Improving Software Quality Through AI-Assisted Discovery in a Large Enterprise System

William Niemiec
Institute of Informatics – UFRGS
Porto Alegre, Brazil
william.niemiec@inf.ufrgs.br

Matheus Pereira
Technology – Paytrack
Blumenau, Brazil
matheus.pereira@paytrack.com.br

Douglas Silva
Technology – Paytrack
Blumenau, Brazil
douglas.silva@paytrack.com.br

Daniela Caroline Kock Klitzke
Technology – Paytrack
Blumenau, Brazil
daniela.koch@paytrack.com.br

Ian Tomelin Serpa
Technology – Paytrack
Blumenau, Brazil
ian.serpa@paytrack.com.br

Gilbran Cristovão
Technology – Paytrack
Blumenau, Brazil
gilbran.cristovao@paytrack.com.br

Cristian Zimmermann de Araujo
Technology – Paytrack
Blumenau, Brazil
cristian.araujo@paytrack.com.br

Abstract

Large enterprise systems frequently evolve through continuous maintenance activities, changing business requirements, and contributions from multiple teams without corresponding updates to their technical documentation. Over time, this process leads to fragmented knowledge, reduced system understanding, and increasingly reactive maintenance practices. This paper reports the experience of restructuring the mobility module of Paytrack, a large-scale travel and expense management platform. To address recurring operational issues, a dedicated team conducted a systematic discovery and reverse engineering process combining collaborative code analysis, technical story decomposition, abstraction-oriented documentation, artifact generation through artificial intelligence, and continuous knowledge-sharing sessions. One of the main contributions of this initiative was the *Abstract Sketch Strategy*, an approach focused on representing complex system flows using higher-level abstractions to improve readability and reduce cognitive overload. After four months, the initiative resulted in a 75% reduction in bugs escalated to the development team and a 90.13% reduction in contact rate related to the module. Additionally, the initiative improved collective knowledge sharing and increased the maintainability of the system, making the team shift from a reactive development to a preventive one.

Keywords

Software Engineering, Artificial Intelligence For Software Engineering, Reverse Engineering, Industrial Experience Report

1 Introduction

Large enterprise systems commonly evolve through continuous maintenance activities, changing business requirements, and contributions from multiple teams over long periods of time. When these changes are not accompanied by dedicated ownership or systematic documentation updates, technical knowledge gradually becomes fragmented and difficult to recover [5, 8]. As a consequence, organizations frequently experience progressive knowledge loss, reduced system understanding, and increasingly reactive maintenance practices. This issue is particularly severe when modules relate to business-critical processes that directly affect customer experience and operational continuity [6].

Paytrack[1] is a Brazilian enterprise platform focused on corporate travel and expense management. It provides several business products, including a mobility module, which is responsible for processing and reconciling corporate transportation expenses generated during employee business trips, including integrations with external transportation providers and Enterprise Resource Planning (ERP) systems.

The mobility module did not have a dedicated team for it. Instead, it had been maintained inside the expenses team, which is responsible for the management of corporate expenses, reimbursement processes, and related financial workflows. This was a business decision because the module directly affects the expenses module. Over time, the mobility module evolved without dedicated ownership or sufficiently structured documentation practices, and the knowledge became fragmented, business rules became difficult to understand, and maintenance activities gradually became reactive instead of preventive.

As a consequence, the company has seen a rise in the number of support tickets over time. In addition, as the documentation did not evolve asynchronously with the complexity of the system, it has been difficult for the expenses team to safely evolve and develop this module due to a lack of comprehension about what has previously occurred with it, along with the constant demand for expense-related issues.

To address these issues, the company created a dedicated team in January 2026. The goal was to improve the maintainability, operational stability, and technical understanding of the module. Instead

[1] https://paytrack.com.br

of focusing exclusively on reactive maintenance, the team prioritized a preventive approach through a structured discovery process centered on reverse engineering [1, 7], collaborative analysis, abstraction-oriented documentation, and knowledge dissemination.

The relevance of this experience lies in demonstrating how relatively lightweight practices, supported by Artificial Intelligence (AI)-assisted tooling and human validation, can substantially improve operational indicators and technical understanding in large enterprise systems. In addition, the reported approach can be reproduced by other industrial teams facing similar scenarios involving undocumented or poorly understood software modules.

The main contributions of this work are:

- an industrial experience report describing a structured discovery initiative in a large enterprise system;
- practical insights regarding the use of AI-assisted tooling for reverse engineering and documentation activities;
- evidence that collaborative discovery and abstraction-oriented documentation can improve operational and maintainability indicators.

