Open Science
BRAZIL

DOI: https://doi.org/10.37497/opsbrazil.38

PREPRINT

FROM ALERT TO PATCH: MCP AND APM IN AI AGENT-ASSISTED BUG RESOLUTION

Do Alerta ao Patch: MCP e APM na Resolução Assistida de Bugs por Agentes de Inteligência Artificial

FELIPE O. B. SOUZA
Independent researcher in the fields of Software Engineering, Artificial Intelligence, and Distributed Systems
E-mail: fobsouza@gmail.com

ABSTRACT | The increasing complexity of distributed digital systems has amplified the challenges associated with software failure diagnosis and resolution. In modern environments, essential information for incident investigation is distributed across multiple sources, including alerts, traces, logs, metrics, deployment records, technical documentation, and source code. Although artificial intelligence-based agents have demonstrated the ability to analyze code, execute tests, and propose modifications, a scientific gap remains regarding how structured access to operational context influences the quality and efficiency of these activities. This paper presents an experimental design to investigate whether read-only access to observability data through the Model Context Protocol (MCP) can assist artificial intelligence agents in diagnosing failures and producing verifiable patches. The study proposes a comparison among three experimental conditions: investigation based on static information, investigation supported by manual Application Performance Monitoring (APM) tools, and investigation assisted by agents connected to MCP-enabled tools. The research does not present results from the controlled experiment and does not assume prior improvements in time, accuracy, or quality. However, it includes motivation based on anonymized industrial observations conducted with an MCP-assisted agent and a commercial APM platform. The objective is to establish a reproducible protocol capable of evaluating different dimensions of the software bug resolution process, including time to identify the root cause, quality of produced patches, evidence used, need for human intervention, and risks associated with the use of intelligent agents. The expected contribution is to provide a scientific methodology for evaluating how context architectures can influence the performance of artificial intelligence agents in complex software engineering tasks, promoting greater transparency, traceability, and reproducibility.

Keywords | Artificial Intelligence; Software Engineering; Intelligent Agents; Model Context Protocol; Application Performance Monitoring; Observability; Failure Diagnosis; Automated Bug Resolution.

Submetido: 30-07-2026
Postado: 30-07-2026
e-ISSN: 2764-9822

Endereço de correspondência:
Felipe O. B. Souza – E-mail: fobsouza@gmail.com

Alumni.in

RESUMO | A crescente complexidade dos sistemas digitais distribuídos tem ampliado os desafios relacionados ao diagnóstico e à resolução de falhas de software. Em ambientes modernos, informações essenciais para investigação de incidentes encontram-se distribuídas entre diferentes fontes, incluindo alertas, traces, logs, métricas, registros de implantação, documentação técnica e código-fonte. Embora agentes baseados em inteligência artificial apresentem capacidade para analisar código, executar testes e propor alterações, permanece uma lacuna científica sobre como o acesso estruturado ao contexto operacional influencia a qualidade e a eficiência dessas atividades. Este artigo apresenta um protocolo experimental para investigar se o acesso somente leitura a dados de observabilidade por meio do Model Context Protocol (MCP) pode auxiliar agentes de inteligência artificial no diagnóstico de falhas e na produção de patches verificáveis. O estudo propõe comparar três condições experimentais: investigação baseada em informações estáticas, investigação com apoio manual de ferramentas de Application Performance Monitoring (APM) e investigação assistida por agentes conectados a ferramentas MCP. A pesquisa não apresenta resultados experimentais concluídos e não assume ganhos antecipados de tempo, precisão ou qualidade. Entretanto, apresenta motivação baseada em observações industriais anonimizadas realizadas com agente assistido por MCP e plataforma comercial de APM. O objetivo é estabelecer uma metodologia reproduzível capaz de avaliar diferentes dimensões do processo de resolução de bugs, incluindo tempo até identificação da causa raiz, qualidade dos patches produzidos, evidências utilizadas, necessidade de intervenção humana e riscos associados ao uso de agentes inteligentes.
Palavras-chave | Inteligência Artificial; Engenharia de Software; Agentes Inteligentes; Model Context Protocol; Observabilidade

1. INTRODUCTION

The evolution of digital systems has profoundly transformed the way software applications are developed, deployed, and maintained. Modern architectures based on microservices, cloud computing, and distributed integrations have enabled greater scalability and flexibility; however, they have also increased the operational complexity of software environments.

