FIELD NOTES / 01 · APPLIED AI

A development process for AI coding agents

How goals, architecture and quality are safeguarded over time – and the role people play.



AUTHOR Sebastian Günther · IT mit Sicherheit · sg-it.info

VERSION dev-process v2.28.0 · English edition (translation of the German original)

DATE October 2026

TEMPLATE github.com/Crashman1983/dev-process

Contents

01 The challenge and the answer
02 How the process is structured
03 Rules and risk tiers
04 Review, memory, goals and measurement
05 Architecture requirements: hard and soft
06 The role of the human
07 A work item from start to finish
08 One option: GitHub with Copilot
09 Scaling to an organisation
10 Risks and current state
11 Open questions and outlook

The process was developed and refined over many iterations in day-to-day use; it has yet to prove itself at scale. One possible next step is a pilot, for instance on GitHub with Copilot. This document describes what is implemented today, distinguishes between rules that are checked by machine (hard) and rules left to judgement (soft), shows how the building blocks could be transferred to such a platform, and gathers the open questions at the end. Operational details of the reference project are examples, not requirements.

Chapter 01

The challenge and the answer

> AI agents can write good code, but they work under poor conditions: no memory, no evidence, no independent sign-off. The process changes the conditions, not the agents.

AGENT WEAKNESS	WHAT HAPPENS WITHOUT A PROCESS	HOW THE PROCESS RESPONDS
No long-term memory: after a break or once the conversation has been automatically truncated, rules and agreements are lost.	The agent breaks agreements made an hour earlier and makes settled decisions all over again, possibly differently.	The most important rules live in the start file that every agent reads first; decisions are listed in the plan. After every truncation, a program automatically feeds both back in.
Asserting instead of checking: statements about existing code come from memory.	The agent builds on a function that does not exist at all – or overlooks an existing one and writes it again. The result: duplicated code and dangling references.	Rule 1: every statement needs evidence or must be marked as an assumption. For documentation, a gate (an automated check before merge) also verifies that it references only existing files.
Patching symptoms: a bug is fixed where it becomes visible.	The cause remains; its symptom is patched separately in five places, and the defect rate rises.	Rule 6: after no more than two attempts at the symptom, the search turns to the cause. A gate flags the third fix to the same file, and the metrics show which spots keep being corrected.
Self-approval: the agent that built something also judges whether it is good.	Review becomes a formality; defects only surface in production.	Independent review by an uninvolved instance; the result is recorded as an attestation (a written review record in the journal, the running work log in the repository), which the gate requires.
Parallel work without coordination: several agents change the same files.	The last change overwrites the previous one; work is lost.	An overview of all work items in progress (the status board) shows overlaps; two efforts tackling the same problem are coordinated rather than pursued in parallel.

The essentials in five sentences

1. **A program checks the rules; nobody has to remember them.** Fifteen automated checks (“gates”) run before every merge; whatever fails is not merged.
2. **Risk determines the effort.** Four risk tiers – Tiers 0 to 3 – determine whether a change may be merged directly or needs a plan, an independent review and, at Tier 3, an additional refutation review (a targeted hunt for defects).

3. **Review is always independent.** Nobody signs off their own work; from Tier 2 upwards, an uninvolved instance (a separate agent session) carries out the review, and its verdict applies only to the exact code it reviewed.
4. **All knowledge lives in files.** Plans, decisions and journals are kept in the repository and are automatically fed back to every agent as soon as its memory has been truncated.
5. **People decide, agents work.** A coordinator agent assigns the work to worker agents; the human prioritises, decides, approves designs and reviews a weekly sample.

Chapter 02

How the process is structured

> Five layers, each with a clear responsibility. People judge, agents work, programs check, files preserve the knowledge.

