Addressing functionality gaps, data integrity, and system interoperability in enterprise systems

Matiss Gaigals*

*Riga Technical university, Kipsalas 6a, 1048, Riga, Latvia

Abstract

Mistakes and errors are an integral part of almost any complex enterprise system. Many of them face challenges, including limited functionality, data integrity issues, and interoperability problems. As a result, user operations may be blocked, requiring manual fixes. This paper proposes a method designed to detect such issues and provide AI-assisted temporary mitigations with user oversight. It has been implemented and evaluated by an experimental application, with the open-source code made publicly available. The method has been validated within three practical scenarios, demonstrating its effectiveness in handling tactical failures in enterprise systems.

Keywords

functionality gap, data integrity, system interoperability, ai-assisted error mitigation

1. Introduction

Modern enterprise systems become increasingly complex due to legacy dependencies, evolving business logic, and integration with heterogeneous services. Consequently, systems frequently suffer from functionality gaps, data integrity violations, and interoperability issues. Such errors may arise from edge cases, historic data model changes, or insufficient test coverage, and often remain undetected until users are affected in testing or production environments. Software engineers can address such issues quickly in development environments. Errors that reach production typically require costly, coordinated intervention involving diagnosis, patching, and even modification of customer data. This increases operational risk and cost, particularly when errors block users from completing their operations.

Advanced system monitoring tools, such as Sentry [1] or New Relic [2], typically notify software engineers only after failures occur. Usually, such a process heavily relies on manual diagnostics and code changes. This reactive paradigm often fails to meet modern enterprise resilience demands, contributing to increased recovery costs due to late-stage error detection.

Research on defect repair efficacy in enterprise resource planning (ERP) production systems [3] and ERP digital transformation [4] highlights the persistent nature of system repair and architectural adaptation challenges. Similarly, interoperability issues in integrated environments such as hospital ePrescribing [5] systems are well-documented. Interest in AI-based anomaly detection frameworks is increasing. For instance, tools such as ConAnomaly [6], Logformer [7], and HitAnomaly [8] demonstrate how AI/ML can proactively identify system log anomalies, indicating a growing shift toward autonomous and context-aware system management.

The systematic review by [9] further confirms the importance of runtime adaptation in self-adaptive systems, particularly for managing system uncertainty and evolving operational contexts. In response, this paper proposes an AI-assisted method for mitigating temporary errors, aimed at unblocking user operations in enterprise systems while preserving data and system integrity. The

BIR-WS 2025: BIR 2025 Workshops and Doctoral Consortium, 24th International Conference on Perspectives in Business Informatics Research (BIR 2025), September 17-19, 2025, Riga, Latvia.

* *Corresponding author.*

Matiss.Gaigals@edu.rtu.lv (M. Gaigals)

research approach aligns with empirically grounded design practices, as outlined by [10], which combines systematic literature reviews, design science, and scenario-based evaluation.

The research question addressed by the paper is as follows: How can errors that lead to functionality gaps, data integrity violations, and interoperability issues in enterprise systems be mitigated through automated detection and temporary AI-assisted actions that unblock user operations?

The paper addresses the ongoing challenge of handling complexity within enterprise systems by presenting a runtime approach that improves system resilience through AI-supported detection and automatic mitigation. The approach aligns with the managed complexity paradigm by addressing errors dynamically, reducing manual intervention, and improving the self-healing capabilities of enterprise software at the middleware level. The author proposes error event resolution with log anomaly detection as the extended approach.

An enterprise system refers to an integrated software platform that supports and automates the core operational processes, data flows, and information management across an organization. Following the Systems Engineering Body of Knowledge perspective on Enterprise Systems Engineering (ESE), an enterprise system operates across multiple layers: strategic business processes, information processing, and IT/software infrastructure.

This paper focuses on the software and information processing layers of enterprise systems, specifically middleware-level components responsible for runtime error detection, data consistency, and interoperability between subsystems, including ERP, CRM, and third-party APIs. The paper proposes a method for mitigating functionality gaps, data integrity violations, and interoperability issues in enterprise systems through the use of automated detection and agent-based AI techniques for temporary mitigation.

The intended users of the proposed method are enterprise system administrators, DevOps and software engineers, and experts in organizations responsible for maintaining data integrity and operational continuity. The method is designed to integrate into existing enterprise IT ecosystems as a middleware service, interfacing with ERP/CRM systems via APIs or event listeners. Detected errors and proposed mitigations are available through a dedicated administrator dashboard. Data can also be forwarded to issue-tracking systems or notification tools. The given design ensures corrective actions are traceable, thus minimizing disruption to ongoing operations. The rest of the paper is structured as follows: Section 2 describes related work, Section 3 presents the proposed method, and Section 4 concludes the paper.

2. Related Work

This section discusses related works on strategies, techniques, and methods that address (1) functionality gaps, (2) data errors, or (3) interoperability issues in enterprise systems. It also touches upon anomaly detection in enterprise systems. The related works help define the requirements for an AI-based method to address the issues mentioned above.

