualstudio

35 37 multi edits scrambling scripts · Issue #6567
https://github.com/microsoft/vscode-copilot-release/issues/6567?utm_source=chatgpt.com

38 40 41 46 93 107 118 Changelog · Cursor
https://cursor.com/changelog

43 99 Documentation Page - Schema undocumented for rules
https://forum.cursor.com/t/documentation-page-schema-undocumented-for-rules/151461?utm_source=chatgpt.com

48 50 51 101 Memory Bank: How to Make Cline an AI Agent That Never ...
https://cline.bot/blog/memory-bank-how-to-make-cline-an-ai-agent-that-never-forgets?utm_source=chatgpt.com

62 68 GitHub's official MCP Server
https://github.com/github/github-mcp-server?utm_source=chatgpt.com

63 66 69 Enhancing GitHub Copilot agent mode with MCP
https://docs.github.com/en/copilot/tutorials/enhance-agent-mode-with-mcp?utm_source=chatgpt.com

70 71 72 75 76 Agent Skills
https://developers.openai.com/codex/skills/?utm_source=chatgpt.com

73 77 Agent Skills Enable a New Class of Realistic and Trivially Simple Prompt Injections
https://arxiv.org/abs/2510.26328?utm_source=chatgpt.com

81 Failed to Create a GitHub Pull Request from Cursor Web
https://forum.cursor.com/t/failed-to-create-a-github-pull-request-from-cursor-web/147959?utm_source=chatgpt.com

84 Using Cursor background agents - madewithlove
https://madewithlove.com/blog/using-cursor-background-agents/?utm_source=chatgpt.com

85 87 88 98 120 130 Copilot coding agent now supports pull request templates
https://github.blog/changelog/2025-11-05-copilot-coding-agent-now-supports-pull-request-templates/?utm_source=chatgpt.com

86 121 Will AI Replace Software Developers or Redefine Their Role?
https://www.morganstanley.com/insights/podcasts/thoughts-on-the-market/ai-replace-software-developers-sanjit-singh

132 There's a new kind of coding I call "vibe coding", where ...
https://x.com/karpathy/status/1886192184808149383?lang=en&utm_source=chatgpt.com