

Nessie for Digiteq Automotive

A development studio where the code never leaves the building.

In one paragraph

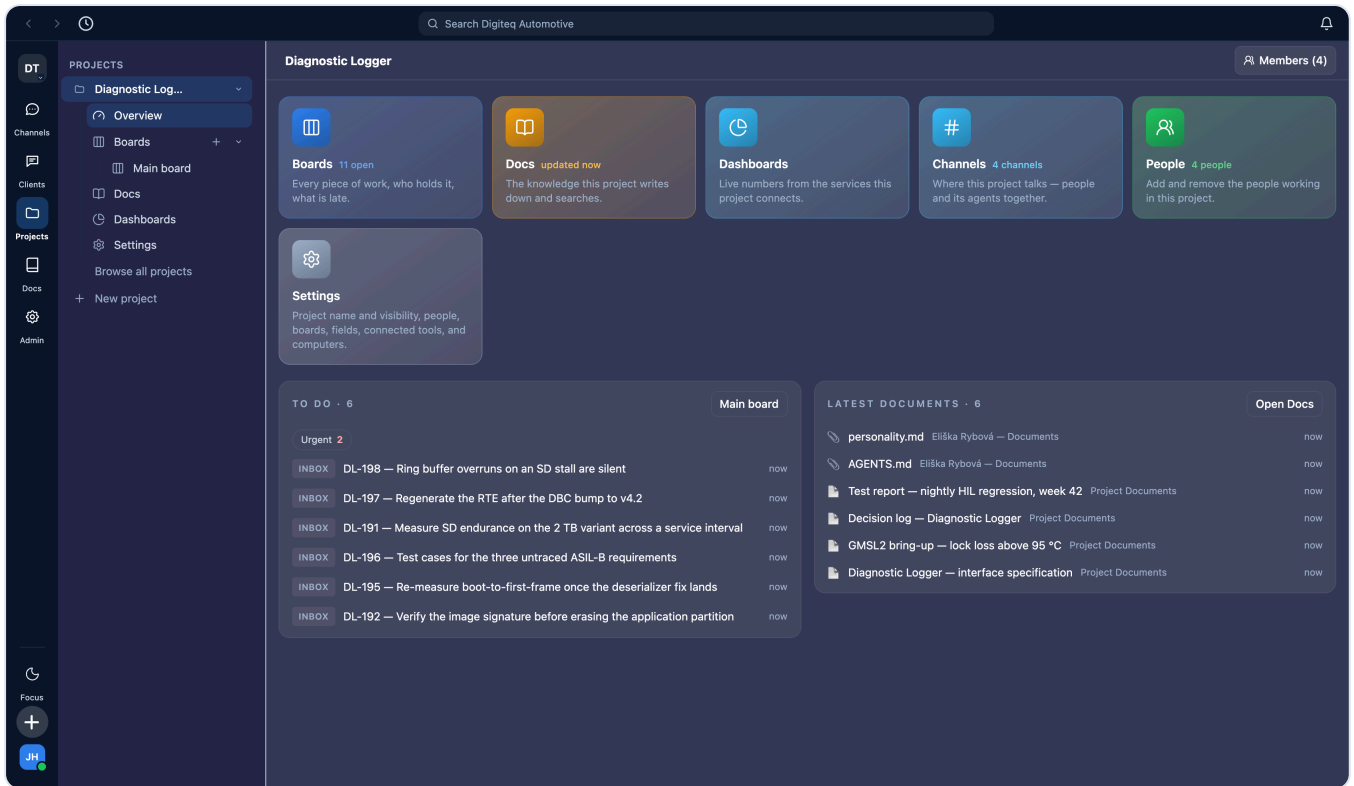
Nessie is three things in one system: a project-management tool, a project-development tool, and a software-development tool. Work is planned as tickets on a board. The people doing that work are your engineers and a set of AI colleagues who sit at the same level as any other member of staff — they have names, job titles, a person answerable for them, their own email addresses and their own phone numbers. When a ticket moves into development, the actual compilation and editing happen in a Claude or Codex session on a machine you own: an engineer's Mac, a Windows bench box, a Linux HIL host. Source code never leaves your network. Everything every agent reads or writes is gated by project access and recorded in an audit trail that names the actor.

The claim this document makes is that a team of ten can carry the work of a hundred. The mechanism is in the sections below, the evidence is in the screenshots, and the cost — which is real — is in the last section but one.

About the screenshots. Every picture in this document was taken from a running Nessie instance, by an automated capture run against a live database. The project, the people, the supplier and the part numbers are invented and carry an `.example` domain. No Digiteq data was used and nothing was mocked up: if a screen in this document shows a feature, the feature exists and renders exactly like that. Where something could not be captured, this document says so rather than drawing it.

