From excel gymnastics to lean workflow

Business Process Re-engineering and Architectural Excellence at Retta Property Management

---

Author

Mikko Valtonen

Mikko Valtonen is an Aalto University graduate, a quality black belt in processes, and helps customers at Zeal AI to set bottom line targets, execute change, and measure success using the best AI tech stack on the market. At Zeal AI, we deliver AI Agents, AI Assistant and no-code SAAS. Our solutions drive higher quality work by braking constraints of human resources availability. To make sure the customer gets the best outcome with risky AI investments, Zeal AI delivers all work with a full happiness money-back warranty.

At free time Mikko is father of 8 children and holds both navigation and piloting licences.

Abstract

This technical case study examines the comprehensive transformation of invoice re-billing processes at Retta, a Finnish property management company, through the implementation of agentic artificial intelligence automation. Beyond documenting business outcomes, this analysis reveals the architectural decisions, technical patterns, and implementation strategies that enabled successful production deployment. The research demonstrates how careful technology selection, memory optimization techniques, intelligent error handling, and human-centred design principles combine to create AI systems that augment rather than replace human expertise. The study provides detailed insights into serverless architecture choices, database selection rationale, large language model (LLM) integration patterns, prompt engineering methodologies, and organizational change management approaches. These findings offer practical guidance for organizations seeking to implement similar AI-driven transformations while avoiding common pitfalls and maximizing return on investment.

Chapter 1: The silent knowledge challenge

Retta Real estate management consists of multiple business units. Retta Plus is internal business unit growing size of 4M€ revenue and growing 100 % per year. Retta Plus provides bit better real estate management services, promising time savings, cost savings and for housing companies. [1]

Making everyday life easier with unique benefits

Retta Plus services are services produced for housing companies managed by Retta, which facilitate property management and optimize costs.

Retta Plus services are pre-conceived services and benefits that you can easily implement if you wish. The services make it easier to manage your property, improve security and generate savings for your housing company.
These services are suitable for both small housing companies and large real estate companies.

Interested? Leave a contact request.

I want to hear more about Retta Plus services

Discover Retta Plus services

Fire safety services

Property tax service

Making things easy for housing company means that lot of work in done to serve them at Retta Plus.

Problem

25 % + of real estate property manager time is wasted to non-core work that can be automated with Agentic Process Automation : Procurement decision making studies, Request for Proposal work and classifying incoming invoices into own paid and customer pain items.

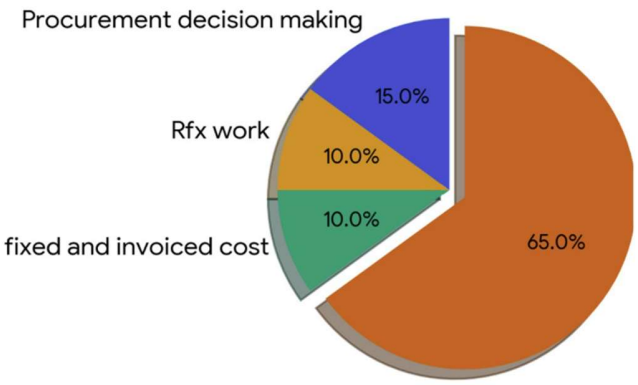

Figure 1: The manual invoice processing workflow consuming forty percent of property managers' productive capacity, representing a systemic inefficiency common across the property management sector.

In order to improve quality and reallocate scares human resources non-routine tasks, Retta identified the need for an **AI assistant** capable of understanding context, handling fuzzy matches, and rapid continuous improvement process.

An AI agent promised the ability to interpret varied product names and pricing rules using natural language understanding – something beyond the reach of template-based automation. This approach aligns with an emerging paradigm in software development: moving beyond explicit code or static rules toward solutions that can *learn and adapt* from data and human feedback. Some have dubbed this paradigm **"Software 3.0,"** where instructions to the computer are given in natural language and large models handle the logic internally[2]. In essence, prompts in English become the new source code, enabling domain experts (not just programmers) to directly encode business logic. This shift dramatically lowers the barrier to building intelligent workflow tools because *"everyone is a programmer"* when they can instruct the system in plain language[3]. For Retta's invoice processing problem – with its nuance and need for on-the-fly judgment – an AI agent built on a large language model offered a path to re-engineer the process without the constraints of legacy automation. The decision to pursue an AI-powered solution was made with clear eyes: the goal was not to replace humans, but to **free them from tedious minutiae** so they could focus on higher-value work, while letting the AI handle the repetitive cognitive heavy lifting in an adaptive way.

Chapter 1: Retta Plus tech stack decision

According to A. Carpathy [4], AI Agents, AI Assistants and no-code business app development apps contain 3 generations of software.

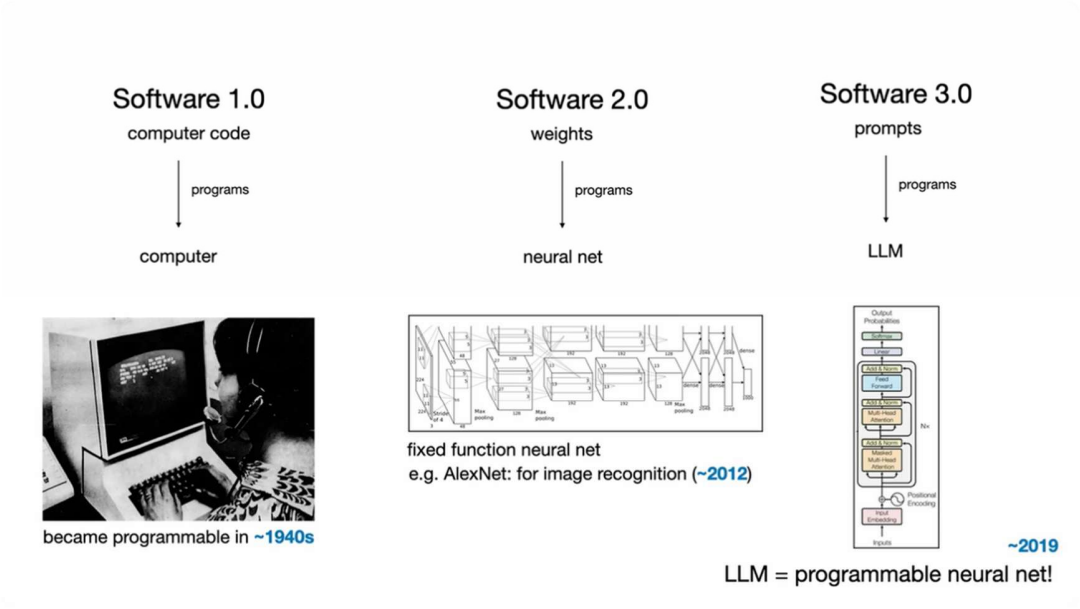

Figure 2: AI Agent software generations

Programming language for software 1.0

Team chose typescript as programming language to save valuable time from dev/ops engineering related to docker containers, code compiling and server restarts.

Hyperscaler selection for software 1.0

From an infrastructure perspective, the team chose a **serverless deployment** (using Vercel) over traditional long-running servers. Invoices come in bursts – morning spikes and month-end surges – making automatic scaling advantageous. The serverless model offers zero infrastructure management and pay-per-use economics. Cold start latency was not a concern since the dominant time cost is calling the AI model, which is orders of magnitude higher than a few hundred milliseconds of initialization. The serverless approach also naturally aligns with stateless LLM invocations, where each function call can be treated as an isolated event.