LAYER	WHAT IT DOES
Human (owner)	prioritises, decides, approves designs, reviews a weekly sample, keeps the rules up to date
Coordinator	maintains the overview, assigns work items, starts and stops workers, forwards questions to the human, triggers the merge – writes and reviews no code itself
Workers and reviewers	a separate session on its own branch for each work item and phase; the reviewer is always a different instance from the worker
Gates	fifteen check programs before every merge: rules intact, decisions made, attestation present and matching the code, design contracts and acceptance criteria consistent, licences permitted, documentation valid
Repository	rule core, plans with decision lists, journal with attestations, status board, contracts, metrics – the single source every layer reads from

The process is distributed as a template (dev-process), from which a fully configured repository is generated: start file and phase commands for the chosen harness (GitHub Copilot, Claude Code or a neutral AGENTS.md), the gates, and GitHub Actions workflows for gates, owner digest and metrics. A single command pulls later versions of the template into a repository; project-specific files are left untouched.

Chapter 03

Rules and risk tiers

> Nine rules always apply; how much process a change needs depends on its risk, not its size.

The nine rules appear in every start file, and a gate checks them character by character against the original:

#	RULE	#	RULE
1	Evidence before assertion – every statement about code is backed by evidence or marked as an assumption	6	Cause before symptom – after no more than two attempts at the symptom, the cause is sought

#	RULE	#	RULE
2	Plan before work – from Tier 2 upwards, a written plan before any code is written	7	Review before merge – as independent as the risk demands
3	Contract first – interfaces and UIs are specified in binding form before any code is written	8	Small, clearly named changes – exceptions are stated in the commit
4	One responsible component per behaviour – no copies; hard-to-reverse decisions are documented before code builds on them	9	Readable code – written for the next reader
5	Tests prove acceptance – every criterion has a test		

The four tiers. What matters is the risk, not the scope: a ten-line change that other components depend on is Tier 2; a formatting change is Tier 0. The assigned tier may only be corrected upwards. It rises to at least Tier 2 as soon as the answer to any of these questions is yes: Is data stored permanently? Does user input leave the process? Is access control involved? Does other code depend on the interface? Is more than one user interface affected? Is data loss possible? Is the same spot being changed repeatedly? Tier 3 applies when the change touches access control, data storage and migrations, a security boundary or a contract spanning several repositories.

TIER 0	TIER 1	TIER 2	TIER 3
No change in behaviour, or local and reversible	Small, self-contained change	Something others depend on: interface, schema	Access control, data storage, security boundary
Merge directly	Fast track (goal, files and risk in one sentence, no written plan) plus one test	Plan, implementation, independent review	In addition: approved design, threat question, refutation review, second model (where available)

Refutation review means that at Tier 3 an additional instance reviews the change with the sole brief of finding defects. **Second model** means a model from a different vendor, where one is available; if none is, the attestation says so explicitly. The gate does not require a second model, but it does require its absence to be stated openly.

Chapter 04

Review, memory, goals and measurement

> A review verdict counts only if it was reached independently and is bound to exactly the code reviewed. What agents need to know – decisions, goals, metrics – lives in files and is fed back to them after every truncation.

Review. From Tier 2 upwards, an uninvolved instance receives a read-only bundle – changes, plan, tests, images – but not the conversation in which the work was done. Its attestation in the journal names the work item, tier, reviewer, model, independence attributes, verdict, round and the checksums of the code. The gate checks whether an attestation exists, whether its attributes match the tier and whether it belongs to the merged code. What it cannot check: whether the reviewer really was a different instance. That relies on the reviewer's own declaration – hence the weekly sample taken by the human.

Memory. The plan with its decision list, the journal and the task list live in the repository. From them, a single command derives what is in progress, what comes next, which decisions apply and which question is open. After every truncation of a session, a program automatically feeds the nine rules, verbatim, and the last twelve decisions back in. This was prompted by an observation in operation: the bare instruction to re-read the rules after a truncation was itself lost in the truncation.

Goals. A work item does not start until its type is settled and its acceptance criteria are written as testable sentences ("When <trigger>, the system shall <response>", including negative, edge and permission cases). It is accepted only when every criterion has a passing test, all gates are green and the documentation has been brought up to date. Both sets of conditions are kept as lists in the repository (the start list and the acceptance list) and are continually extended: if a human says "in the end, X must hold", X is added.

Measurement. A cockpit (an analysis script in the repository) reads the journal, the Git history and the status reports; every figure states how meaningful it is. Four metrics matter most:

- **Review rounds to approval.** Target: at most two for 90% of work items. If a work item is still rejected in round 3, a rule is recorded or the work item is split.
- **Fix rate.** Share of features that had to be corrected within seven days; only the trend counts.
- **Fix clusters.** Spots that keep being corrected – Rule 6 in numbers.
- **Review rounds per model.** Shows which model performs well in which phase.