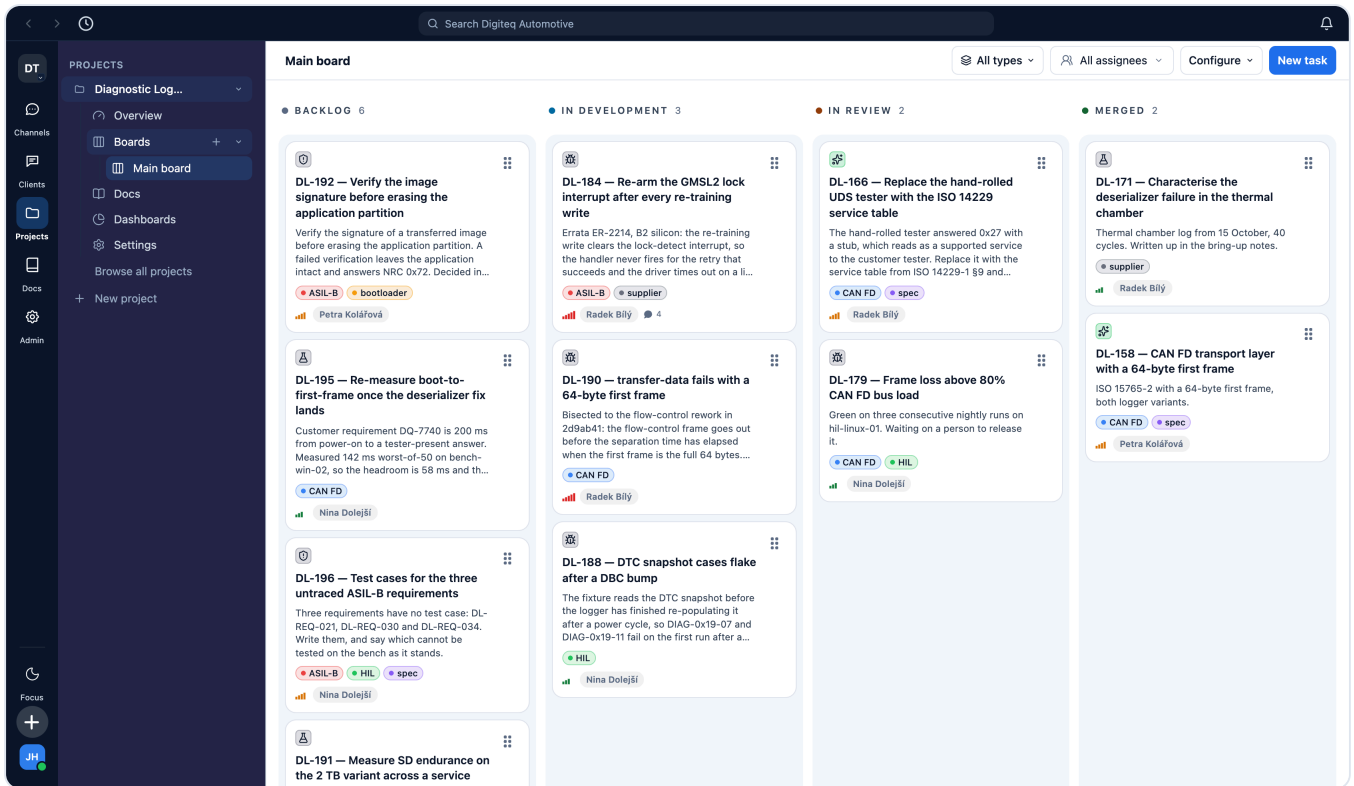
1. The project is the unit of work

Everything hangs off a project: its boards, its documents, its channels, its people, and the machines it is allowed to use. A person with access to the project sees all of it. A person without access sees none of it — including an organisation owner who is not in the project's team, because teams in one organisation are wholly independent of each other by design.



The project's own front page: boards, documents, dashboards, channels, people and machines in one place, with what is open and what changed most recently.

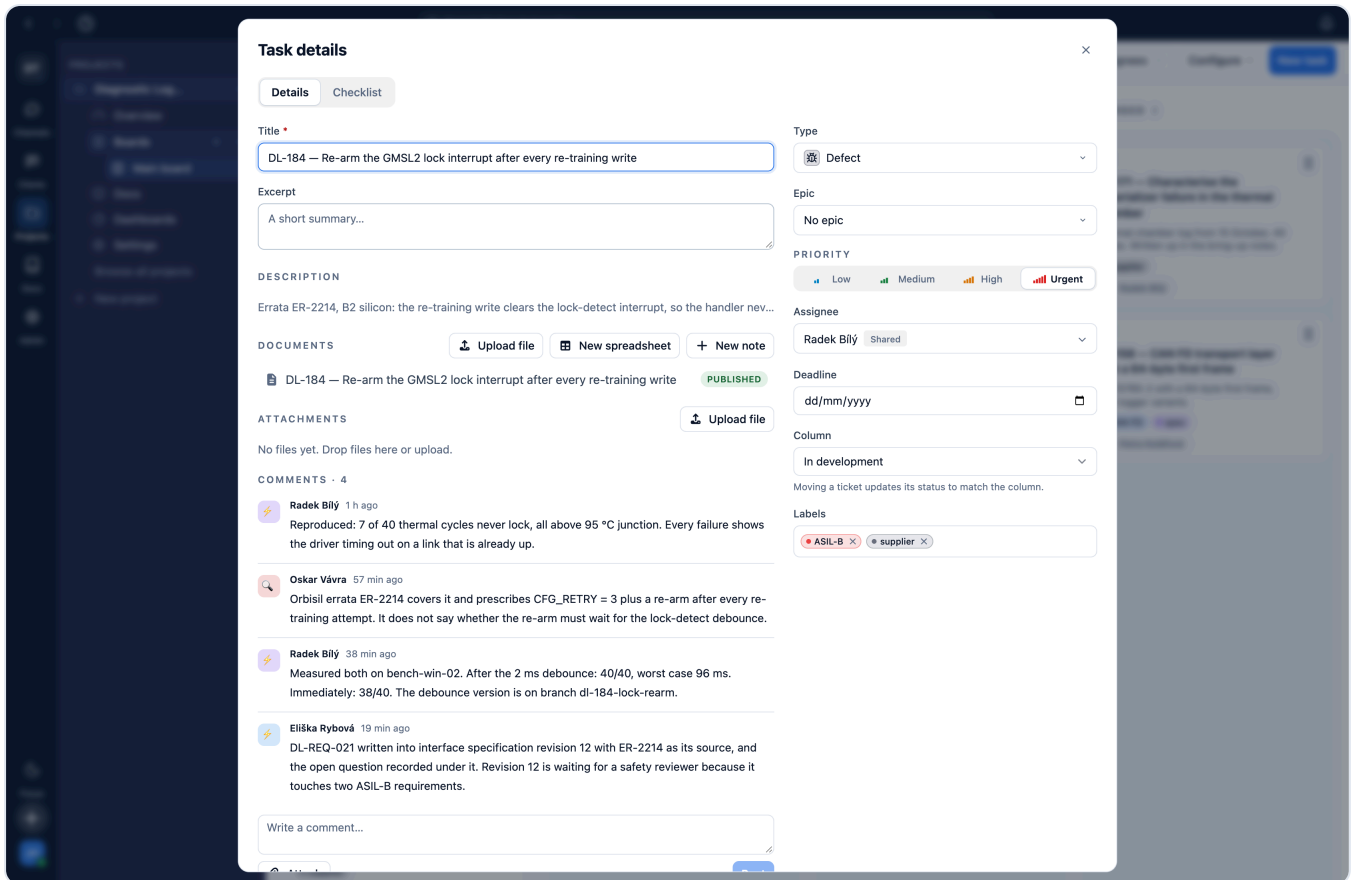
The board



An ordinary Kanban board. Tickets carry a type, labels, story points and an assignee — and the assignee is a person on some cards and an AI colleague on others. The board is configured per project: these four columns were named by the team, and each one is bound to a lifecycle category so that moving a card updates the ticket's status.

A board column can be configured to **start work**. Drag a ticket into *In development* and it is picked up by the agent that holds it, on the machine that project is allowed to use. That is the whole interaction: the planning gesture an engineering manager already makes is the gesture that dispatches the work.