Database selection for software 3.0

For data storage, a **NoSQL document database (Firebase Firestore)** was selected in lieu of a traditional SQL database. The semi-structured nature of AI outputs (which can evolve

as prompts change) made a JSON document store appealing – it allowed the team to iterate quickly without the overhead of altering rigid schemas. Firestore also offered real-time synchronization, useful for building a collaborative validation UI where changes by one user could instantly reflect for others. The trade-off is that Firestore provides eventual consistency rather than the strong ACID guarantees of SQL, but this was deemed acceptable for the invoice processing workflow, where absolute transactional consistency was not mission-critical. Security and authentication were also simplified by leveraging Firebase's built-in identity management.

LLM Selection (Software 2.0)

Another strategic choice was the cloud and AI model provider (Sofware 2.0). **Google Gemini 2.5 pro model** was chosen over alternatives like Azure/OpenAI's GPT. Gemini 2.5 Pro offered a 2x larger context window than OpenAI flagship model GPT-5.

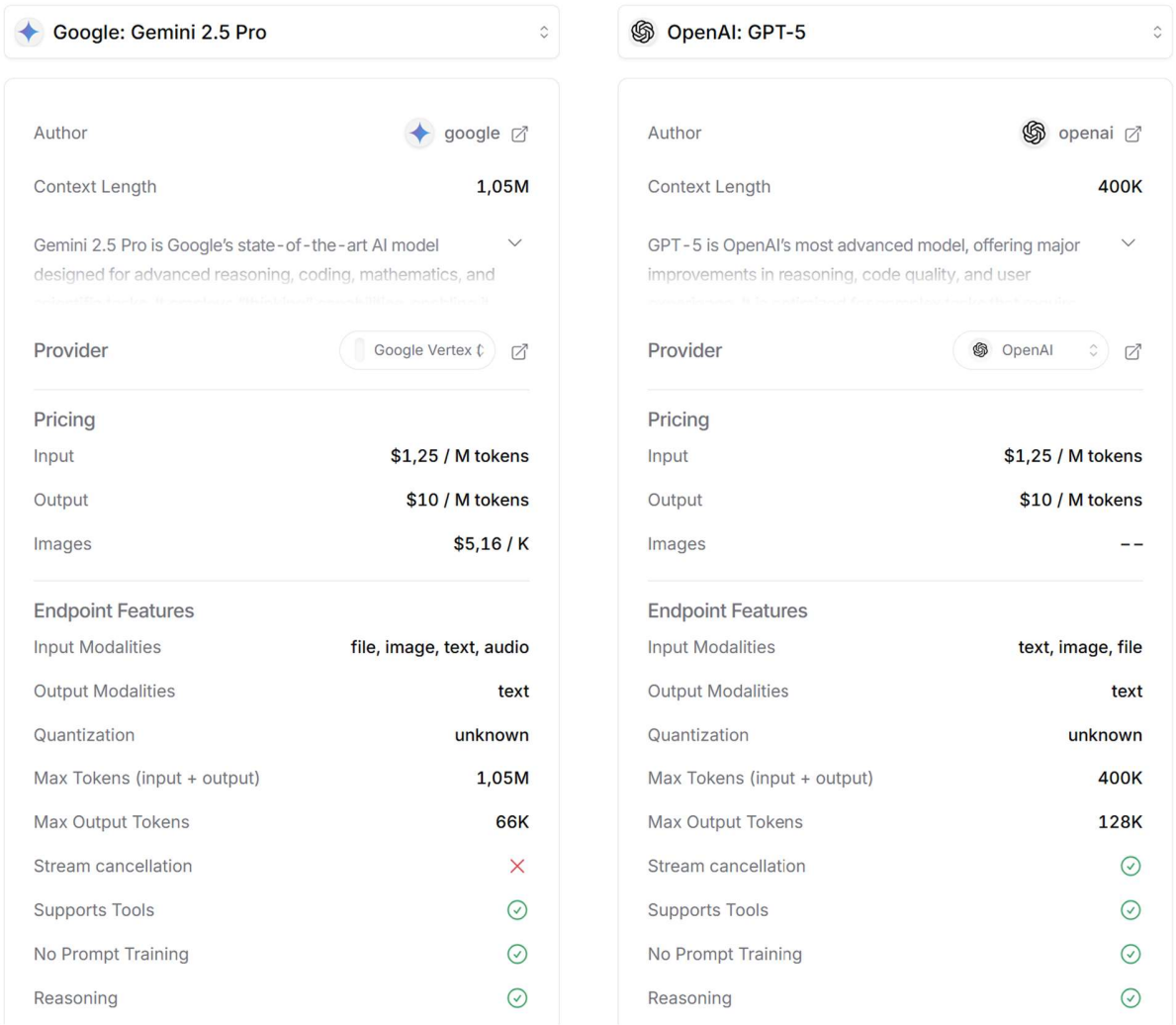

Figure 3: LLM comparison at open router

AI Agent framework (Software 1.0)

Finally, the team confronted the build-vs-buy question for the AI agent orchestration layer (Sofware 1.0). Prebuilt agent frameworks (e.g. Microsoft's Copilot Studio, LangChain, or other orchestration SDKs) could have jump-started development by providing standard templates for conversational flows and tool use. Indeed, Retta evaluated Microsoft's Copilot Studio given the existing Dynamics 365 infrastructure and the promise of a low-code configuration approach. However, these frameworks introduced **constraints that proved incompatible with Retta's requirements**. Template-driven platforms often limit how much one can customize context handling or memory strategy; they tend to assume one-size-fits-all patterns that can lead to suboptimal token usage and higher API costs for a specialized task like invoice processing. Additionally, relying on a proprietary framework can lead to **vendor lock-in** and slower adaptation – Retta wanted the freedom to optimize and tweak the agent behavior at a very granular level, which pointed toward a custom implementation. It was noted in the MIT AI report that generic AI tools often fail to deliver value unless heavily tailored to an organization's workflows[7]. In other words, an out-of-the-box solution would likely misalign with Retta's daily operations, given that *workflow integration and customization* (not just model quality) are what separate successful AI deployments from failures[8].

Retta's engineers built the agent UI and UX scaffolding (Sofware 1.0) **from scratch in TypeScript**, creating a conversation orchestration layer without an overbearing SDK. This custom approach required more initial development effort, but it paid off in ultimate flexibility. The team had complete control over how prompts were constructed, how context was managed between steps, how API call retries and timeouts were handled, and how the user interface interacted with the AI outputs. They implemented conversation memory, function calling logic, and error handling logic directly, optimizing for their specific use case. This is a case where **investing in custom architecture** made sense: the process being automated was core to the business and complex enough to warrant a tailor-made solution. All the tech stack decisions are visualied in picture below.