Chapter 05

Architecture requirements: hard and soft

> Requirements hold not because someone knows them but because a gate checks them ("hard"). Where only a reviewer judges ("soft"), the table says so explicitly.

REQUIREMENT	MECHANISM	HARD / SOFT	REFERENCE PROJECT
Decisions before code	Hard-to-reverse decisions are documented before code builds on them; a gate checks whether required decisions have been made. Rules that must apply everywhere get a responsible component and a test covering all known cases.	hard: required decisions made; soft: whether one is needed at all	73 decision records
Interfaces first	An interface is specified before anyone uses it (Rule 3). Tests and reviewers check whether the specification exists and the code adheres to it.	soft: review and tests	active
User interfaces by contract	A design contract specifies spacing, colours and states by ID; reference images are sealed with a checksum. A gate checks IDs and seals; the reviewer compares the result against the image.	hard: IDs and seals; soft: appearance	active
Layers, dependency direction	The architecture description specifies which layers may not use which others. Only an architecture linter can check this by machine.	hard only with a linter	linter not set up

REQUIREMENT	MECHANISM	HARD / SOFT	REFERENCE PROJECT
Security	Tier 3 requires a threat question, a refutation review and – where available – a second model. In addition, there is an SBOM (a list of all third-party components) whose licences a gate checks against an allow list.	hard: Tier 3 procedure, SBOM licences	active; each project creates its own licence list
Performance	The review checklist asks about performance. What is not measured cannot fail a check.	soft	no target values defined
Maintainability, documentation	The reviewer assesses maintainability against Rules 4, 6 and 9; the metrics show repeated fixes. For documentation, gates check whether the files and references it names exist.	soft; hard for references	active
Legacy	Existing violations are recorded in a baseline and tolerated; new ones are not. The baseline may only shrink.	hard, once a baseline exists	procedure in the template

Life cycle of a requirement: a new requirement begins with a decision record. Whatever can be checked by machine becomes a forbidden dependency in the architecture linter or a test covering all known cases; the rest becomes a question in the review checklist. Existing violations go into a baseline that may only shrink. A new violation fails the gate; an exception is possible only as a decision record with an expiry date. New non-functional goals take the same route and also get an entry in the acceptance list.

Chapter 06

The role of the human

> The human writes no code and does not read every change. What remains is judgement – at five points no agent may take over.

The owner's five tasks

- **Prioritise.** The owner decides what is released next. If an agent creates dozens of work items, work begins only once the owner has sorted them.
- **Decide.** Every product, architecture or risk question arrives as a choice with options and a recommendation; the answer becomes a decision line in the plan.
- **Approve designs.** At Tier 3, nothing is built until the design has been approved. The merge itself does not need the owner's consent; gates and attestation take care of that.
- **Review a sample.** Once a week, the owner reads one merged work item thoroughly; the calendar week determines which one. No agent knows in advance which one will be picked.
- **Keep the rules up to date.** The start list and the acceptance list are continuously extended: an owner's requirement becomes a permanent part of the process, not just a one-off.

The owner works through their harness's interface: on GitHub, for instance, via issues, pull requests and the owner digest generated by a workflow; in the reference project, via a chat app, including on mobile. What the owner no longer does: remind anyone of the rules, ask for progress updates, merge, start sessions (in the reference project; still open on GitHub), tidy up.

Chapter 07

A work item from start to finish

> Each phase runs in its own session with the model that the model policy (a configuration file) specifies for the tier and phase. The phases follow one another without delay; the only waiting is for human decisions.

READY	PLAN	IMPLEMENTATION	REVIEW	MERGE	DEPLOY
Owner releases	Plan + decisions	Test first, one commit per task	Uninvolved instance, attestation	Merge queue: gates and tests run once	One deploy per queue run

Questions. If a question needs the owner's decision, the worker enters it in the plan with options and a recommendation and reports "blocked". The coordinator presents it to the owner as a choice, the answer is recorded as a decision in the plan, and the worker carries on.