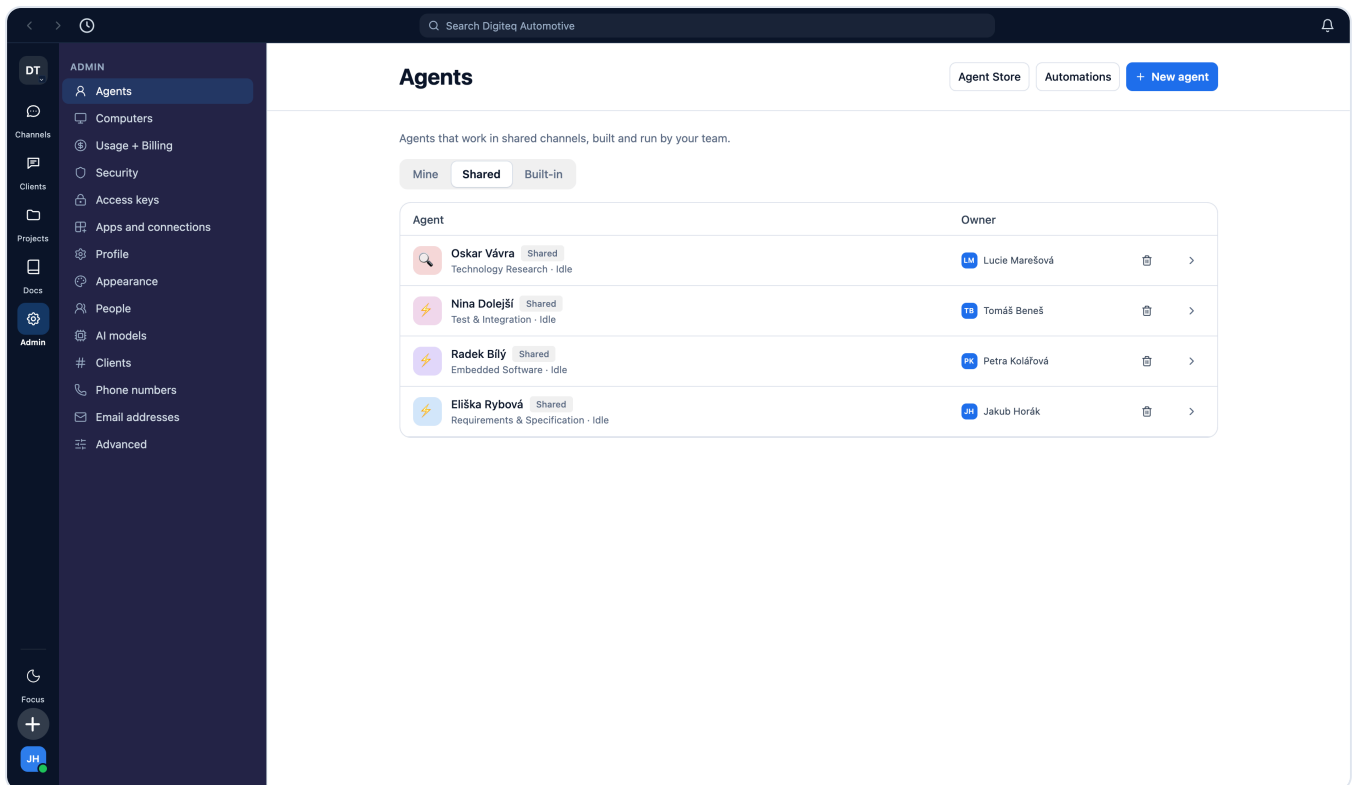
The ticket



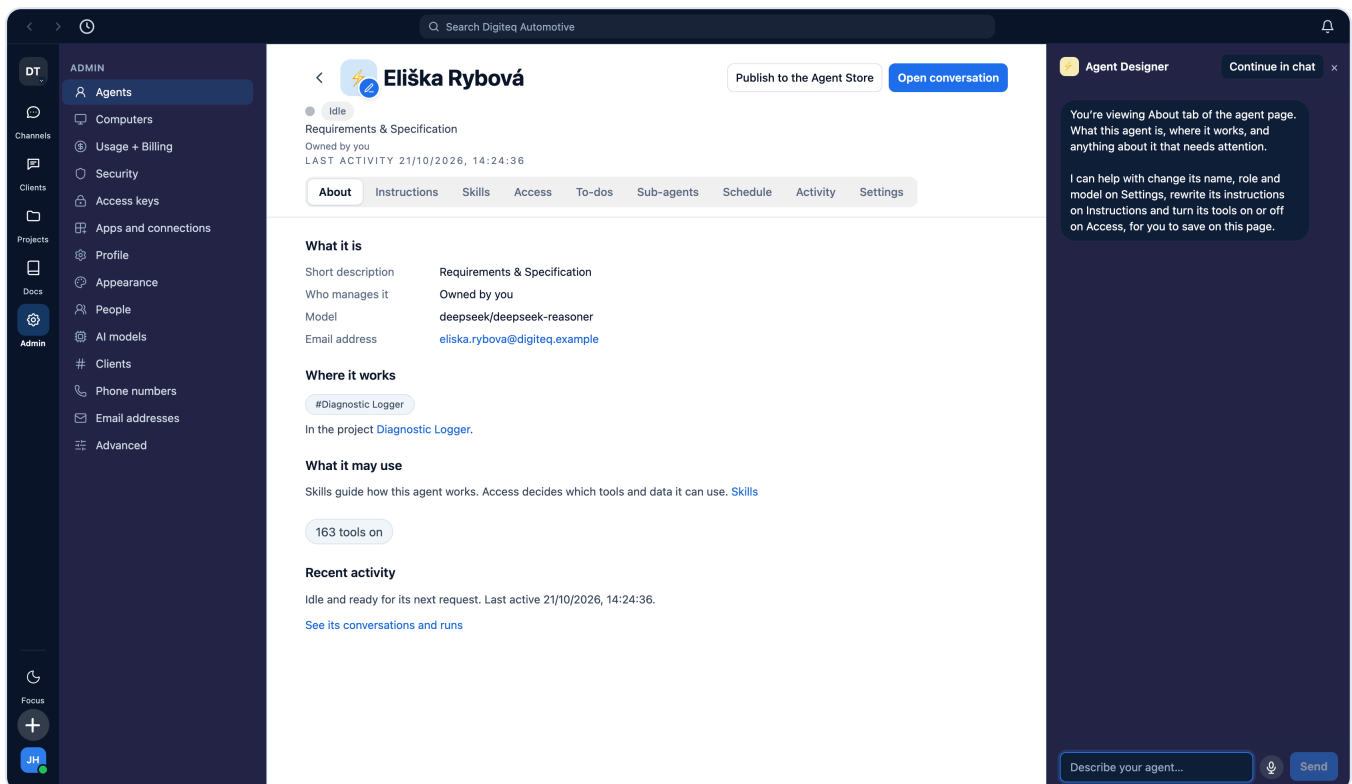
One ticket. Description, checklist, type, priority, labels, assignee, column — and a comment history in which the engineer who measured the problem and the agents that found and specified it appear side by side, each by name. The comments here were written by three different agents; the product does not need a person to transcribe anything.