2.1. Functionality Gaps and Architecture Debts

Although researchers have focused only limited attention on missing functionality, two main observations can be made from the selected documents of the literature review.

First, functionality gaps appear due to wrong architectural, engineering, or management decisions, such as developing a system in an unsustainable manner. The result could be a system state where changes to functionality are almost impossible to implement [3], [4].

Second, resolving interoperability issues, which indirectly lead to potential functionality gaps, has a direct impact on the mitigation or resolution of missing functionality. Documents reviewed suggest that functionality gaps can be closed by identifying interoperability gaps and insufficiencies, thereby mitigating or resolving missing functionality [5] , [10].

Selecting documents related to enterprise architecture debts can lead to three main conclusions. First, researchers suggest that architectural debts must be identified and documented [11]. Second, several approaches to discovering given debts are proposed: special workshops, structured interviews, and software analysis [12] , [13] , [14]. Third, solutions must be implemented to prevent new debts and repay existing ones [11].

2.2. Data Integrity Provisioning Techniques

The analysis of the reviewed literature revealed limited findings regarding data integrity provisioning techniques, including data repair, in enterprise systems. Circular dependency on manual corrections and data repair due to poor system functionality, as covered by [3] , [4] , suggests the reactive nature of the potential issue.

From [4] , which explains how undocumented and extensive ERP system customization leads to a state with poor data and difficulties for further system development, it can be learned that manual and time-consuming data repair is a forced necessity and not a sustainable solution. Data quality decreases due to manual data changes, and "the digital transformation connected with the ERP change", by [4] , is a strategy for the organization as a potential solution.

The defect-repair cycle influence on ERP systems is discussed in [3] , which focuses on codebase repairs in the context of desired functionality. Similar conclusions could be reached regarding enterprise system repair in general, including data corrections. The source [3] classifies effective and defective repairs, where the first addresses the issue with a single attempt, while the second requires multiple attempts. The document demonstrates how defective repairs are more prevalent (~51% of repair effort dedicated to recurring issues) than effective ones, highlighting potential improvements in the repair process. Therefore, the repairs should be minor and have a potential reversible strategy to return the system to its previous state. Another suggestion is to assign a special "repair manager" to satisfy stakeholder requirements[3].

In the context of system migration, [10] explains specific data challenges that arise when transitioning from an on-premises system to a cloud-based system, as data migration can be nearly impossible. The document suggests that overcoming data model differences could be a solution. [15] focuses on the data integration phase of ERP implementation, where researchers propose the "extracting, transforming, and loading" method as one of the solutions to ensure successful data migration. In addition, the document reviews the list of application tools. As noted in [16], the Enterprise Services Bus (ESB) is used to ensure data integrity.

2.3. Interoperability in Enterprise Integration

Interoperability-related questions have gained comparatively high attention from researchers. For instance, although [5] does not propose technical solutions to improve (hospital) system interoperability, it discusses challenges and benefits of improved interoperability, which are discussed in other documents in this research. [17] defines the incompatibility of data models and the need for some intermediate tables as one of the challenges. [18] explains the intermediate table as a solution for interoperability, and both documents demonstrate how different architectural styles and design patterns solve interoperability issues.

Data exchange plays a paramount role in system interoperability. A deep-learning-based framework for enhanced data integration is proposed in [19]. Enterprise systems typically comprise multiple components, such as microservices, which engineers may want to deploy individually. An approach to maintaining compatibility between system parts is proposed in [20], and researchers describe their idea as "not to have schemas explicitly modeled by the developer (or API designer), but to derive these as well as the internal representations from a set of supported API definitions."

2.4. Anomaly Detection and Healing Solutions

While this paper does not directly address anomaly detection, it is included in the related work because it provides another perspective on the issues discussed in previous three subsections. Several documents discuss log-based and stream-based anomaly detection strategies. Researchers of [8] propose an improved "a log-based anomaly detection model utilizing a hierarchical transformer structure to model both log template sequences and parameter values" called "HitAnomaly." A different and slightly more advanced approach is proposed by [6] , which introduces "ConAnomaly" – "a log-based anomaly detection model composed of a log sequence encoder (log2vec) and multi-layer Long Short Term Memory Network (LSTMN)". The third and the most advanced among the reviewed is named "Logformer" by [7] , which is "with two cascaded transformer-based heads to capture latent contextual information from adjacent log entries, and leverage pre-trained embeddings based on logs to improve the representation of the embedding space".

The models described in the previous paragraph are examples of AI/ML-driven anomaly detection, including transformer-based, sequential, and context-aware approaches, as well as anomaly detection without log parsers and pre-trained embeddings for log analysis. Another example of ML usage is [21] , where researchers evaluate machine learning-based occupational fraud detection approaches by utilizing five ML algorithms.