All activities described in this work followed the company's internal security, privacy, and governance policies, as well as the requirements established by the Brazilian General Data Protection Law (LGPD). No sensitive customer information, personal data, or production-critical operational details were exposed during the discovery, reverse engineering, documentation, or AI-assisted analysis activities. All generated artifacts were restricted to technical and architectural information necessary for the initiative.

2 Proposed Approach

The proposed approach (Figure 1) was designed to support the reconstruction of technical and business knowledge from a poorly documented large enterprise system while simultaneously improving collective understanding and operational maintainability. The approach combines five complementary phases: (i) technical story decomposition, (ii) reverse engineering with abstraction-oriented refinement, (iii) AI-assisted documentation generation, (iv) visual flow reconstruction, and (v) collaborative knowledge-sharing sessions. Each phase is further explained in the next sections.

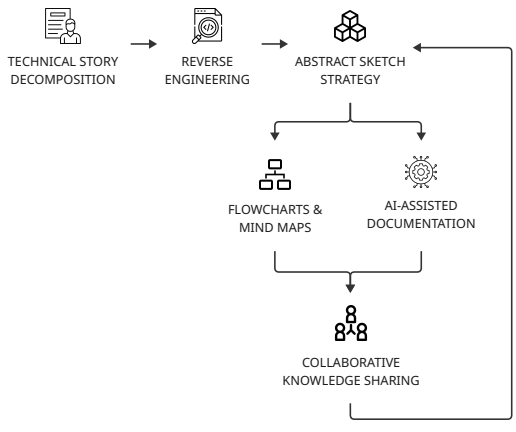

Figure 1: Proposed approach overview.

2.1 Technical Story Decomposition

During the first month of the team's creation, we conducted a discovery initiative. The process was designed to balance technical investigation, documentation generation, and collaborative knowledge sharing. In the context of this work, the term discovery refers to a structured software understanding process aimed at reconstructing technical and business knowledge from a software module with insufficient documentation and fragmented ownership.

The first challenge was understanding how to distribute the investigation among the developers on the team. Because the module had multiple interdependent business domains and there were no consolidated technical documents, an unstructured investigative process could produce duplicate analysis, fragments of knowledge, or an uneven distribution of work.

To reduce these risks, the team conducted a series of alignment meetings where they attempted to decompose the module into independent technical domains. The goal of the decomposition effort was to balance three factors: (i) functional cohesion in order to keep related flows together; (ii) the load of work in a manner that did not provide one developer with significantly larger amounts of complex analysis than another; and (iii) minimizing the overlap between investigations.

After the decomposition process, each identified domain was transformed into a dedicated technical story. Each developer became responsible for the reverse engineering, documentation, and technical analysis of a specific story. This strategy produced two important benefits. First, it reduced context switching during the investigation process, as studies have shown that working repeatedly in the same context is associated with higher productivity and frequent changes in context lead to distraction, stress, and loss of efficiency [4]. Second, it improved traceability by associating findings, identified gaps, technical tasks, and documentation artifacts with a single investigation scope.

2.2 Reverse Engineering and the Abstract Sketch Strategy

Given the absence of reliable documentation and the limited understanding of the module's actual runtime behavior, the team adopted a reverse engineering process centered on source code inspection, runtime analysis, and iterative flow reconstruction.

Initially, developers performed detailed investigations of their assigned domains by analyzing service interactions, business rules, API calls, integrations, database access patterns, and decision points. The first generated artifacts were highly detailed flow representations intended to capture the complete operational behavior of each subsystem.

During the first round of review cycles, the team reached consensus that producing diagrams with excessive detail compromised the documents' readability and made it difficult to view the module as a whole. To address this issue, the team created a strategy to increase the abstraction level of the diagrams, as it would reduce cognitive overload and allow people to focus only on the level of detail relevant to their current goal [3].

The strategy, called *Abstract Sketch Strategy*, consisted of refining generated flows iteratively by removing implementation details without compromising the original module's behavioral design. The

refined diagrams represent the primary business flow, main decision point(s), integration boundary(es), subsystem dependencies, and exceptional execution path(s) without documenting every technical operation at the implementation level. Implementation details, such as internal validations/external methods/infrastructure-related operations, were purposefully excluded from the refined diagrams unless required for explaining critical behavior.

The flow reconstruction and abstraction refinement activities became the most time-consuming stage of the initiative. Even though many implementation details were intentionally omitted from the final flowcharts, the members of the project team needed to be familiar with those details to ensure the semantic integrity of the abstract representations. At the end, instead of forcing readers to simultaneously process all implementation details, the abstract representations enabled progressive exploration of the module behavior while preserving the essential architectural and business-flow information.