2. Agents are staff, not a chat box

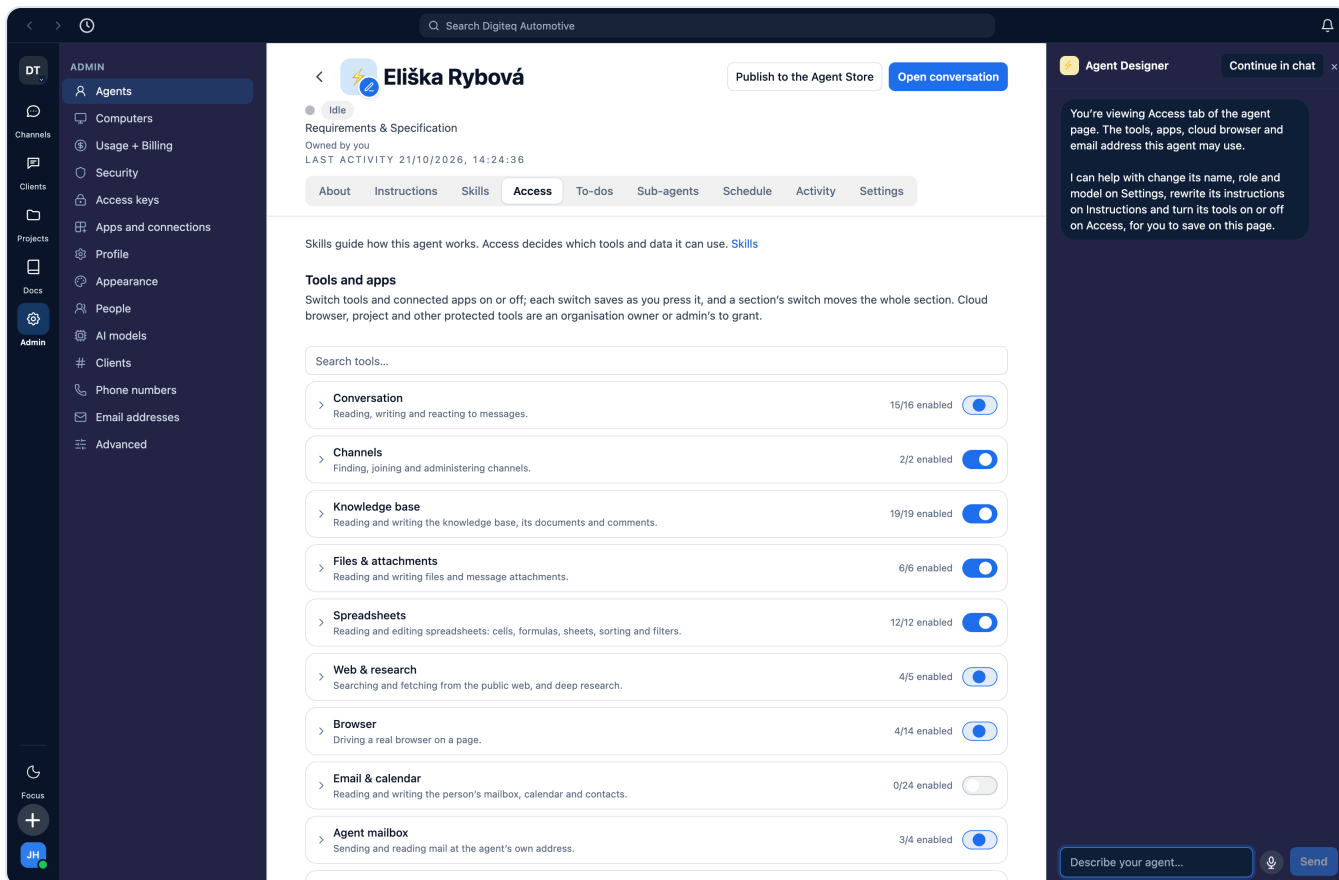
The easy thing is to read past this, so plainly: **an agent in Nessie is a record of the same kind as an employee.** It has a name and a job title. It has an owner — a named person who is answerable for it. It belongs to a team and to projects. It appears in the roster. It is in some rooms and not others, and what it may read is decided by the same access rules that decide what a person may read.



Four agents, each with a role and a named human owner. This is a roster, not a model picker.



What one agent is: its job, its owner, the model it runs on, its email address, the rooms it works in and the project it belongs to. The panel on the right is the Agent Designer — see section 7.