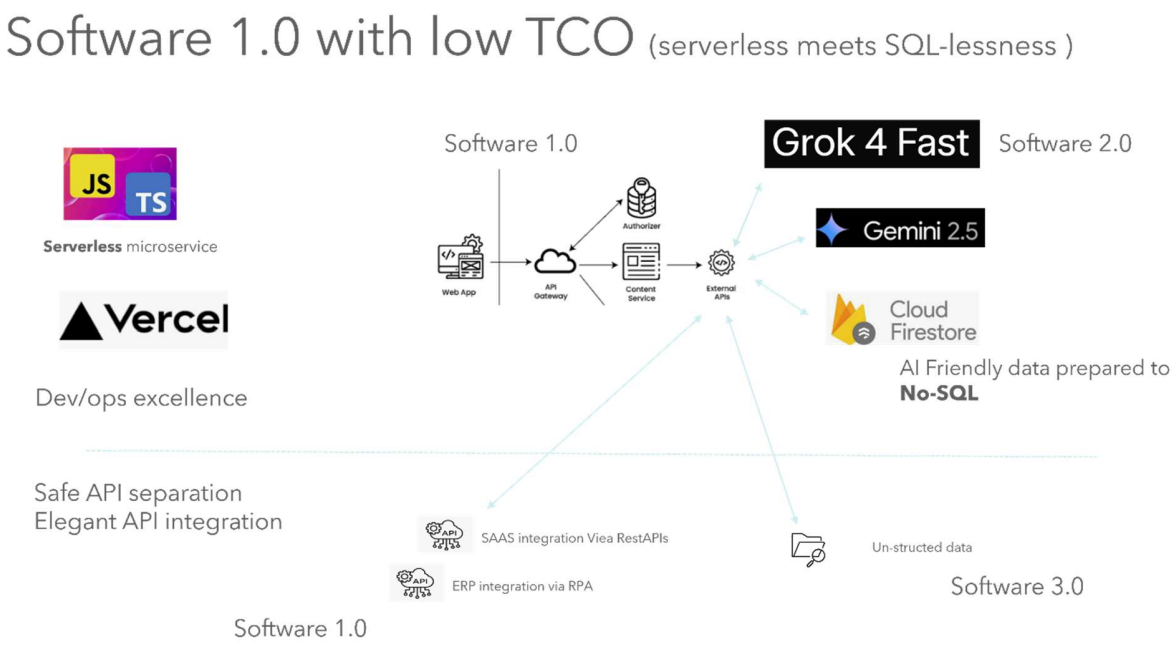

Figure 4: The technology architecture combining a React frontend, Google's Gemini AI, Firebase services, and serverless deployment infrastructure – a stack representing deliberate choices optimizing for development velocity and operational reliability.

Chapter 3: Team roles and project method

Understanding team roles and project methodology starts from understanding project deliverable. Are so called “AI Agents” complex or bulk products? Analog could be searched from other industry. Why nobody purchases good and cheap perfume? This is because quality and scope of perfume is difficult to measure and compare.

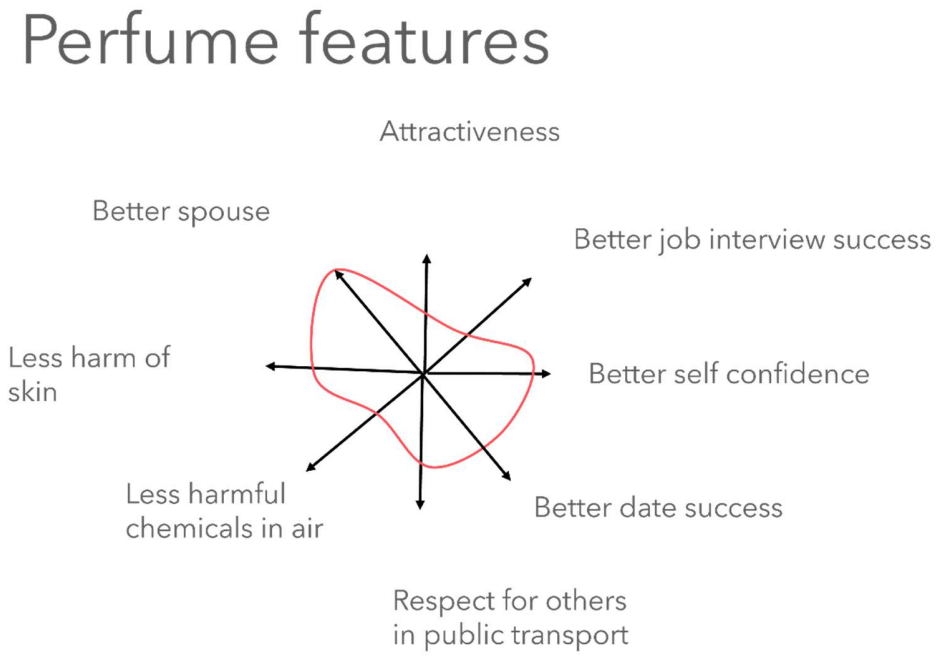

Figure 3: The perfume is iconic example of non-bulk product.

Are so called “AI Agents” specs clear, measurable and comparable ? Team conclusion was no, ai agent features and capabilities are on bar with perfume in terms for complex collaboration and innovation level.

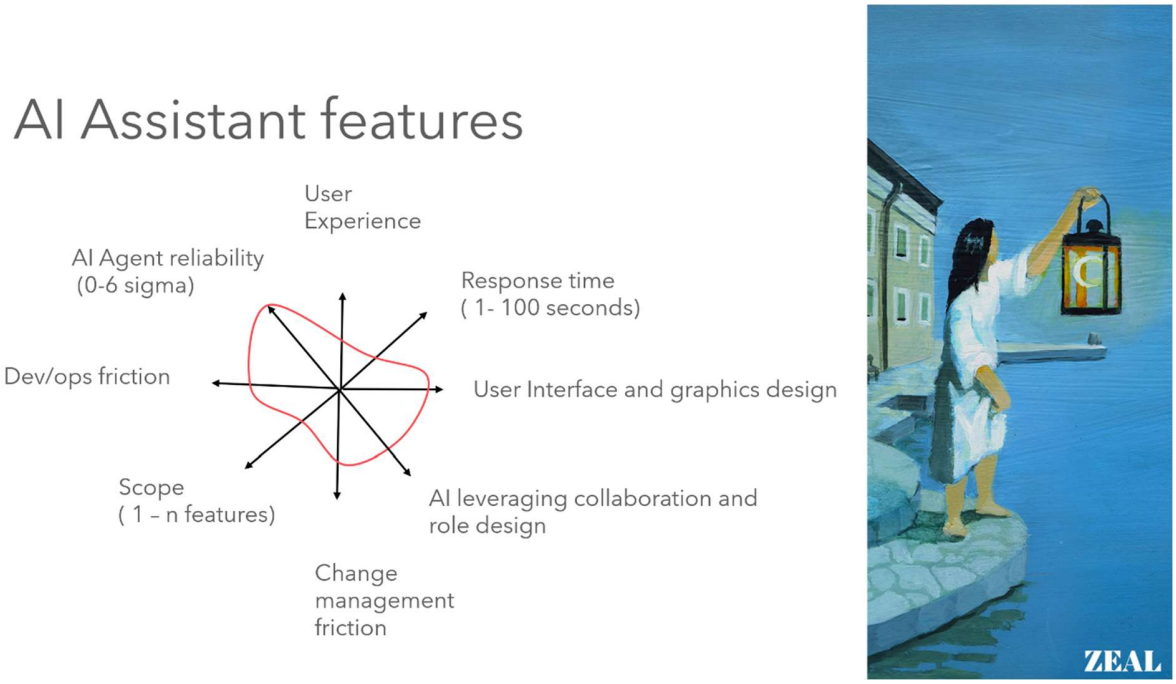

Figure 4: The Retta AI agent's autonomous capabilities included intelligent product matching across naming variations, automated price validation against business rules, and context-aware billing justification generation – demonstrating innovations and complex collaboration needed to achieve bottom line impact.