A model proposed in [22] enhances graph link prediction, facilitating the detection of missing or incorrect relationships and enabling anomaly detection. Researchers in [22] employ "graph neural networks" to learn representations of system components, a "graph attention mechanism" to transfer knowledge across graphs, and, through inductive knowledge transfer, detect anomalies without retraining. Although researchers of [22] do not provide self-healing solutions, the ability to transfer knowledge from multiple graphs can aid in anomaly detection and thus address incomplete or missing data.

2.5. Main Findings from the Literature Review

To address functionality gaps, data integrity, and system interoperability in enterprise systems, the gaps or drawbacks of the listed approaches help to identify further potential requirements that could address the given issues.

A robust and at least partly automated solution is needed to address potential gaps and drawbacks related to missing functionality or architectural debts. Subjective human decisions or actions can lead to missing functionality. Another aspect is the speed of decision-making and acting because incomplete or long-lasting processes can introduce or provoke functionality gaps.

A proactive automated solution is needed to address gaps in data integrity issues, including data repair. Maintaining documentation or having a dedicated employee in the organization will not guarantee the mitigation of data integrity issues. A scalable solution for ensuring data integrity, including data repair, is needed to address the identified gaps and drawbacks of approaches to ensuring data integrity.

The interoperability-related issues cover many challenges, from short-term tactical to long-term strategic. This paper focuses on short-term tactical interoperability issues and their mitigation approaches. For instance, suppose an intermediate table was used for data synchronization between two systems. Still, due to some network issues, it was not executed, resulting in an outdated report on one of the systems. To address this interoperability issue, an automated solution is needed to detect and resolve synchronization issues.

In enterprise systems, log-based anomaly detection and graph link prediction approaches can be the most suitable; however, user oversight and interaction might be needed to reduce errors and increase customer value.

3. AI-assisted Error Mitigation

The author of the paper divides system issues into two main categories: short-term tactical problems and long-term strategic challenges. Short-term issues – like inconsistencies in date formats or misalignments in database schemas – are typically fixable by a dedicated feature team within a manageable period. In contrast, long-term strategic gaps, such as a lack of multi-tenancy support or missing external API integrations, require extensive system-wide modifications and may take several teams and months to implement.

This paper focuses on addressing short-term tactical issues while considering long-term strategic concerns for future research. Based on the main findings of the related work (sub-section 2.5), this section presents a high-level method to detect and resolve system deficiencies in enterprise systems. The method integrates error or anomaly detection, AI service-based issue mitigation, an automated repair application, and human oversight.

3.1. Method Requirements

According to the Framework for Evaluation in Design Science (FEDS) framework for Design Science evaluation by [23], the method requirements definition employs a "Quick & Simple strategy" that focuses on a small set of representative cases defined as addressable issues. Conducting formative evaluation on the selected instances enables efficient mitigation of design risks and provides an early utility indication of the artifact.

The functional requirements focus on three addressable issues: a functionality timeout, a data integrity violation, and an API attribute mismatch, as derived from the first three subsections of Section 2. The selection of addressable issues is methodologically grounded in the principles of Design Science Research. According to [23] , "design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation".

Three addressable issues are intentionally selected to reflect distinct categories of problems commonly faced in enterprise systems and are sufficient to evaluate the core capabilities of the proposed method. Each addressable issue is relevant to at least one finding from the related work's main findings (sub-section 2.5):

- Functionality timeout. Related work explains how functionality gaps occur when a system is developed in an unsustainable manner. While the missing functionality issue has received minimal researchers' attention, still, based on the knowledge and insights revealed by [3] , basic requirements related to the given addressable issues can be derived.
- Data integrity violation. As [3] explains: "by not documenting these prior customizations correctly, it directly impacts the data quality and indirectly hinders the ongoing ERP change because of poor data. Furthermore, there is a lot of data missing within the current system solution for the company to go fully digital," the orphaned records phenomenon matches this insight, thus allowing it to be included in addressable issues.
- API attribute mismatch. Although the API schema matching approach described by [18] can improve system interoperability, API changes or attribute mismatches are still risks. Based on this insight, they are included in the addressable issues.

The examples of detection and mitigation requirements are shown for each of the addressable issues in Table 1 to illustrate the expectations of the method better. The common functional requirement is that an AI service performs detection and mitigation, and the system offers user oversight with choices. The given requirements apply to both the base and extended approaches.

Table 1
Functional requirements for addressable issues

#	Addressable issue	Detection	Mitigation	User oversight
1	Functionality gaps: The application times out	When the application times	Evaluate if the database query can	Ask the user to choose AI service

	when a significant number of associated records must be loaded	out after a predefined time (seconds), flag the issue.	be optimized to load data faster.	action: a) improve database query, b) do nothing, and report this to the administrator dashboard.
2	Data integrity: The application crashes with error 500 due to referential integrity violations with orphaned records	When error 500 is returned, capture an exception to detect potential orphaned records causing inefficiencies.	Optimize the query dynamically to exclude orphaned records.	Ask the user to choose AI service action: a) temporarily exclude orphaned records, b) do nothing, and report this to the administrator dashboard.
3	Interoperability: third-party API attribute mismatch.	When the model validation error is returned, capture an exception to detect potential attribute mismatch.	Modify backend code to comply with third-party attribute requirements.	Ask the user to choose AI service action: a) agree on making code changes to match the API attributes, b) do nothing, and report to the administrator dashboard.