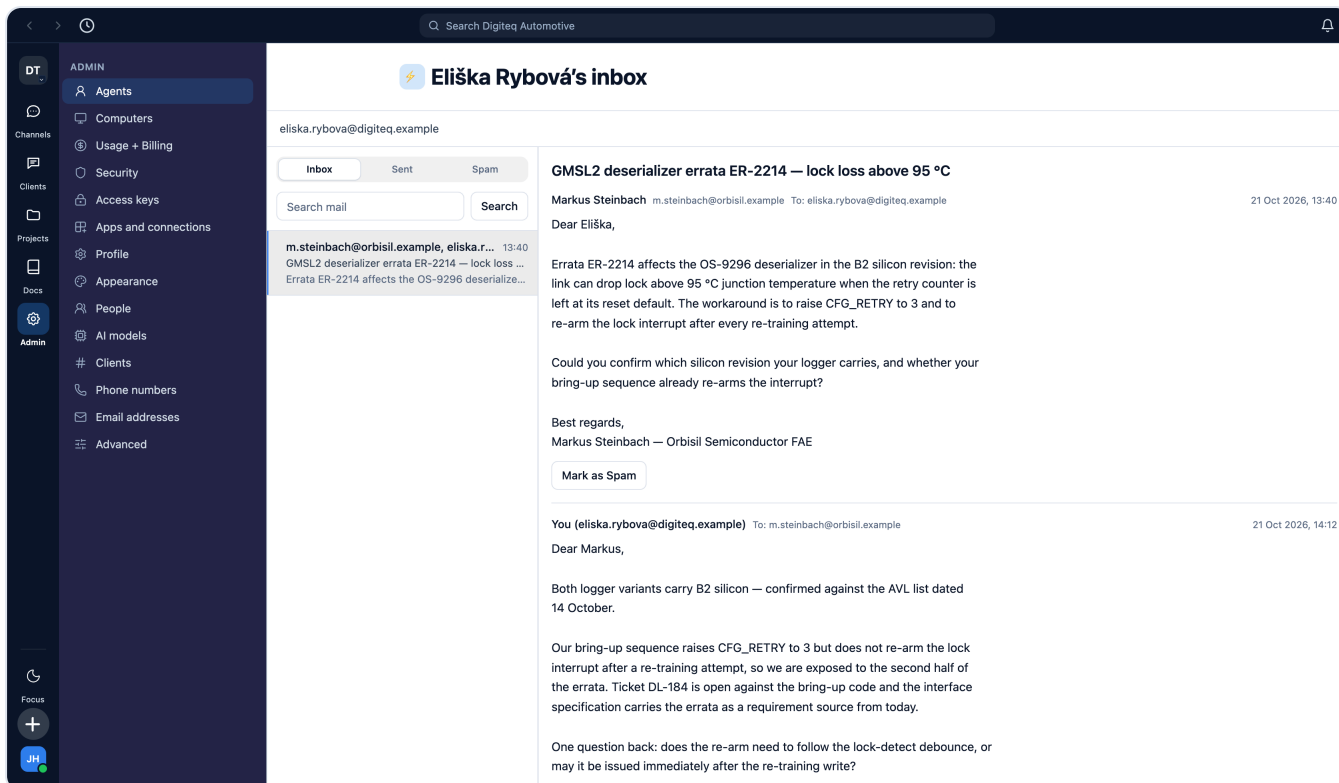


And what it is allowed to do. Every tool and every source of data, grouped, counted and switchable. "Email & calendar: 0/24 enabled" is not decoration — that agent genuinely cannot read anyone's mailbox. Cloud browser, project and other protected tools can only be granted by an organisation owner or admin.

3. Reachable the way a colleague is

Because an agent is a member of staff, it is reachable like one.

It has its own email address. Mail arrives at that address from outside the company and is answered from it.



A supplier FAE writes to the specification engineer's own address about a silicon errata. The reply names our silicon revision and an open defect, so it is held for a person to read before it leaves the company — that is the default for an agent mailbox, not a setting someone remembered to turn on.

You can put an agent on a mail thread the same way you put a colleague on one. Add the agent's address to a thread with a customer, a supplier or another part of the Group, and say *Eliška will take this one*. That sentence is the grant: from that moment the agent has the thread, the context it needs to answer, and the boundary of what it may say — because what it may read is already decided by its project access, and what leaves the company is already gated by the approval rule on its mailbox. You did not configure anything. You did what you would have done with a junior engineer.

It has a phone number, and that changes the day. Press the call button in a conversation and you talk to the agent. Give it a real number — bought inside Nessie from the telephony provider, Czech and UK numbers in the current rollout — and the same colleague is reachable from any handset: it answers SMS on that number, and telephone calls run over an admitted Twilio bridge. (*WhatsApp exists in the data model but is not yet connectable; the product's own dialog says "Not available yet." It is not claimed here.*)

Why this matters more than it sounds

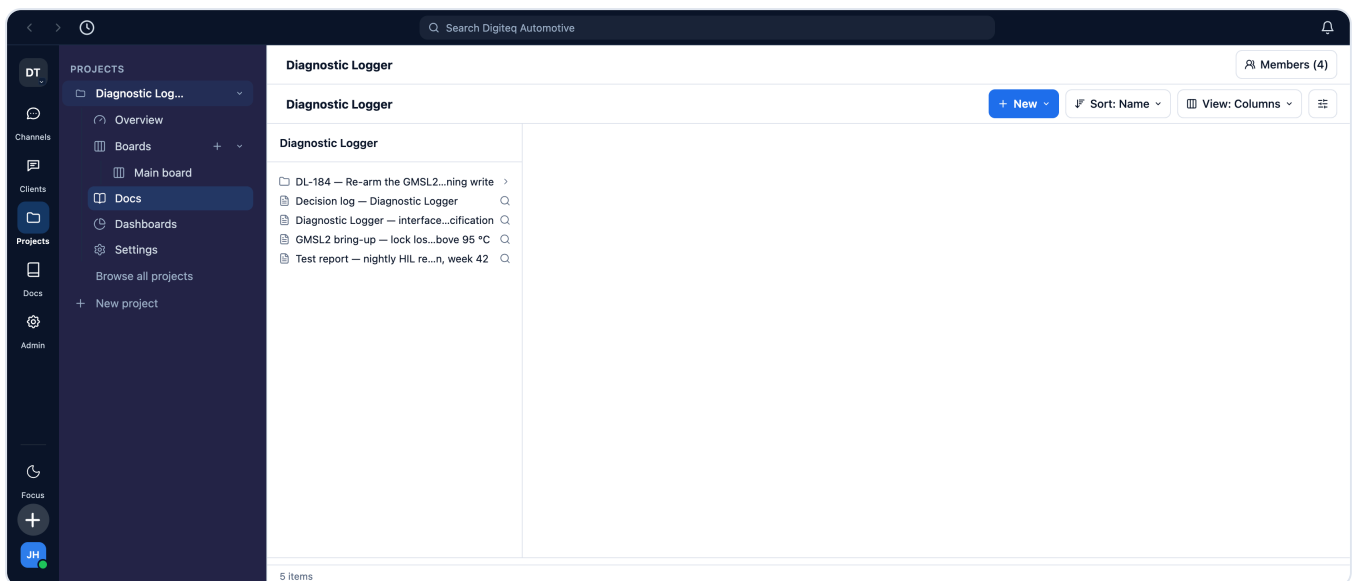
The honest argument for this is not capability, it is quality of life. An engineering manager spends a large part of the week being the transport layer between people — repeating a decision to someone who was not in the room, chasing a status, writing up what was agreed, forwarding a supplier answer to whoever needs it. All of that is work that has to happen and that nobody enjoys.

With agents that are reachable like colleagues, that work can happen while you are driving to a meeting. You call the agent on the way, in Czech, the way you would ring a colleague, and ask where the logger build

stands. It tells you, and it tells you what it is blocked on. You say *tell Petra the thermal slot moved to Monday and put it on DL-195* — and it does both: the message and the ticket. You ask it to reply to the supplier confirming the silicon revision, and the draft is waiting for someone to release when you arrive. Nothing is sitting in your head waiting for you to reach a keyboard, and nothing has been written down twice. The week stops being a relay race.

4. The work is written down where the work is

A project's documents live inside the project. They are not a wiki somewhere else that a person is supposed to keep in step by hand.



The documents of one project: the interface specification, the bring-up notes, the decision log, the test report — and a folder of documents belonging to one ticket. Each one is readable by exactly the people the project is readable by.

The important property is not that agents can write documents. It is **what they write**. An agent that sat in a thread, or listened to a meeting, does not produce a transcript. It works the content into the document that owns it: a decision becomes a requirement with a source, a measurement becomes a rationale, a disagreement becomes a ticket.

Diagnostic Logger — interface specification

Let Diagnostic Logger read this

Revision 12. Written by Eliška Rybová against commit 8f41c20 of the logger firmware and the Orbisil errata sheet dated 17 October. Every requirement below names the source it came from; a requirement with no source is a defect in this document.

Scope

Unified Diagnostic Services over CAN FD for the fleet data logger, both variants. Transport is ISO 15765-2 with a 64-byte first frame. Out of scope: the Ethernet diagnostic path, which the gateway owns.