Merge. Approved branches are collected in a merge queue, run through the gates and the test suite together once, and merged in order. If the joint run fails, the branch responsible is identified, removed from the queue and handed back to its worker.

Coordinator failure. Nothing is lost: the state lives in Git; workers stop when they need a decision; a new coordinator has caught up on the state within a minute.

Example from the reference project: a Tier 2 work item – plan at 21:18, two questions to the owner during implementation, review in three rounds (Block in round 1), merge at 04:24. Seven hours, two human decisions, no other manual steps.

Chapter 08

One option: GitHub with Copilot

> Rules, gates, journal and metrics are files and scripts built on Git. This is why the process can be transferred to a platform such as GitHub: it could run the gates, track work items as issues and collect merges in the merge queue; Copilot would read the rules like any other harness.

The following table shows how the building blocks could be mapped and which of them are already part of the template. It is one route, not a commitment.

BUILDING BLOCK	POSSIBLE GITHUB MECHANISM	STATUS
Start file, rule core	"copilot-instructions.md" with the checked rule block; path-specific rules under ".github/instructions".	in the template
Phase commands	Prompt files under ".github/prompts" for brainstorm, plan, implementation, review, fast track, debug, commit, recovery.	in the template
Gates	Actions workflow on every pull request, registered as a required status check in branch protection; a script sets it up without overwriting existing rules.	in the template

BUILDING BLOCK	POSSIBLE GITHUB MECHANISM	STATUS
Work items	Issues with type and acceptance criteria; a gate checks that plans name their issue and that Tier 3 work does not begin without an issue.	in the template
Workers	One conceivable option is a Copilot session per issue on its own branch – in the editor or as the Copilot coding agent, which opens a pull request.	planned, untested
Review	Uninvolved reviewer with review prompt and bundle, attestation in the journal; Copilot code review as an additional voice. Since Copilot offers models from several vendors, the second model would be just a configuration entry.	prompt and gate in the template
Merge queue	The merge queue could collect approved pull requests, check them together and merge them in order.	GitHub feature; untested (reference project: its own merge train)
Overview and clean-up	Owner digest and metrics run as Actions workflows; a script (tidy) cleans up finished branches, specs and journals.	in the template
Coordinator	Overview and status reports can run as Actions. What remains open is who starts and stops sessions on GitHub.	open (Chapter 11)

What would change: in the reference project, the gates run on a single server; on GitHub, they would run within the organisation; with branch protection, only an administrator could bypass the gates; and nobody would need to switch harnesses any more to get the second model at Tier 3. A pilot is to show whether this holds up in practice.

Chapter 09

Scaling to an organisation

> The unit of the process is the repository. Machines scale along with it; the limit is the number of decisions an owner can make.

What scales: every repository receives the template with its own gates, its own model policy and its own status board. The template is versioned centrally and contains organisation-wide rules such as layering rules or the permitted licences; an update arrives in every repository as a pull request. Gates run centrally, for instance in GitHub Actions, and scale with the organisation. Workers are sessions of the harness, for example one per issue, and the model policy specifies not only the models but also how many of these sessions run concurrently.

What does not scale by itself: every repository needs an owner, and every work item needs on average one to two of the owner's decisions. How many parallel work items an owner can handle is the real capacity limit – and it is not measured today. It is equally open where the coordinator runs in a larger environment and whether one coordinator can manage several repositories.

What changes for developers: they take on the owner role for a repository, review samples and write acceptance criteria, contracts and decisions. The role profile, training and deputy arrangements have yet to be described (Chapter 11).

Chapter 10

Risks and current state

> A process that demands evidence for every statement names its own limits. The figures are a baseline, not a verdict.