All non-functional requirements are related to the AI service since it is the leading actor in error resolution. Here, data integrity, security, and auditing are of paramount importance. Each non-functional requirement aligns with ISO/IEC 25010 product quality characteristics as explained by [24] .

3.2. High-Abstraction-Level Definition of the Method

Based on the defined requirements, the “base approach” and the “extended base approach” for the method are proposed. Both approaches include three main components: application, Healer service, and AI service. The second approach is extended with a log-based anomaly detection service.

As surveyed in [9] and [25], self-adaptive systems are classified into reactive and proactive types. Reactive systems respond only after an issue arises, such as exception handling. Proactive systems are anticipating or mitigating issues before they escalate. Citation:

“Self-adaptive approaches range from static, reactive, parametric solutions to dynamic, proactive, structural solutions. The former approaches are based on predetermined plans and configurations, while the latter approaches commonly leverage the power of AI/ML” [9].

The base approach aligns with reactive adaptation (catching timeouts and exceptions). By incorporating log monitoring and analysis, the extended approach is an example of proactive adaptation. It leverages runtime information to detect anomalies early and act preemptively—characteristics highlighted by [9] in the proactive self-adaptive system (SAS) description.

While the base approach is designed to mitigate errors and exceptions that have already occurred, the extended base approach attempts to reduce functionality, data integrity, or system interoperability degradation that might occur, according to log data.

Implementing the Model View Controller (MVC) pattern is a suitable use case for the “base approach”. The application MVC framework typically has built-in functionality, allowing it to catch timeouts or exceptions at the controller level. A Healer service can be implemented as classes of code

providing given functionality. AI services can be implemented either as an external API service, such as OpenAI [26], or as an entirely locally installed service, such as DeepSeek-Coder-V2 [27].

Figure 1, with a flowchart-like representation, explains the functions and interactions of the five components for the first approach at a high abstraction level. The second, “extended base approach,” includes three components from the first approach but extends their functionality and adds the fourth component — a log-based anomaly detection service. By adding the log-based anomaly detection service, the second approach can be suited for use cases when errors or anomalies cannot be detected by the MVC framework, for instance, slow queries within a predefined timeout or silent errors in background jobs.

Figure 2, with a flowchart-like representation, explains the functions and interactions of the six components for the second approach at a high abstraction level. A periodic job, for instance, once an hour, is triggered to capture the latest system logs for further anomaly detection. If the anomaly is detected, relevant log records are sent to the Healer service for further action. If any evidence of an error is found, the Healer service collects artifacts like the base approach and continues the process to prepare a prompt for the AI service.

For both approaches, the result is an “ErrorEvent" record with complete information of the error, prompt to the AI service, and its proposed method code, which is unit tested. For the new method to take effect and thus mitigate the previous error, both approaches must execute the latest code. Figure 3, with a flowchart-like representation, explains, on a high abstraction level, the functions and interactions of the application and the Healer service to find the previously prepared mitigation code and execute it to address missing functionality, data integrity, or system interoperability issues.