2.3 AI-Assisted Documentation

After finishing flowchart generation using the *Abstract Sketch Strategy*, the team initiated the documentation formalization process.

The manually created notes and flowcharts were used as structured input for Large Language Models, including enterprise-grade AI-assisted tooling using a zero-shot approach [2]. These tools were employed to support multiple activities, including: (i) transformation of informal notes into structured documentation; (ii) identification of abstraction opportunities; (iii) refinement of textual explanations; (iv) support for reverse engineering interpretation.

The generated artifacts accelerated the documentation process by reducing the effort required to transform fragmented technical findings into cohesive visual and textual representations. However, despite the productivity gains, the team observed that the generated content frequently contained inconsistencies, incorrect flow interpretations, or missing business constraints. Approximately 66% of the generated artifacts required manual adjustments, refinements, or validation before becoming part of the official documentation. As a consequence, AI was treated as an acceleration mechanism rather than an authoritative source of truth. Human validation remained mandatory throughout the entire process.

2.4 Flowcharts and Global System Visualization

After consolidating the documentation, the team also created a global mind map connecting the domains analyzed by each developer. This provided an integrated view of the module and clarified how the independently analyzed subsystems interacted with each other.

The combination of flowcharts, mind maps, and collaborative review sessions significantly improved the team's collective understanding of the module and reduced isolated ownership of technical knowledge. It also helped the team to identify what could be improved in the mobility module and possible points of flaw.

2.5 Collaborative Knowledge Sharing

As developers finished their investigations, they conducted periodic knowledge-sharing meetings with the entire team. During these sessions, discovered gaps, architectural dependencies, business rules, and identified inconsistencies were collectively discussed.

This practice reduced isolated expertise and ensured that all developers acquired a broad understanding of the complete module instead of becoming specialists in isolated subsystems only. The final outcome included visual documentation, refined flowcharts, mind maps connecting module domains, and technical artifacts supporting future maintenance and onboarding activities.

Also, these sessions were not conducted only after the completion of the discovery activities. They continuously fed back into the process, as depicted in Figure 1. As developers shared findings, architectural interpretations, and identified inconsistencies, the team frequently revisited previously reconstructed flows and refined the abstraction level of the generated artifacts. This feedback loop aimed to reduce the probability of propagating incomplete or incorrect interpretations throughout the discovery initiative.

3 Results

The discovery and documentation initiative produced both operational and organizational improvements during the first four months after the creation of the dedicated team (from January to April 2026). We noted a reduction in bug and contact rates, and each is further detailed in the next sections.

3.1 Bug Reduction

One of the most significant outcomes was the reduction in bugs escalated from the support team to the development team. The company observed a 62.5% reduction in reported issues related to the mobility module. Figure 2 presents the monthly evolution of escalated bugs.

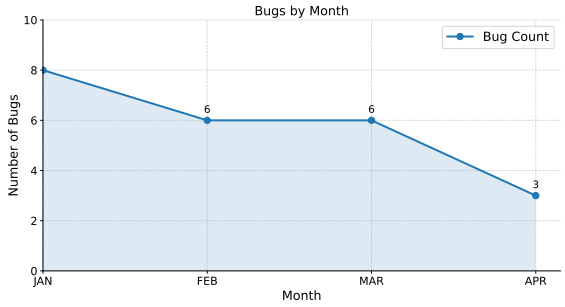

Figure 2: Evolution of escalated bugs during the initiative.

The observed reduction in escalated bugs was strongly associated with the increased understanding of the module obtained during the discovery process. As developers reconstructed the system behavior and documented hidden dependencies, the team became more capable of identifying fragile flows, incomplete business rules, and architectural inconsistencies before they produced operational incidents.

3.2 Contact Rate Reduction

The initiative also significantly reduced the module's Contact Rate (CR), defined as:

$$CR = \frac{N_{escalated\_tickets}}{N_{customers}} \times 100 \quad (1)$$

where $N_{escalated\_tickets}$ represents the number of customers whose support requests were escalated to the development team, and $N_{customers}$ represents the total number of customers using the mobility module during the evaluated period. Figure 3 presents the evolution of this indicator, which achieved a 90.13% reduction during the evaluated period.

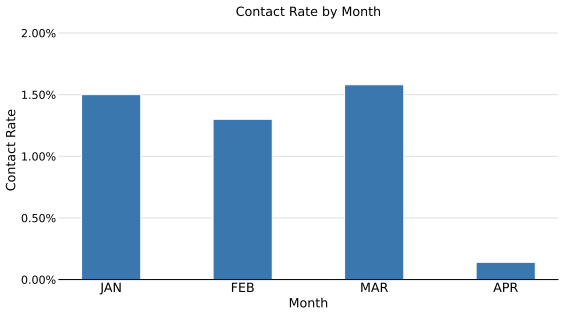

Figure 3: Evolution of CR during the initiative.