RISK	WHAT IT MEANS	CURRENT COUNTERMEASURE
Gates can be bypassed	Without branch protection, a merge can skip the gates.	Every bypass must be named in the commit; a Tier 3 work item that lands on main without review retroactively turns the gate red; branch protection makes the gate mandatory; the owner takes samples.
Attestation is self-declaration	The gate cannot verify whether the reviewer was a different instance.	Review sessions are started by a program, not manually by a human; the attestation is bound to the code; the owner takes samples.
A single model family	In the reference project, two thirds of Tier 3 reviews ran without a second model; the gate then only requires an explicit statement.	The refutation review remains; platforms such as Copilot offer models from several vendors; whether the gate should require the second model is still open.
Model policy is code	The file also contains the start command for sessions; whoever changes it can run commands on the machine that starts them.	Treated like CI configuration: every change goes through review.
Vendor lock-in	The coordination layer (status board, coordinator, merge train) has so far been tested only with Claude Code.	The process files are vendor-neutral; the template ships with adaptations for Copilot, Claude Code and AGENTS.md.
A single human	If the owner is unavailable, decisions stall.	All knowledge lives in files; a deputy arrangement is not yet in place.
Young coordination layer	Status board, coordinator and merge train (the reference project's own merge queue) are new and were reworked eight times in quick succession.	Two independent reviews with 45 findings, all addressed; intention: one week of operation before every extension.

METRIC (REFERENCE PROJECT)	VALUE	CONTEXT
Commits, 14 days	1,505	of which 537 documentation and process, 316 fixes, 243 tests, 190 features, 219 other
Review verdicts, approx. 4 weeks	611 Pass / 211 Block	Reviews are finding defects. The Block rate has risen (36% in the more recent half of the period, 24% before).
Work items with more than 2 rounds	82 of 396 (21%)	Target: at most 10%. From round 3, a rule is recorded or the work item is split.
Fix rate (features corrected within 7 days)	44.4% (20 of 45 features)	DORA band "medium", close to "low"; only the trend over three periods will be meaningful.
Fix cluster	11 fixes in 5 places	A business rule in the code was patched one place at a time; it is now getting a responsible component and a test.

The core – gates, review, rule core, journal, design contracts – is stable; whatever did not hold up in the reference project has been removed. The coordination layer is the youngest part. Its eight rapid reworkings have led to an intention, not yet a rule: one week of operation before every extension, and changes to the coordination layer go through the same tiers as product code.

Chapter 11

Open questions and outlook

> Whatever remains unanswered today is listed here as a question, with its next step.

AREA	QUESTION	NEXT STEP
Organisation	How many parallel work items can an owner handle?	In the pilot, measure decisions per work item and time spent waiting for the owner; derive the number of repositories per owner from that.
Organisation	Where does the coordinator run, and can one manage several repositories?	Run one coordinator per repository in the pilot; then trial one coordinator across two repositories.
Organisation	What becomes of the developers? Who deputises for the owner?	Describe the role profile and training; define deputy arrangements covering questions, sampling and model policy.
Organisation	How does a template change propagate to many repositories?	Send updates automatically to every repository as a pull request; set up a second repository from the template.
Architecture	Should the gate require the second model at Tier 3?	Introduce a rule: if the model policy provides for a second model, the gate requires it too.
Architecture	Is the dependency direction checked by machine? Is there a minimum security standard? How is performance safeguarded?	Set up an architecture linter; create a security rule set; define performance targets as a test and as an entry in the acceptance list.
Architecture	How are exceptions and legacy code handled?	Document every exception as a decision record with an expiry date and have a gate flag expired ones; create a baseline per repository that may only shrink.
Process	Does the process hold up on another platform, such as GitHub with Copilot?	Trial workers per issue and the merge queue in a pilot repository and compare the results with the reference project.
Process	What kinds of defects occur?	Tag every Block verdict with a keyword so that the cockpit counts defect types, not just numbers.

What this document does not claim: that the process is finished. What it does claim: that every rule is either checked by a program or explicitly marked as a matter of judgement, that every figure states how meaningful it is, and that the open questions are known. That is the starting point for a pilot.

The process template is public and available at: github.com/Crashman1983/dev-process

About this version

This English edition is a translation of the German original and is based on the documentation of dev-process v2.28.0. The procedures, measurements and open questions described refer to that state. The project continues to evolve.

Current project status: github.com/Crashman1983/dev-process
Web version: sg-it.info/artikel/entwicklungsprozess-mit-ki-agenten/

© 2026 Sebastian Günther · IT mit Sicherheit · 11 pages

AUTHOR
Sebastian Günther

WEB
sg-it.info

How could AI fit into your development processes? Let's talk about it.