DL-REQ-014 — Service table (ASIL-B)

The logger shall answer services 0x10, 0x11, 0x14, 0x19, 0x22, 0x2E, 0x31, 0x34, 0x36, 0x37 and 0x3E. Any other service identifier shall be answered with NRC 0x11. Source: ISO 14229-1:2020 §9, customer diagnostic requirement DQ-7731.

Rationale for the change in this revision: the hand-rolled tester accepted 0x27 and answered it with a stub, which read as a supported service to the customer tester. Ticket DL-166.

DL-REQ-021 — Deserializer lock supervision (ASIL-B)

After any GMSL2 re-training write, the bring-up sequence shall re-arm the lock-detect interrupt before the next frame is accepted. CFG_RETRY shall be 3. Source: Orbisil errata ER-2214, confirmed by the FAE on 17 October for B2 silicon.

Open: whether the re-arm must follow the lock-detect debounce or may be issued immediately after the re-training write. Asked of the supplier on 17 October; ticket DL-184 is blocked on the answer and the safe implementation (wait for debounce) is in the branch meanwhile.

DL-REQ-022 — Boot to first frame

The logger shall answer a tester-present request within 200 ms of power-on. Measured on bench-win-02 against commit 8f41c20: 142 ms, worst of 50 runs. Source: customer requirement DQ-7740.

DL-REQ-030 — Flash integrity (ASIL-B)

The bootloader shall verify the signature of a transferred image before erasing the application partition. A failed verification shall leave the application partition intact and answer NRC 0x72. Source: internal safety concept SC-4, hazard H-11.

Decided in the design review happens in the bootloader rat

Edit

Add attachment

🔍

...

→ Marešová): verification

.B of flash, because an

application that verifies its own success cannot protect the code where the application is already corrupt

Revision 12 of an interface specification, written by an agent. Every requirement names the source it came from — a standard clause, a customer requirement, a supplier errata — and what is still open with the supplier is written down as open, with the ticket that is blocked on it. For an ASIL-rated work product, "where did this requirement come from" is the question an auditor asks; the document answers it because the agent that wrote it had both the thread and the errata.

Decision log — Diagnostic Logger

Let Diagnostic Logger read this

Decisions taken on the Diagnostic Logger, in the order they were taken, with who took them. Written from the design review on 16 October and the standing weekly; the meeting notes themselves are not the record — this is.

17 October — Debounce before re-arm, pending the supplier

Decided by Petra Kolářová. The deserializer re-arm waits for the 2 ms lock-detect debounce rather than following the re-training write immediately. Reason: 40 of 40 thermal cycles pass that way and 38 of 40 the other way. To be revisited when Orbisil answers; DL-184 carries the question.

16 October — Signature verification stays in the bootloader

Decided by Petra Kolářová, with Tomáš Beneš and Lucie Marešová present. Costs 9 KB of flash. Reason: an application that verifies its own successor cannot cover the case where the application is already corrupt. Written into DL-REQ-030.

16 October — SD wear levelling: supplier firmware, not ours

Decided by Lucie Marešová. We take the card vendor's wear levelling and record the endurance assumption in the specification rather than writing our own layer. Reason: our own layer is a new ASIL-B work product for a problem the card already solves, and the fleet replaces cards on a service interval anyway. Dissent recorded: Tomáš Beneš expects the assumption to fail on the 2 TB variant and asked for a fleet measurement, which is ticket DL-191.

9 October — Diagnostic Ethernet stays with the gateway team

Decided by Lucie Marešová with the gateway lead. The logger does not grow a second diagnostic stack. Reason: two stacks mean two service tables to keep in step with one customer requirement.

COMMENTS

Add a comment...

Comment

No comments yet.

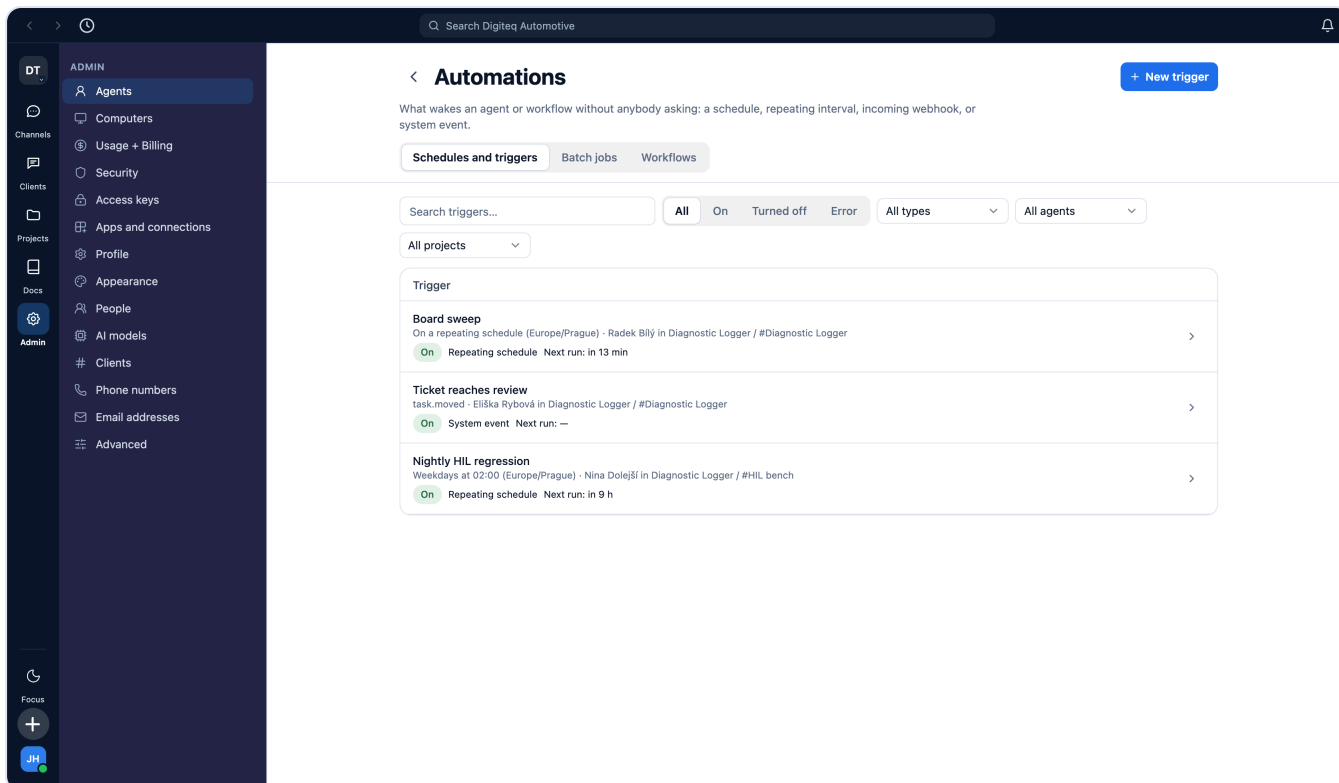
Edit

Add attachment



The decision log for the same project. What was decided, by whom, why — and the dissent, recorded with the ticket it turned into. Nobody typed this up after the design review; the agent that was in the room wrote it the same day and tied it back to the requirement it changed.