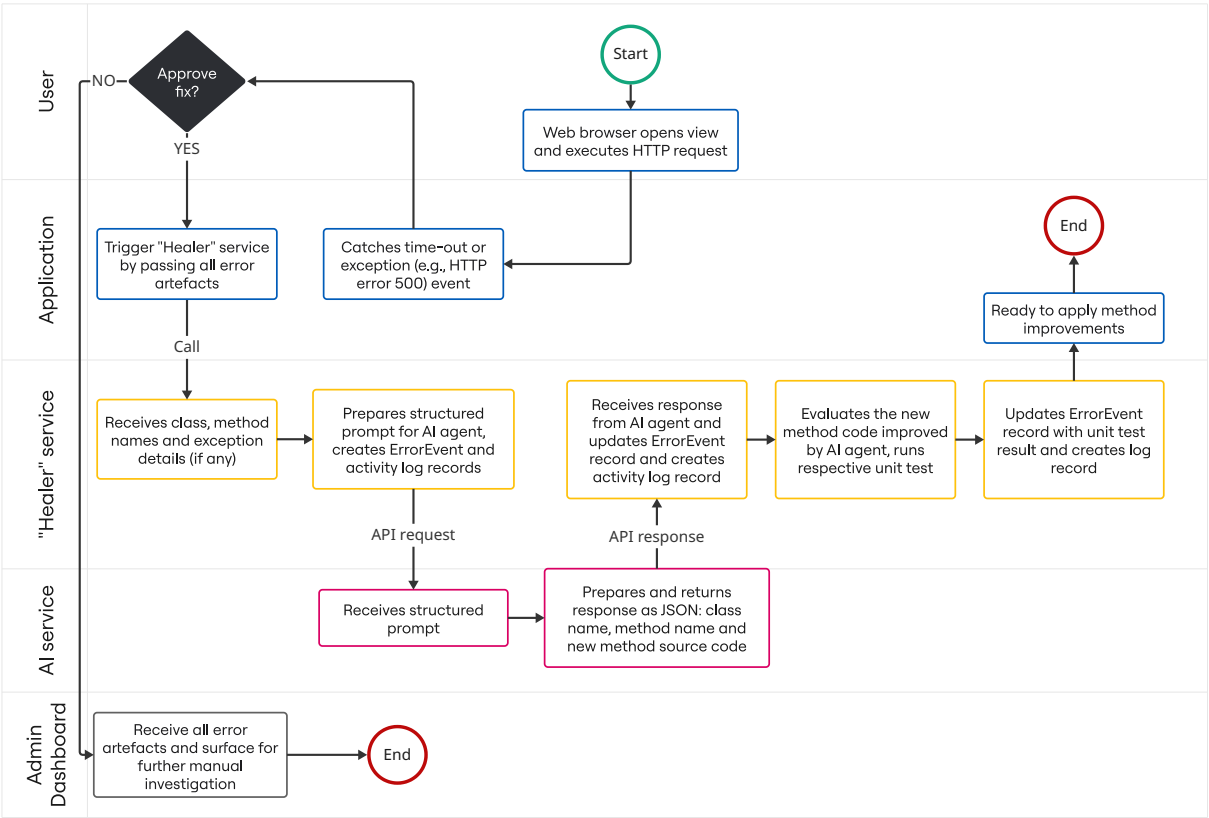

Figure 1: Error event resolution process for the base approach.