Answer for complex collaboration and high innovation experimental project have been Agile method. Agile team consists project owner, sprint master and developers. How ever, traditional agile method has been criticized in the context of AI Agents by Palantir, Sierra and many y-combinator authors. Flaws in agile method triggered social media discussion in Q4/2025 when MIT published report stating that 95% of generative AI initiatives fail [9]

This lead to AI-sceptic engineers cheered the "AI hype won't impact the bottom line" conclusion. As counter argument AI acceleration advocates blamed internal IT engineers and external consultants at big agencies for most failures. Accoring to y-conbinator[10] accelerationist counterargument and lens on the MIT study was:

- The problem isn't AI – it's project method and team building leading to poor execution.
- Internally built IT systems often offer poor UX/UI compared to top commercial software.
- AI agents built by internal IT and or IT consultants are likely to deliver the same poor UX as internal IT tool SAAS — and fail.

Assuring customer success and for fixing the flaws in agile method and traditional team building, Sierra and Palantir offer so called Forward Deployment Engineering approach.

AI agent development methodology

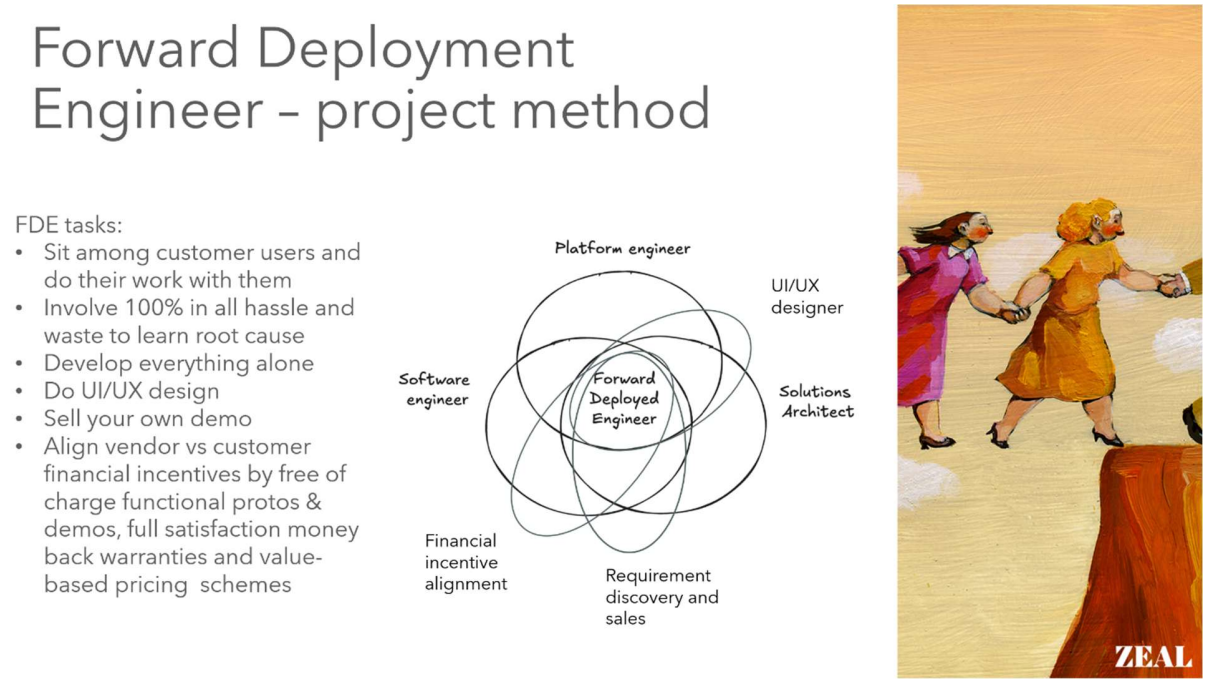

Figure 5: The iterative development methodology combined rapid prototyping, systematic evaluation, and continuous refinement – a cycle distinguishing AI system development from traditional software engineering approaches.

As Building an AI system like this is not a straightforward linear project – it's an empirical, experiment-driven process. Retta had to decide on a development methodology that would accommodate the uncertain, data-dependent nature of AI behavior. The classic **Agile software development** approach was abandoned and the team adopted **Forward-Deployed Engineering (FDE)** model, borrowing a concept pioneered by Palantir and now increasingly used in AI startups[10]. In a forward-deployed model, engineers embed closely with end-users, iterating on solutions in real-time with constant feedback. It's akin to having a developer sitting in the property management office, observing how invoices are processed, tweaking the AI's behavior on the fly, and continuously aligning it with user needs.

This approach differs from a typical Agile cadence where a development team might be somewhat removed, implementing features from a backlog and then presenting them in demo sessions. Instead, Retta's AI team worked *hand-in-hand with the business users* daily – a true DevOps-style collaboration where development and operation blended. Such an FDE-style approach has been highlighted in industry as a key to success for AI deployments: by **co-designing with users** and rapidly iterating, the solution stays tightly aligned to real requirements. As one expert put it, a forward-deployed engineer *"sits at the customer site and fills the gap between what the product does and what the customer needs"*[11]. That is exactly what was done here – the AI solution wasn't built in a vacuum,

but through constant interaction with the actual invoice data and the managers who understood the process nuances. AI Engineer was forced eat it own dog food. ‘

Biggest induvial benefit of FGE method is incentive alignment, if customer don’t like project deliverable, they don’t have to pay. Full money back warranty made motivated vender to build AI Agent that is usable and reliable in messy real life, not only in clean laboratory,

FDE method high level project learning levels are illustrated in picture below.