The reduction also produced operational benefits for both the development and support teams. As fewer customer issues required escalation to developers, the initiative reduced reactive maintenance demand and allowed the team to dedicate more effort to preventive improvements and long-term maintainability activities.

4 Lessons Learned

The experience generated several practical lessons regarding reverse engineering, collaborative documentation, and AI-assisted software engineering in industrial environments.

- **AI accelerates documentation but requires human validation:** AI tools substantially accelerated activities such as documentation writing, reverse engineering support, and flowchart generation. However, the generated artifacts frequently contained incomplete interpretations or inconsistencies with business rules. As a result, the team concluded that AI should be treated as a productivity accelerator rather than a reliable source of truth.
- **Abstraction improves readability:** Highly detailed diagrams frequently reduce understanding instead of improving it. Initially, the team produced extremely detailed flowcharts through reverse engineering activities, but these diagrams became difficult to navigate and maintain. The iterative refinement process demonstrated that higher-level abstractions significantly improved readability and communication effectiveness while still preserving the system behavior.
- **Shared knowledge increases team robustness:** Continuous knowledge-sharing meetings prevented the emergence of isolated specialists and increased collective ownership of the module. This practice improved technical discussions, reduced dependency on specific developers, and increased team robustness during maintenance activities.
- **Large enterprise systems may hide architectural gaps:** The reverse engineering process exposed some underexplored flows in the architecture. Many of them would remain hidden without a systematic investigation process focused on understanding the module as a whole.

5 Conclusion

This paper presented an industrial experience report describing the discovery, reverse engineering, and documentation process conducted over the mobility module at Paytrack[1]. The initiative emerged from a scenario characterized by high complexity, fragmented technical knowledge, reactive maintenance, and recurring operational issues in a business-critical large enterprise system. To address these challenges, a dedicated team adopted a collaborative process combining technical story decomposition, reverse engineering, abstraction-oriented documentation, AI-assisted artifact generation, and continuous knowledge-sharing sessions.

The obtained results demonstrated significant operational improvements, including reductions in escalated bugs and customer contact rate. Also, the initiative improved collective ownership of the module and increased the maintainability of the system. The experience also reinforced that AI tools can substantially accelerate documentation and reverse engineering activities, but still require careful human validation.

Additionally, the initiative increased the team's ability to discuss architectural improvements and identify structural issues proactively instead of reacting only after production incidents occurred. Future work includes refining AI-assisted workflows and replicating the proposed approach in other modules of the company.

Artifact Availability

Data will be made available on request.

References

[1] Gerardo Canfora, Massimiliano Di Penta, and Luigi Cerulo. 2011. Achievements and challenges in software reverse engineering. *Commun. ACM* 54, 4 (2011), 142–151.
[2] Hai Dang, Lukas Mecke, Florian Lehmann, Sven Goller, and Daniel Buschek. 2022. How to prompt? Opportunities and challenges of zero-and few-shot learning for human-AI interaction in creative applications of generative models. *arXiv preprint arXiv:2209.01390* (2022).
[3] Michael Jackson. 2012. Aspects of abstraction in software development. *Softw. Syst. Model.* 11, 4 (Oct. 2012), 495–511.
[4] Karina Kohl, Bogdan Vasilescu, and Rafael Prikladnicki. 2020. Multitasking Across Industry Projects: A Replication Study. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 93–100. doi:10.1145/3387940.3391495
[5] Ruiyin Li, Peng Liang, Mohamed Soliman, and Paris Avgeriou. 2022. Understanding software architecture erosion: A systematic mapping study. *J. Softw. Evol. Process* 34, 3 (March 2022), 45 pages. doi:10.1002/smr.2423
[6] Peter Rex Massingham. 2018. Measuring the impact of knowledge loss: a longitudinal study. *Journal of Knowledge Management* 22, 4 (2018), 721–758.
[7] Hanan Abdulwahab Siala, Kevin Lano, and Hessa Alfraihi. 2024. Model-driven approaches for reverse engineering—a systematic literature review. *IEEE Access* 12 (2024), 62558–62580.
[8] Davi Viana, Tayana Conte, Sabrina Marczak, Raymundo Ferreira, and Cleidson de Souza. 2015. Knowledge creation and loss within a software organization: An exploratory case study. In *2015 48th Hawaii International Conference on System Sciences*. IEEE, 3980–3989.