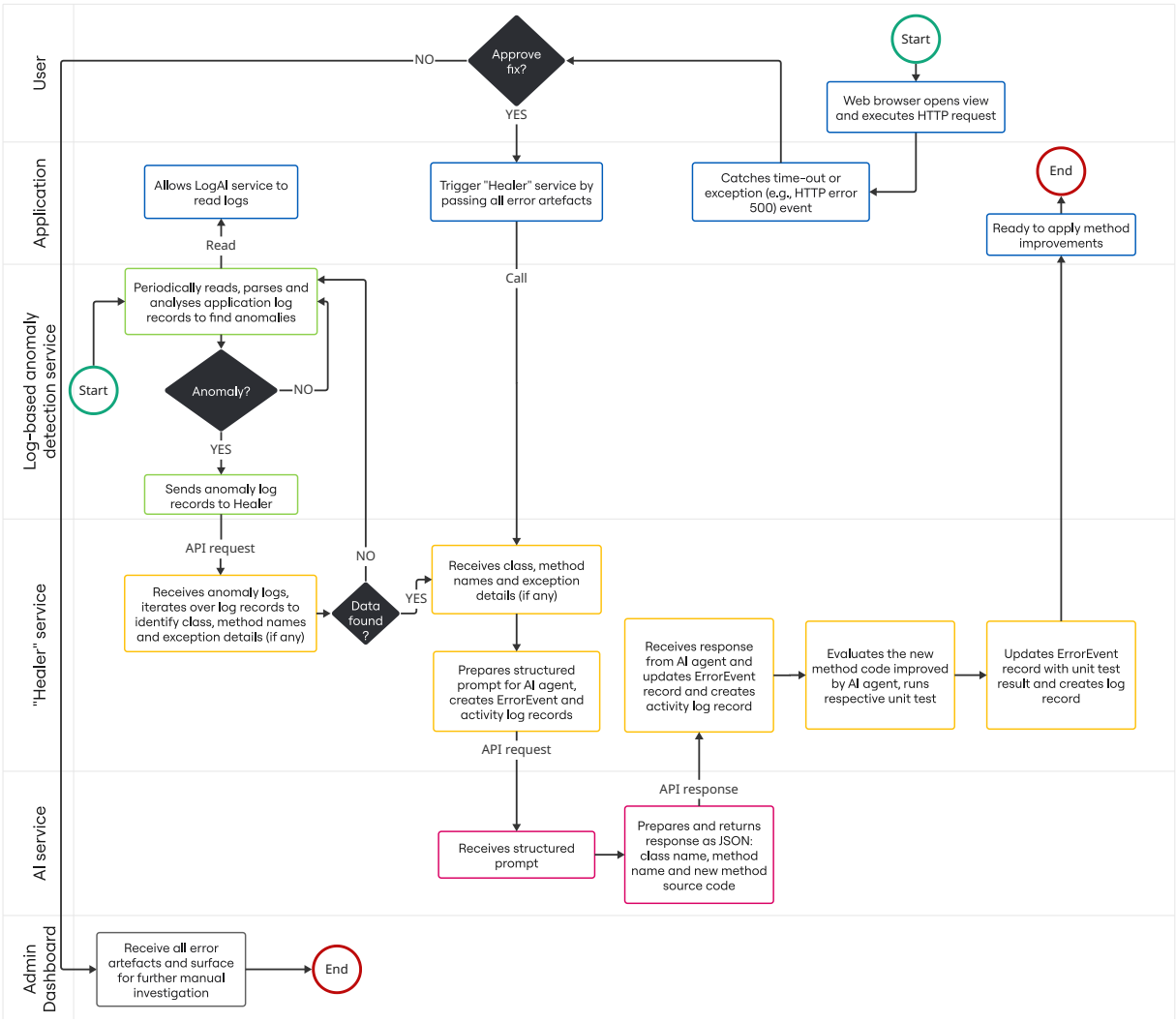

Figure 2: Error event resolution process with log anomaly detection.

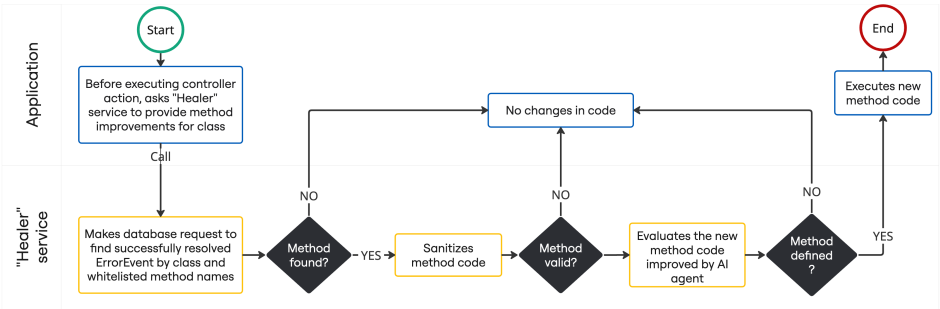

Figure 3: New method code execution process.

To ensure safe application of AI-generated fixes, the method includes a unit testing step before any patch is applied. Fixes are not executed immediately without verification; instead, the system runs automated unit tests to validate correctness (prototype executes unit tests only). In all cases, user or administrator oversight is required before any change takes effect, allowing preview and approval of proposed fixes. All actions are logged for auditing purposes, and rollback mechanisms are included to reverse unintended changes.

A simulated enterprise system example was created that utilizes the method. The Healer service was integrated into the monolith system with the main application; however, the main application

has an external API call to the "Catalog" service to simulate potential interoperability issues. The code of this practical experiment is made publicly available [28]. The author applied the basic approach to the demo application when a third-party API attribute mismatch caused a validation error. The system (1) caught the exception and created an ErrorEvent; (2) prepared an AI prompt and received a patch to translate the incoming attribute; (3) generated and ran unit tests, which passed; (4) displayed the change for user approval; and (5) applied the patch at runtime. After execution, the failing request completed successfully, and the view rendered without error.

3.3. Comparison with Related Frameworks and Tools

While Section 2 presents a synthesis of functionality gaps, data integrity, and interoperability challenges, this section highlights prior work most directly comparable to the proposed method and clarifies its distinct contributions. Table 2 presents a summarized structured comparison.

Table 2
Structured comparison

#	Approach/tool	Log detection	Automated repair	User oversight	Runtime action	Notes
1	ConAnomaly [6]	Yes	No	No	No	Detection only
2	HitAnomaly [8]	Yes	No	No	No	Detection only
3	Logformer [7]	Yes	No	No	No	Detection only
4	ACMA [21]	Partial (metrics)	No	No	No	Diagnosis focus
5	Sentry [1], New Relic [2]	Alerts/telemetry	No	Human in loop	No	Reactive monitoring
6	Author's proposed method	Yes (as extended)	Yes	Yes	Yes	Validated with unit tests

Tools like ConAnomaly [6] , HitAnomaly [8], and Logformer [7] have pushed forward the field of system log anomaly detection by incorporating techniques such as LSTM, transformer models, and pre-trained embeddings. These approaches are reasonably practical at pinpointing anomalies in system logs with high accuracy. Nonetheless, they mainly focus on detection and do not incorporate a real-time feedback mechanism to help address or fix issues once they are identified. The proposed method builds upon these advancements by coupling log-based detection (in the extended approach) with an AI-assisted mechanism that generates and tests resolution strategies in near real-time.

Similarly, frameworks like ACMA [21] link anomaly detection with root cause analysis by using causative metrics. Although effective for diagnosing performance issues, ACMA lacks mechanisms for automated repair or mitigation with user involvement. Conversely, the proposed method offers automated code patching and enables users to oversee the acceptance, rejection, or monitoring of fix execution.