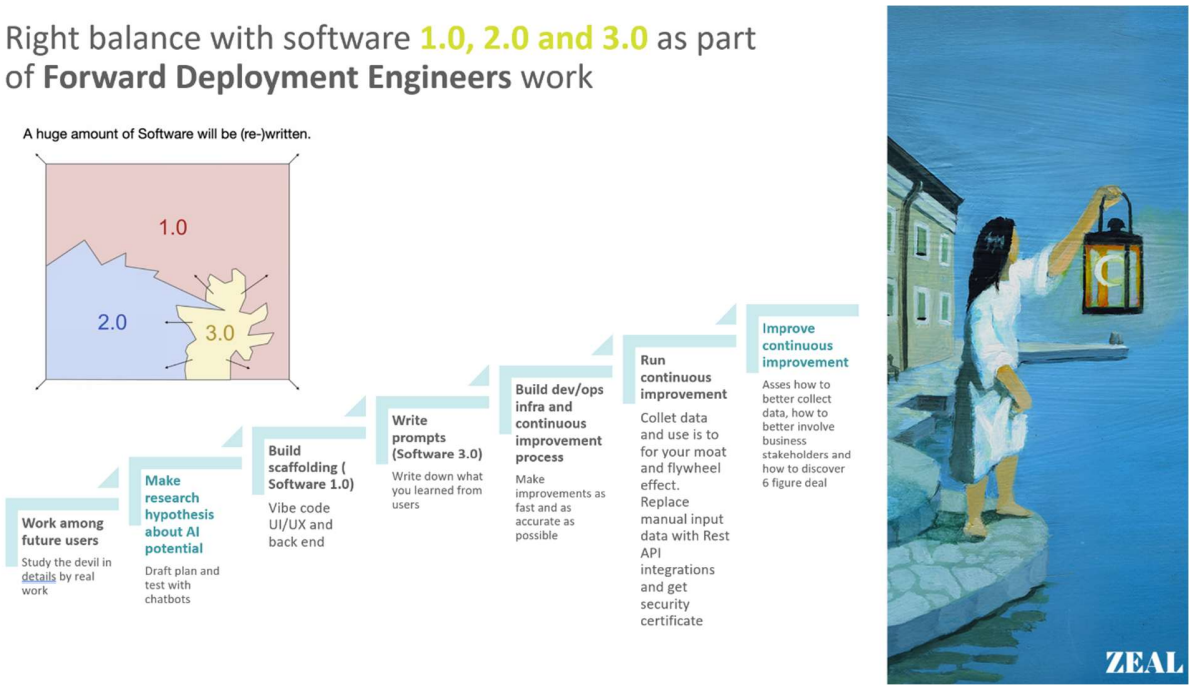

Figure 6: Learning levels. In the beginning FDE engineer learns what it proble and in the end learns how to improve continuous improvement cycle

Chapter 5: Harsh reality of bottleneck

Pretty soon in first demo we found and that invoicing decision require wide content and best in the market 1Million token context window is dominant bottleneck for success.

Retta's team discovered that core interdependent decision tree with detail specification of all subtasks could be stored in one prompt if all pre-processing and post processing was separated and facilities away from main prompts. The process looked roughly like this:

- **Data preparation:** When a new invoice needed processing, the raw Excel or PDF data was first pre-processed. A small script converted the purchase invoice Excel into a clean JSON structure (eliminating extraneous formatting, normalizing numeric formats, etc.). This made sure that the prompt wouldn't be bloated with irrelevant tokens like Excel styles or empty cells. This step alone cut down context size by 30–40% as compared to naively copying the raw file content.

- **Context assembly:** At the start of an AI session (when the conversation with the LLM is initiated), the system would send a structured **system prompt** containing the overall instructions and rules (e.g., "You are an assistant that helps generate billing invoices. You have the following functions available… Here are the business rules for pricing…" etc.). It would not, however, include the actual invoice data yet. Then, the agent would make targeted function calls to gather just the necessary pieces: for instance, call `searchOstolaskuExcel` to retrieve the purchase invoice lines (as JSON), call `searchHinnasto` for each product or in batch, etc. Each function call result was added to the conversation as it came in. This incremental assembly meant the AI always dealt with **focused context chunks**: at one moment it's looking at product search results, at another at customer order history, rather than one giant blob of everything at once.

- **20 Parallel LLM calls:** Incoming well prepared LLM friendly json was divided in 20 rows and each 20 row set was process parallel. This lead to decent 2 min response time and optimal usage or 1M token context

- **Output post-processing:** After the LLM generated a draft output (for example, a markdown table representing the sales invoice to be produced), additional steps occurred outside the model. The client-side code would validate the format (was the output a proper table? did it adhere to the expected schema?), and then convert that markdown or JSON output into a final Excel file using a library (e.g., an XLSX generation library in JavaScript). This meant the heavy lifting of formatting the final deliverable was done with conventional code, not by the AI. It's an example of smart workload division: use the AI for what it's good at (understanding context, applying rules flexibly, generating explanations), and use deterministic code for things like file generation, arithmetic verification, etc.

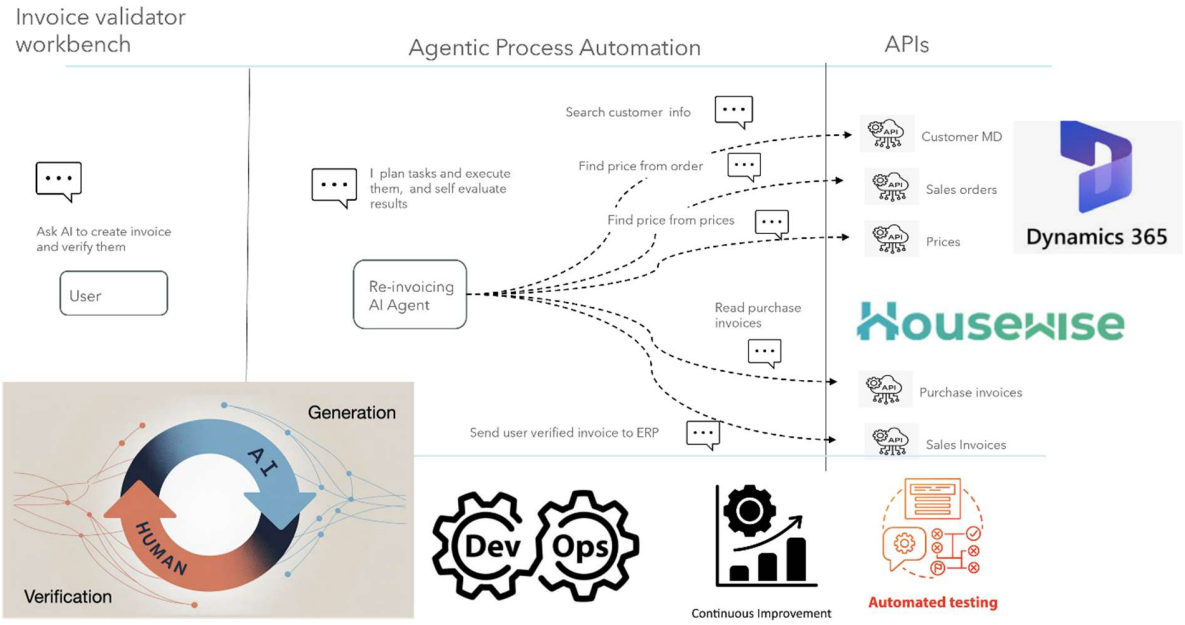

Figure 7: Visualisation of UI/UX alias scaffolding alias software 1.0

By using **maximal prepropessing and postprocessing** instead of one gigantic prompt, the solution maintained a high signal-to-noise ratio in each AI interaction. This modular approach also made the system more **maintainable and interpretable**. All business logis was in one prompt which was easy to understand and edit by humans. If the output was wrong, it was easier to trace whether the error came from, say, the product search step or the pricing application step or the final composition step. Each of those corresponded to a part of the conversation one could analyze.

In conclusion, **Retta's implementation context optimisation method**, and this proved both efficient and effective. The team was able to make system robust, easy to develop in no-code, transparent of mistake root causes, fast and highly reliable.