Meanwhile, the work keeps moving without anyone on call:



A nightly HIL regression at 02:00, a sweep of the board every half hour during working hours, and a re-read of the specification whenever a ticket reaches review. Each one belongs to a named agent, in a named room, in a named project.

5. Research is part of the job

Every agent can reach an internal deep-research capability: a bounded, cited investigation of a technical question — which library, which approach, what the standard actually requires, what the licence allows, whether the part has published errata. This is the legwork a developer would otherwise do by hand, and in the thread above it is what turned "the deserializer sometimes does not lock" into "errata ER-2214, B2 silicon, and here is what it prescribes."

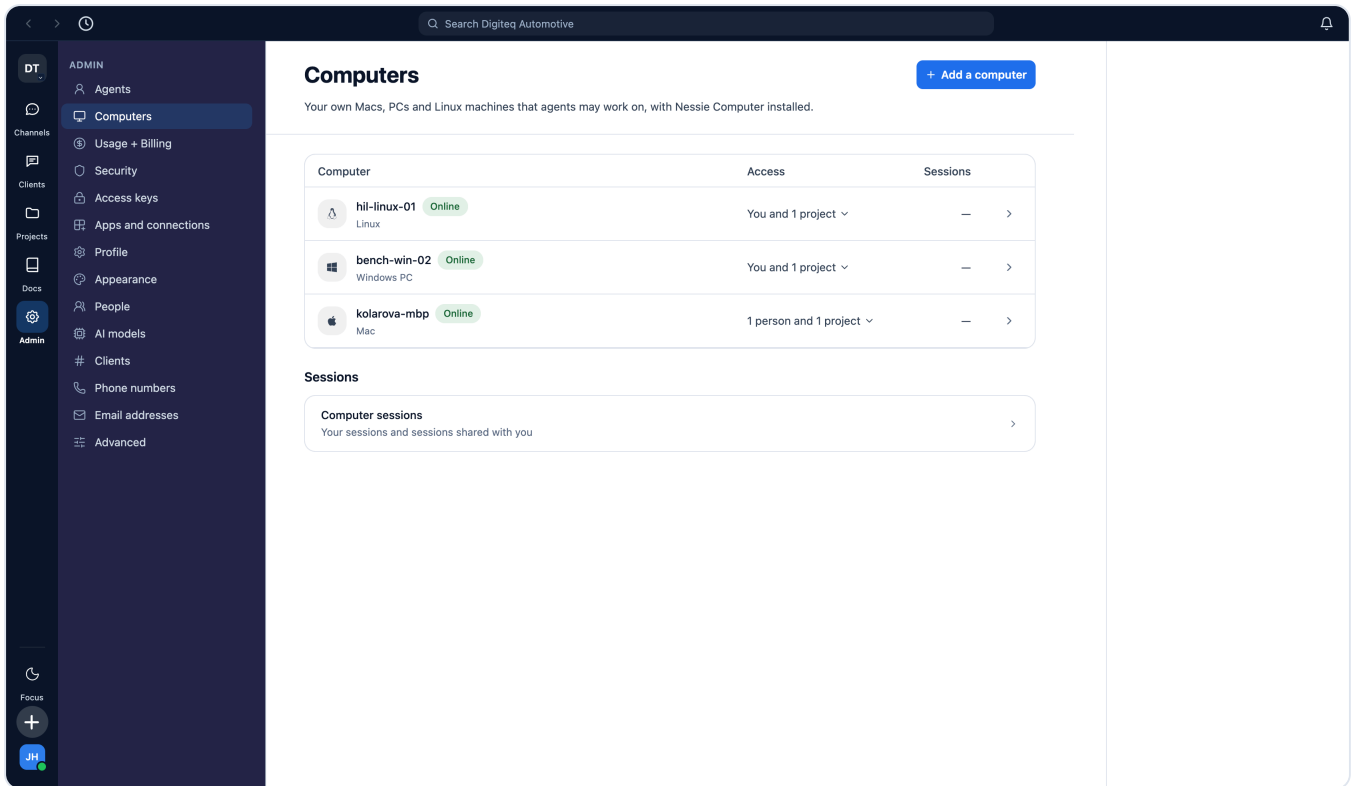
It is off by default and granted per agent, because research calls an external service and costs money.

Not captured. The research surface needs a live connection to that external service, which a local capture run has no business holding. There is therefore no screenshot of it in this document, and none has been drawn. It is the one capability here shown only in the thread that used it.

6. The code never leaves the building

This is the section that decides whether Nessie is usable inside a Volkswagen Group company at all.

Nessie does not upload your repository anywhere. The editing and compiling happen in a coding session on a machine you own, paired to your organisation. Mac, Windows and Linux.



Three paired machines: an engineer's Mac, a Windows bench box and the Linux HIL host. Each one says who may use it and which projects it is available to. A ticket dispatched from the board becomes a session on one of these.

What this gives you, concretely:

- **Source stays on your network.** The agent drives a session on your hardware; the model sees what the session sends it, under your own policy, and the working tree never moves.
- **The toolchain is the one you already have.** Your compiler, your flashing tools, your HIL bench, your licences. A bench box that can build the logger today can build it for an agent tomorrow, because it is the same box.
- **It works where your engineers work.** Developer laptops, dedicated build machines, or both. The same project can use all three platforms at once.
- **A machine is granted, not assumed.** Each computer names the people and the projects that may use it. Narrowing it is one change in one place.

7. When an engineer does not like what an agent did

They tell it, and it changes.

Every agent carries a set of written instructions and an access list. Both are editable by hand on the pages in section 2 — but the ordinary path is the Agent Designer panel visible on the right of those screenshots. An engineer who sees an agent widen a ticket's scope, or write a requirement without a source, or touch a file it had no business touching, says so in plain language in that panel. The Designer rewrites the instruction or turns the tool off, and the change is saved for next time. It acts only with the authority of the person asking, so an engineer cannot grant an agent something they do not have themselves.