Enterprise system middleware approaches, including ontology-based integration [29] and enhanced interoperability protocols [30], [31], [18], address structural and semantic discrepancies between systems. These initiatives typically require extensive, long-term changes to architecture or data modeling. Conversely, the proposed method operates in real-time during operation, providing immediate solutions for schema mismatches, such as changes to API attributes, without requiring major structural overhauls.

Furthermore, although tools such as Sentry [1] and New Relic [2] offer reactive monitoring with notifications to developers, they depend on manual remediation and lack autonomous corrective functionalities. The proposed method bridges this gap by embedding an AI-assisted repair loop that integrates detection, diagnosis, and patch proposal with test-backed validation and optional automated deployment. The key differentiators of this work are:

- Error detection and resolution as a unified middleware solution.
- Use of AI-generated code changes validated with automated unit tests.
- User or administrator oversight is built into the runtime process.
- Practical demonstration through an open-source prototype designed for enterprise systems.

These contributions address an underexplored gap in the literature on transitioning from intelligent error detection to immediate, context-aware mitigation in production environments.

4. Conclusions

This paper introduced a high-level method for runtime error detection and AI-assisted resolution targeting tactical-level issues in enterprise systems. The method comprises two complementary approaches: a reactive base approach and a proactive extended approach, both designed to unblock user operations and enhance system resilience.

The base approach detects errors at runtime, such as timeouts, data integrity violations, and API schema mismatches, and immediately proposes mitigation strategies through an AI-assisted feedback loop. These corrections are presented for user or administrator oversight and undergo automated validation before execution. This enables rapid response to tactical issues with minimal disruption to operations.

To further extend the scope of detection and align with the principles of proactive self-adaptation, the extended approach incorporates log-based anomaly detection. This enables identification of silent failures, performance degradations, or background anomalies that may not manifest as immediate runtime errors. By leveraging system logs, the method can preemptively trigger the mitigation workflow before users are affected, thus enhancing coverage and reducing time to resolution.

Together, these approaches address key enterprise challenges outlined in the paper:

- Functionality gaps: Reactive handling of execution failures and proactive detection of emerging usage anomalies, such as long-running queries.
- Data integrity: Runtime safeguards against data inconsistencies, supported by early anomaly signals in background processes.
- System interoperability: Dynamic adaptation to API changes and early warnings of schema mismatch trends.

This work demonstrates how tactical, AI-supported middleware solutions can alleviate the operational burden of complexity in enterprise systems, complementing architectural-level approaches emphasized in the research agenda on managed complexity. Future work will focus on expanding the method's scope through log anomaly detection, enhancing the security of corrected code execution, and evaluating the method in production-grade enterprise environments.

Declaration on Generative AI

During the preparation of this work, the author used Grammarly to check grammar and spelling. After using these tools, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

References

[1] Sentry, Sentry, error monitoring, https://sentry.io/product/error-monitoring/.

[2] New Relic, New Relic, application monitoring, https://newrelic.com/platform/application-monitoring.

[3] P. Van De Griend, R. Kusters, and J. Trienekens, Defect repair efficacy in ERP production systems, in: Procedia Computer Science, 2024, pp. 1353–1360. doi: 10.1016/j.procs.2024.06.306.

[4] P. Aasi, E. Gråhns, R. Geijer, and L. Rusu, Organizational aspects in achieving a successful digital transformation: case of an ERP system change, vol. 437 LNBIP. 2022. doi: 10.1007/978-3-030-95947-0_46.

[5] C. Heeney, M. Bouamrane, S. Malden, K. Cresswell, R. Williams, and A. Sheikh, Optimising ePrescribing in hospitals through the interoperability of systems and processes: a qualitative study in the UK, US, Norway and the Netherlands, BMC Med Inform Decis Mak, vol. 23, no. 1, 2023, doi: 10.1186/s12911-023-02316-y.

[6] D. Lv, N. Luktarhan, and Y. Chen, Conanomaly: content-based anomaly detection for system logs, Sensors, vol. 21, no. 18, 2021, doi: 10.3390/s21186125.

[7] F. Hang, W. Guo, H. Chen, L. Xie, C. Zhou, and Y. Liu, Logformer: cascaded transformer for system log anomaly detection, CMES - Computer Modeling in Engineering and Sciences, vol. 136, no. 1, pp. 517–529, 2023, doi: 10.32604/cmes.2023.025774.

[8] S. Huang et al., HitAnomaly: hierarchical transformers for anomaly detection in system log, IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2064–2076, 2020, doi: 10.1109/TNSM.2020.3034647.

[9] T. Wong, M. Wagner, and C. Treude, Self-adaptive systems: a systematic literature review across categories and domains, Inf Softw Technol, vol. 148, 2022, doi: 10.1016/j.infsof.2022.106934.

[10] J. Kinnunen, Leaving the lights on? Exploring cloud ERP migrations and IS discontinuance, in: 2021 IEEE 12th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2021, 2021, pp. 58–64. doi: 10.1109/UEMCON53757.2021.9666669.