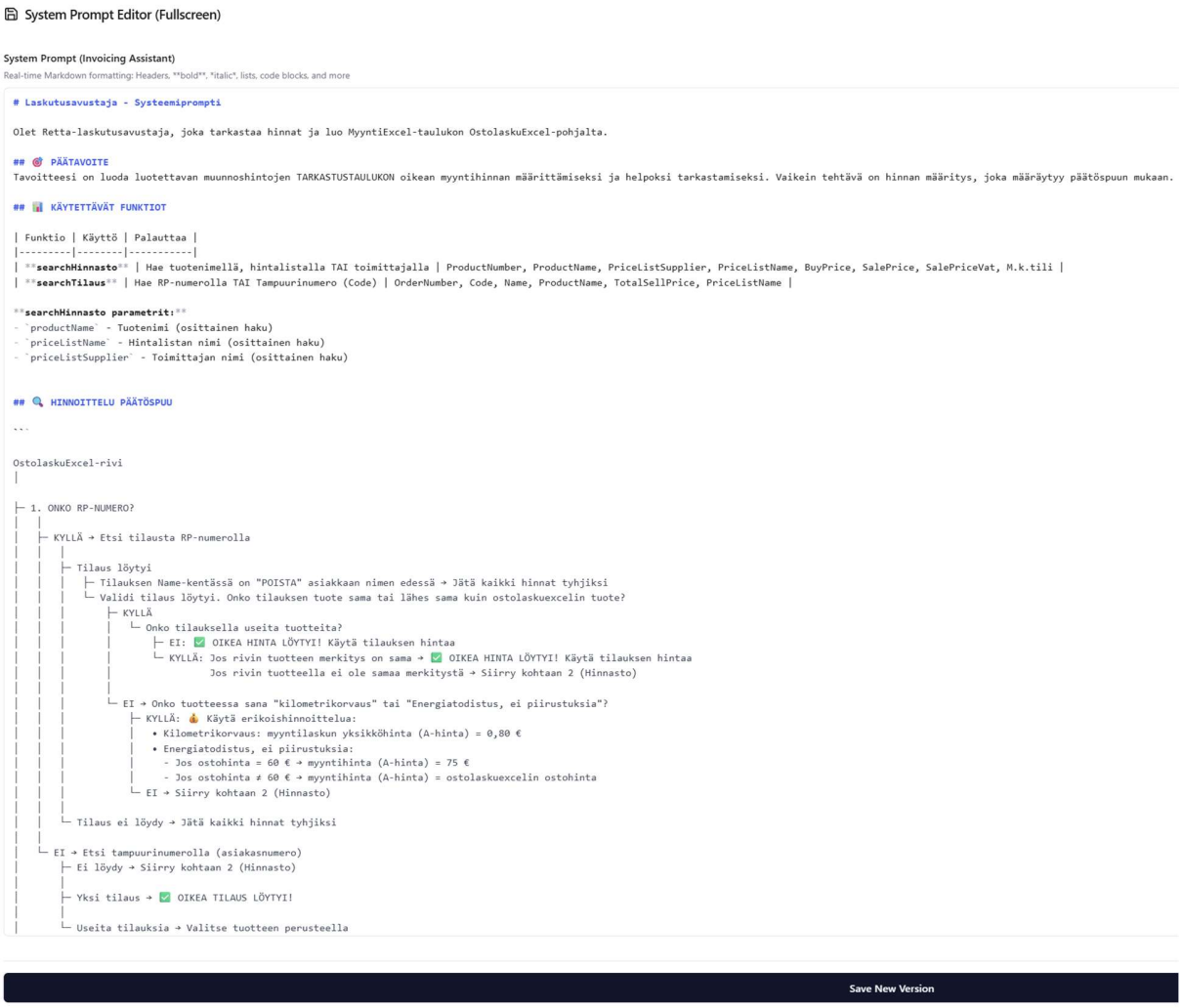

Figure 8: The UI for business writing business logic, stored in google firestore no-sql

Chapter 7: Human-in-the-Loop Verification

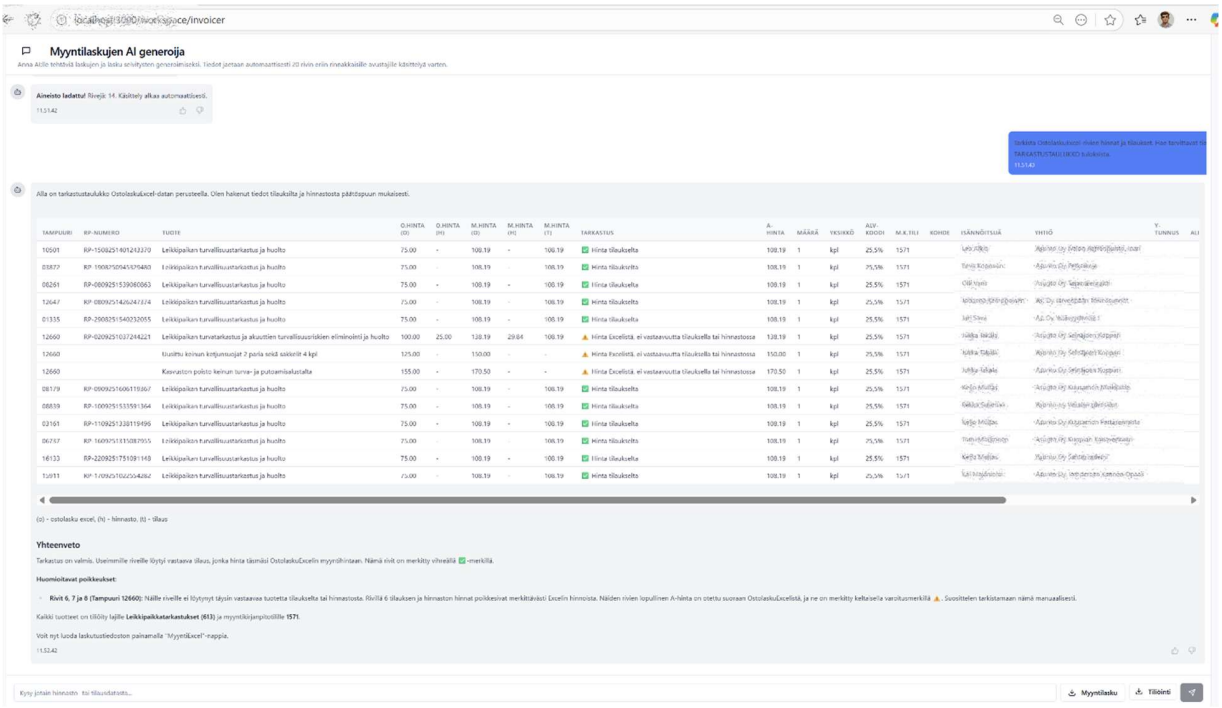

Figure 9: The Invoice Validator Workbench interface demonstrating conversational interaction and transparent AI reasoning. The design maximizes automation benefits while enabling efficient human oversight and intervention when needed.

A natural question in deploying an AI agent for critical business processes is: **how autonomous should it be?** Should the AI run independently, end-to-end, with no human involvement (an "fully independent agent run"), or should it operate under continuous human supervision, with people verifying every result (which can become "tedious result verification" if overdone)? Retta's approach was to find a **sweet spot in between** – leveraging as much autonomy as possible, but with carefully designed feedback loops and oversight to ensure quality and build trust. This falls under what Andrej Karpathy and others call *partial autonomy*, often conceptualized as an **"autonomy slider"** that can be tuned based on confidence and context[15].

In practice, Retta's AI agent handled the majority of invoices without intervention, but the system was built to make verification *efficient rather than burdensome*. The user interface (Invoice Validator Workbench) played a huge role here. It presented the AI's output (suggested sales invoice) side-by-side with the original purchase invoice data, with clear highlights on anything the AI was unsure about. For example, if the AI had only a low-confidence match for a product, that line might be highlighted in yellow. The property manager could quickly scan those highlights, rather than recheck every single line from

scratch. If everything looked good, they would approve the invoice for billing in seconds – meaning **zero-touch processing** for that invoice apart from a quick glance. If there was an error or a question, the interface allowed inline editing or comments. Crucially, any corrections made by the human were fed back into the system's learning process (the AI would get that feedback at least asynchronously, if not in real-time, to adjust its future outputs).