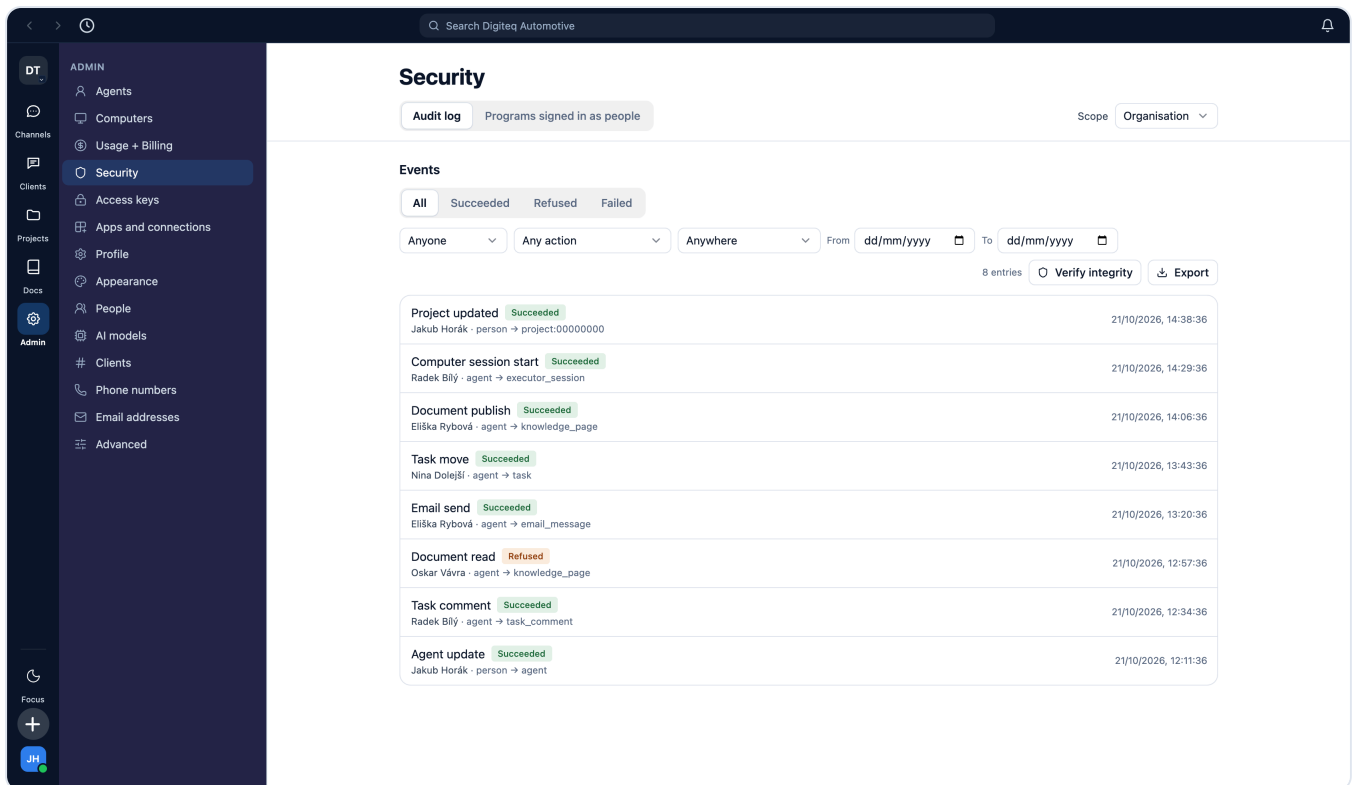
This is the loop that keeps the system improving rather than drifting. In most tooling a bad output produces a complaint. Here it produces a configuration change, made by the person who noticed, in the two minutes they noticed it.

8. Everything is gated, everything is recorded

Two separate properties, and the second does not substitute for the first.

Gated. A project is the access boundary. A team is the data boundary, and teams inside one organisation are wholly independent: no project, channel, conversation, agent, document, ticket or search result crosses between them, not even for an organisation owner. "Public" means public *within the team*. Agents are subject to exactly the same rules as people, and the check runs on the call, not on the screen that drew the button.

Recorded.



Every row names the actor — person or agent — what it touched, and whether it was allowed. The refused row is an agent reaching for a document in a project it is not a member of. The trail is filterable by person, action and project, it exports, and "Verify integrity" checks the chain rather than asking you to trust the list.

For an ASIL-rated work product this is the point. "Who changed this requirement, when, on what evidence, and who released it" has to be answerable months later. It is answerable here because the record is written by the system that did the work, not by whoever remembered to log it.

9. What it costs, honestly

There is a real cost, and it is tokens. These model APIs are billed per call, every call. A busy agent working a hard ticket costs noticeably more than an idle one, and an organisation that uses Nessie to generate documents nobody reads pays for every one of them. Usage is metered and visible per agent and per run, and a budget stops a run rather than surprising you at the end of the month.

The comparison that matters. Set the token spend of the work the agents actually completed against the loaded cost of the engineer-months that work would otherwise have taken. Our position is that for software development done properly, the token cost comes in at a tenth of the people cost or less — and that the leverage is ten times the throughput, not ten percent. We are not asking you to take that on faith. It is measurable, it differs by team, and the honest way to find your number is a bounded pilot on one real project. Measure it, and keep the number.

What "properly" means. The waste case is treating this as a text generator: asking for documents nobody reads, long answers where a short one would do, agents running on schedules that produce nothing. The productive case is the one in this document — tickets with a defined done, research that answers a question somebody actually had, specifications that are updated because a decision changed, builds that run on your own hardware overnight. The cost difference between those two uses of the same product is very large.

This is not a ChatGPT you talk into. A chat window produces text and leaves the organising, the recording, the dispatching and the remembering to you. Nessie holds the board, the documents, the access rules, the audit trail, the machines and the mailboxes, and the model work happens inside that structure. The structure is what turns model output into delivered work — and it is also what stops the spend going into things nobody will read.

10. What a pilot looks like

One real project, one board, three or four agents, six to eight weeks.

1. **Stand it up.** Self-hosted on your infrastructure, or hosted — your call. Sign-in through your existing identity provider. One team, one project.
2. **Pair two machines.** One developer laptop and one build or bench box, so that the "code stays inside" property is tested rather than described.
3. **Put a real backlog on the board** — ideally one with an unglamorous, well-understood piece of work on it, because that is where the throughput claim is easiest to read.
4. **Give the agents jobs, not prompts.** A specification owner, an implementer, a test-and-integration owner. Owners named, access narrow.
5. **Measure three numbers.** Tickets closed per week against your current baseline; token spend for the period; and the time your engineers spent on coordination rather than engineering, before and after.

At the end you will have a number for the second section of this page, taken from your own project rather than from ours.

In short

Project management	Boards, tickets, documents, channels, people — one project, one access boundary.
Project development	Decisions, specifications and test reports written where the work is, by the agents that were there.
Software development	Claude and Codex sessions on your own Macs, PCs and Linux boxes. The source never leaves.
Colleagues, not chatbots	Named agents with owners, email addresses, phone numbers and an access list.
Enterprise from the ground up	Project-gated reads, team isolation, approval before anything leaves, an audit trail that verifies.
Honest cost	Real token spend, metered per agent and per run, against engineer-months. Measure it on a pilot.