[11] P. Alexander, S. Hacks, J. Jung, H. Lichter, U. Steffens, and Ö. Uludağ, A framework for managing enterprise architecture debts - outline and research directions, in: CEUR Workshop Proceedings, 2020, pp. 5–10.

[12] S. Daoudi, M. Larsson, S. Hacks, and J. Jung, Discovering and assessing enterprise architecture debts, Complex Systems Informatics and Modeling Quarterly, vol. 2023, no. 35, pp. 1–29, 2023, doi: 10.7250/csimq.2023-35.01.

[13] J. Jung, S. Hacks, T. De Gooijer, M. Kinnunen, and K. Rehring, Revealing common enterprise architecture debts: conceptualization and critical reflection on a workshop format industry experience report," in Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOCW, 2021, pp. 271–278. doi: 10.1109/EDOCW52865.2021.00058.

[14] B. Tieu and S. Hacks, Determining enterprise architecture smells from software architecture smells," in Proceedings - 2021 IEEE 23rd Conference on Business Informatics, CBI 2021 - Main Papers, 2021, pp. 134–142. doi: 10.1109/CBI52690.2021.10064.

[15] M. Elbadri, A. Altaher, and S. Alkawan, Data quality considerations for ERP implementation: techniques for effective data management, vol. 2097 CCIS. 2024. doi: 10.1007/978-3-031-62624-1_17.

[16] F. Muslimin, A. N. Fajar, and Meyliana, Business process management and service oriented architecture integration for transactional banking application, in: 7th International Conference on ICT for Smart Society: AIoT for Smart Society, ICISS 2020 - Proceeding, 2020. doi: 10.1109/ICISS50791.2020.9307597.

[17] R. Kiesel, A. Weiss, and R. H. Schmitt, Filling the semantics gap in industrial communication: middleware+ for an internet of production, in: IFAC-PapersOnLine, 2021, pp. 432–437. doi: 10.1016/j.ifacol.2021.08.049.

[18] Q. Tong, X. Ming, and X. Zhang, The realization for automated warehouse based on the integration of ERP and WMS, in: ACM International Conference Proceeding Series, 2020, pp. 76–80. doi: 10.1145/3412953.3412954.

[19] H. Lin, Construction of fuzzy ERP data analysis system based on deep learning, in: Proceedings of the 3rd International Conference on Intelligent Communication Technologies and

Virtual Mobile Networks, ICICV 2021, 2021, pp. 855–858. doi: 10.1109/ICICV50876.2021.9388481.
[20] H. Knoche and W. Hasselbring, Continuous API evolution in heterogenous enterprise software systems, in: Proceedings - IEEE 18th International Conference on Software Architecture, ICSA 2021, 2021, pp. 58–68. doi: 10.1109/ICSA51549.2021.00014.
[21] S. Kim, J. S. Kim, and H. P. In, Multitier web system reliability: identifying causative metrics and analyzing performance anomaly using a regression model, Sensors, vol. 23, no. 4, 2023, doi: 10.3390/s23041919.
[22] J. Hao et al., Multi-source Inductive Knowledge Graph Transfer, vol. 13714 LNAI. 2023. doi: 10.1007/978-3-031-26390-3_10.
[23] A. R. Hevner, S. T. March, J. Park, and S. Ram, Design science in information systems research, MIS Q, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.
[24] J. Estdale and E. Georgiadou, Applying the ISO/IEC 25010 quality models to software product, vol. 896. 2018. doi: 10.1007/978-3-319-97925-0_42.
[25] D. Weyns, Engineering self-adaptive software systems - an organized tour, in: Proceedings - 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2018, 2019, pp. 1–2. doi: 10.1109/FAS-W.2018.00012.
[26] OpenAI, OpenAI, https://platform.openai.com/docs/overview.
[27] DeepSeek, DeepSeek Coder V2, https://github.com/deepseek-ai/DeepSeek-Coder-V2, Mar. 2025.
[28] Matiss Gaigals, Healer GitHub, https://github.com/matissg/healer.
[29] A. Polenghi, I. Roda, M. Macchi, A. Pozzetti, and H. Panetto, Knowledge reuse for ontology modelling in maintenance and Iindustrial asset management, J Ind Inf Integr, vol. 27, 2022, doi: 10.1016/j.jii.2021.100298.
[30] N. Prilya Nugraha, M. Saputra, and W. Puspitasari, Corrective maintenance configuration to optimize CPO production using SAP," in Proceedings - International Conference Advancement in Data Science, E-Learning and Information Systems, ICADEIS 2022, 2022. doi: 10.1109/ICADEIS56544.2022.10037424.
[31] X. Ye, M. Yu, W. S. Song, and S. H. Hong, An asset administration shell method for data exchange between manufacturing software applications, IEEE Access, vol. 9, pp. 144171–144178, 2021, doi: 10.1109/ACCESS.2021.3122175.