Retta embraced what we can call a **generation–verification loop**. The AI generates a result; a human verifies and corrects as needed; the AI then learns from that correction for next time. The goal is to make each loop as fast and informative as possible. Research and industry best practices suggest that the *speed of this loop* is critical: the faster the AI can be corrected and the more transparently it can show its reasoning, the quicker it improves and the higher the overall productivity[16]. In the Workbench UI, therefore, verification was streamlined by features like one-click approvals, side-by-side diff views (to see what changed from the original), and even a chat-like interaction window where the AI would explain its reasoning in plain language upon request (e.g., "Why did you choose 15% markup for this item?" and it would answer citing the rule from the prompt). This built user confidence and made the verification process feel like a collaborative review rather than grunt work.

From the AI's perspective, being **kept on a tight leash** in generation means it was not allowed to, for example, send out an invoice email to a customer on its own or make irreversible changes in the ERP without a human sign-off. It would always produce a *proposed* action, and the actual finalization (posting the invoice to Dynamics 365, etc.) was triggered by a human click. This design reflects a recognition of the current limits of AI reliability – as Karpathy quipped, *"Demo is works.any(), product is works.all()"*[17]. An AI might succeed on a demo or a random invoice (works in *some* cases), but a production system needs to work in *all* the necessary cases. The only way to bridge that gap initially is to include a human check for the cases where the AI might fail. Over time, as the AI gets better (and "all cases" becomes closer to reality), the level of autonomy can be dialed up.

Retta explicitly **did not pursue a fully autonomous solution from day one**. They treated the AI agent as a junior colleague to the human managers – one who could do a lot of the heavy lifting but still required oversight on tricky bits. This concept maps to the "Iron Man suit" analogy sometimes used in AI circles[18]: the AI is like a suit that gives the human superpowers (augmentation) and sometimes can act on its own (autonomy), but the human is still in control overall. By giving property managers a conversational interface, Retta made interacting with the AI feel natural. A manager could type a command like "Process invoice #12345 for customer 11011" and the AI would go fetch the data and do it, explaining each step as needed. If the AI got stuck ("I couldn't find a price for item XYZ, what should I do?"), it would ask in the chat. This **interactive mode** ensured that when autonomy broke down, it didn't result in silent failure – it resulted in a question to the human, which is exactly how a human assistant would behave.

Over the first few months of deployment, the amount of verification required steadily dropped. Initially, managers carefully reviewed every line (as one should with a new system). They frequently found small errors or format issues, which were corrected. Those corrections went into improving the prompt or adding new examples for the AI, and soon those particular errors disappeared. By month three, perhaps only 1 in 5 invoices had any highlights or issues to double-check. The others sailed through, and managers began to trust the AI to the point that they would often batch-approve dozens of invoices with minimal spot-checking. At the same time, because the system was *transparent*, they never felt out of the loop – they could see the reasoning and be confident nothing was being hidden.

In summary, Retta's approach to autonomy was pragmatic and **user-centric**. They avoided the trap of all-or-nothing automation. Instead, they implemented **"autonomy sliders"**: if confidence is high and the case is routine, the AI operates almost fully autonomously; if an anomaly is detected or confidence is low, the system requires more human involvement. This flexibility is what made the solution viable and acceptable to the end users. It also aligns with broader findings that the most successful AI deployments are those that *augment human workers rather than replace them*. By designing the workflow as a generation-verification loop, Retta ensured that the AI continuously improved (with human teaching) and that humans remained **accountable supervisors** of the work, which is crucial for compliance and trust. The result was a synergistic human-AI partnership where the AI did in seconds what would take a person minutes, and the person provided judgment that the AI still lacked – together achieving results neither could alone.

Chapter 8: Continuous improvement

Continuous improvement and feedback loops

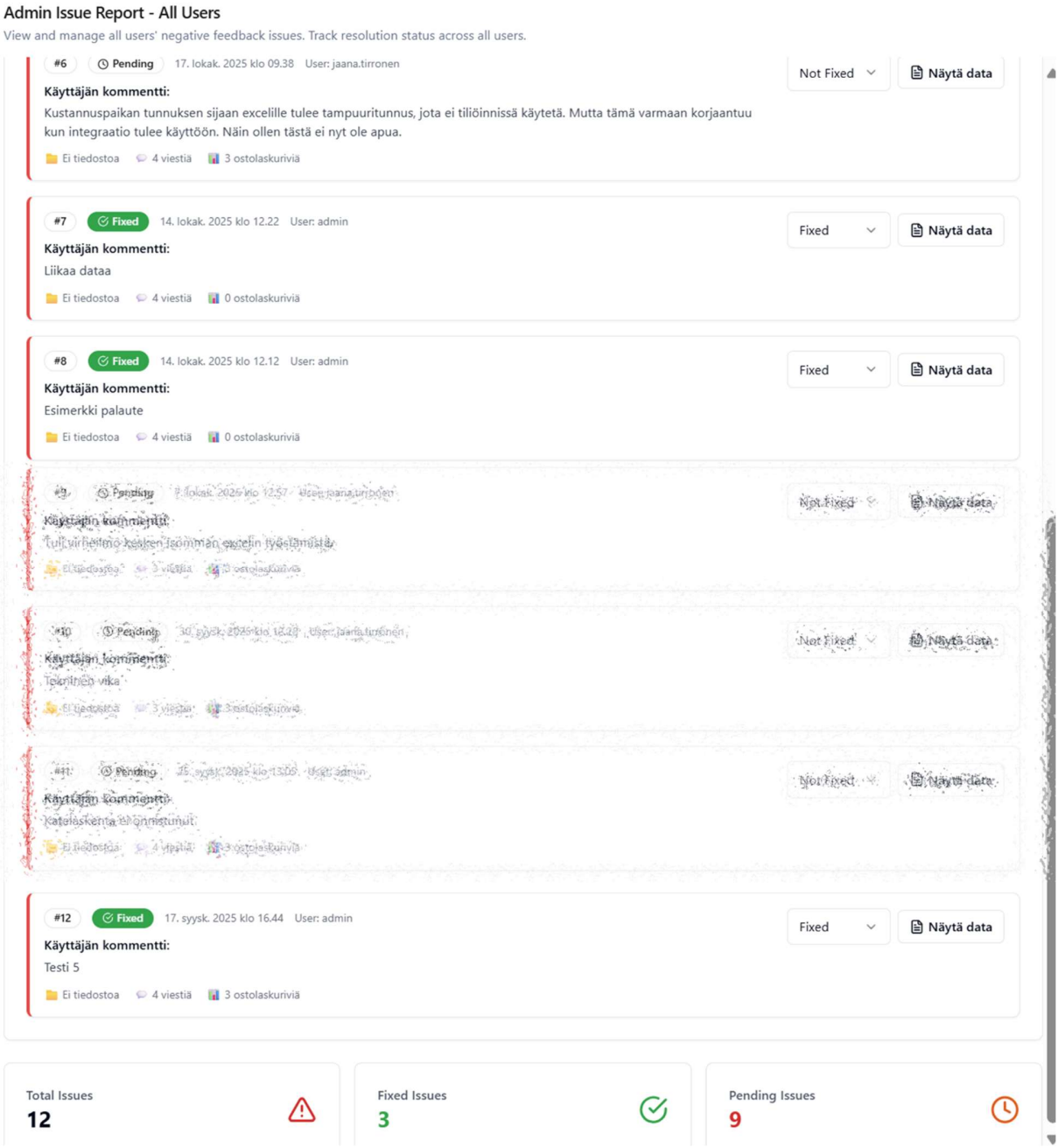

Figure 10: The comprehensive feedback collection and continuous improvement infrastructure enabled systematic performance optimization through structured evaluation, automated logging, and data-driven refinement cycles.

Deploying an AI system into production entails an ongoing balancing act between **improvement speed** (rapidly updating the system as you learn and as conditions change)

and **operational quality** (maintaining stability, reliability, and meeting business SLAs). In traditional software, one might freeze changes for a while to ensure stability (especially in core processes), but with AI – which learns from data and can improve via prompt tweaks – there's a temptation to continuously push updates. Retta's strategy was to embrace a **DevOps mentality, in the AI world**, which allowed for fast improvements without sacrificing quality.

From day one of deployment, the team set up **comprehensive logging and monitoring**. Every interaction the AI had – every function call, every generated output, every user correction – was logged to Firebase. This created a rich dataset for analysis. They categorized logs by outcome (e.g., "successful auto-processed invoice", "flagged for review", "error encountered") and by error type when applicable (mismatch error, calculation error, etc.). This instrumentation meant that at any given moment, the team knew exactly how well the AI was performing (e.g., 87% of invoices this week were processed with no edits, 10% had minor edits, 3% had major issues that required multiple corrections). Having these metrics was crucial for deciding when to push improvements and when to hold back.

Retta opted to prioritize **continuous improvement speed** in the early phases. They operated in what might be called a *90-day sprint*, echoing MIT's finding that top performers often execute AI implementations in roughly 90-day cycles for major gains[19]. In practice, this meant they were updating the prompt or the code on an almost daily basis in the first few weeks, then weekly as things stabilized. Because changes were small (e.g., "adjust wording in prompt on rule X" or "tweak how function Y's output is parsed"), the risk of any given change causing a serious issue was low – and even if it did, they could quickly revert, given the robust version control on prompts and functions.

To ensure **operational quality** wasn't compromised, they had a few safeguards. Any time a new version of the prompt or code was deployed, the system ran a **regression test suite**: essentially a set of sample invoices (including edge cases) that had known correct outputs. The AI would process those in a sandbox and the results would be compared to the expected outputs. If there was a significant deviation, that deployment was halted for review. This is analogous to unit tests in traditional software, but adapted to AI where exact outputs can sometimes vary in wording. They mainly tested for numerical accuracy and the presence of required justifications, rather than exact wording match. This automated check, running in seconds, gave confidence that improvements weren't breaking core functionality.

Importantly, Retta recognized that **continuous learning and adaptation** is a hallmark of AI systems. Unlike traditional software that remains static until a developer changes it, an AI system *can improve on its own* if set up correctly. The logging and feedback loop served as a de-facto training signal. Although the team did not retrain the underlying LLM (Gemini) – that was out of scope – they did effectively *train* the system via prompt engineering adjustments informed by feedback. This is akin to the ModelOps concept Gartner highlights: moving from brittle rules to continuously learning models in production[20].

Conclusions and Implications

This comprehensive technical and organizational analysis of Retta's invoice processing transformation demonstrates that achieving success with AI is a multidisciplinary endeavor. It required **excellence across architecture, engineering, user experience, and change management**. By carefully addressing each of these dimensions, Retta realized a rare outcome: a production AI solution delivering substantial, measurable business value – something only an estimated 5% of enterprise AI pilots manage to do[26][27].

From a technology standpoint, the case illustrates the importance of aligning architecture with the problem's nature. Retta did not uncritically follow hype or defaults; every major decision was context-driven. The choice of serverless deployment matched the usage pattern and need for scalability without ops overhead. The decision for a NoSQL database aligned with the flexibility needed for AI-generated data structures. Selecting a particular LLM (Gemini) over another (GPT) was based on memory. Perhaps most notably, the decision to build a custom agent in TypeScript – rather than using a generic AI orchestration platform – gave Retta fine control to optimize memory usage and collect custom logs to understand error root causes. This runs somewhat counter to the current trend of "graphical workflow" AI solutions like 8n8. Retta's experience suggests that for complex, mission-critical workflows, **investing in a custom solution can yield superior outcomes**, as it avoids the compromises and "brittle workflows" that generic tools often impose[28].

The implementation details in Chapters 5 and 6 about prompt engineering and memory optimization provide a blueprint for **practical LLM application development**. We saw that efficient data preprocessing, parallel LLM processing, and coding all business logic with natural language as possible allowed the system to perform well without exorbitant costs.

A striking takeaway from Retta's project is the **empowerment of domain experts** via prompt-based logic. This case validated what many in the AI community have speculated: that we are entering an era where subject matter experts can directly "program" AI systems using natural language. The speed and fidelity with which Retta could adapt the AI to new pricing rules or invoice types, using prompt edits by non-developers, was a revelation. It compresses the cycle from business idea to operational implementation dramatically. However, it also introduces a new kind of risk – the prompt is powerful, and an erroneous instruction could wreak havoc if not checked (imagine a typo that says "markup 150%" instead of "15%"). Thus, governance of these human-language "code" changes is important. Retta managed this by treating prompts like code: reviewing significant edits, maintaining a history, and testing thoroughly. Other organizations following this path should instill similar discipline. But the upside of doing so is immense: **business agility** at a pace that classical IT change management could never allow. In sectors where rules or products change frequently (finance, healthcare, retail), this could be a game-changer.

Contact

Ready to Transform Your Operations?

We work with property management companies across Finland to implement AI-driven process automation. Schedule a consultation to explore how these approaches might apply to your operations.

Contact: mikko@zealsourcing.fi

Technical Excellence in AI-Driven Business Process Re-engineering

---

[1] [2] [3] [4] [20] [29] Why Legacy RPA and OCR Automation Falls Short in an AI-Powered Future with IDP

https://www.lightico.com/blog/why-legacy-rpa-and-ocr-automation-falls-short-in-an-ai-powered-future-with-idp/

[5] [6] [14] Software 3.0: the LLM Revolution According to Andrej Karpathy | by Antonio Troise | Medium

https://levysoft.medium.com/software-3-0-the-llm-revolution-according-to-andrej-karpathy-5778d0295c8d

[7] [21] [22] [24] [25] [27] Why 95% of Enterprise AI Pilots Fail: Lessons from MIT's 2025 Report

https://kendallai.org/blog/why-95-of-enterprise-ai-pilots-fail-lessons-from-mits-2025-report/

[8] [9] [19] [23] [26] [28] [30] MIT: Why 95% of Enterprise AI Investments Fail to Deliver | AI Magazine

https://aimagazine.com/news/mit-why-95-of-enterprise-ai-investments-fail-to-deliver

[10] Agent Deployment Engineers: The Evolution of Deployment Roles in Enterprise Software

https://beam.ai/agentic-insights/agent-deployment-engineers-the-evolution-of-deployment-roles-in-enterprise-software

[11] [12] [13] Lessons from Bob McGrew

https://www.antoinebuteau.com/lessons-from-bob-mcgrew/

[15] [16] [17] [18] Andrej Karpathy on Software 3.0: Software in the Age of AI

https://www.latent.space/p/s3