In traditional systems, failure identification often relied on direct code analysis or the examination of specific application records. However, in distributed environments, a single issue may involve multiple services, external dependencies, recent configuration changes, variations in workload, and different infrastructure layers.

In this scenario, incident investigation has become an activity based on the correlation of multiple sources of evidence. An initial alert may indicate only a symptom, while the actual cause may be recorded in a distributed trace, a recent deployment change, a specific configuration, or an architectural decision embedded in the source code.

Observability tools, such as Application Performance Monitoring (APM) platforms, were developed to support professionals in collecting and analyzing this information. Logs, metrics, and traces enable the understanding of system behavior and the identification of patterns associated with failures. However, these sources of information typically remain separated across different tools, requiring specialized knowledge for interpretation.

At the same time, advances in generative artificial intelligence models have enabled the emergence of agents capable of performing increasingly complex software engineering

tasks. These agents can analyze repositories, interpret source code, execute commands, suggest modifications, and generate patches for problem resolution.

Despite these capabilities, the efficiency of such agents directly depends on the information available during their decision-making process. An agent may possess advanced reasoning capabilities but still produce an inadequate solution if provided with incomplete, outdated, or insufficiently contextualized information.

This highlights the importance of **context engineering**, a concept related to the selection, organization, and delivery of the information required for intelligent systems to perform tasks with greater accuracy and safety.

The **Model Context Protocol (MCP)** introduces a structured approach for connecting artificial intelligence models with external tools, enabling controlled access to information and resources. When applied to observability environments, MCP can establish a bridge between intelligent agents and operational data, allowing investigations to be conducted based on real evidence of system behavior.

However, technical feasibility does not necessarily represent a proven improvement. Providing access to additional information may also introduce new challenges, including data overload, incorrect interpretation of signals, inappropriate exposure of information, and the creation of corrections that address symptoms rather than fundamental causes.

Therefore, this study proposes an experimental protocol to scientifically evaluate the impact of structured, read-only access to telemetry data on the process of AI-assisted software failure diagnosis and correction.

The research aims not only to determine whether agents can find solutions, but also to understand how these solutions are reached, which evidence is used, what limitations emerge, and which risks remain even in controlled environments.

2. THEORETICAL BACKGROUND

2.1 Artificial Intelligence Applied to Software Engineering

The use of artificial intelligence in software engineering has evolved from simple automation-based support tools into systems capable of participating in multiple stages of the software development lifecycle. Large Language Models (LLMs) have demonstrated the ability to interpret code, suggest modifications, explain behaviors, and assist in solving technical problems.

With the advancement of these models, intelligent agents capable of executing sequences of actions, querying external sources, using tools, and adapting their strategies according to predefined objectives have emerged. Unlike traditional code suggestion systems, agents possess greater operational autonomy, allowing them to participate in activities such as failure analysis, test creation, and correction generation.

However, the ability to generate responses does not automatically guarantee the production of correct solutions. In real software environments, decisions depend on

contextual information, business rules, change history, and accumulated knowledge regarding the existing architecture.

The gap between a technically plausible response and a truly appropriate solution represents one of the main challenges in the adoption of artificial intelligence agents for software engineering activities.

2.2 Failure Diagnosis in Distributed Systems

Failure diagnosis in distributed systems is a complex activity due to the number and diversity of components involved. Modern applications typically rely on multiple independent services, databases, message queues, external APIs, and infrastructure layers.

A failure perceived by users may originate far from the point where the error becomes visible. For example, an increase in response time may be related to database issues, unavailability of an external service, configuration changes, or unexpected behavior after a software update.

In this context, **Root Cause Analysis (RCA)** depends on the ability to gather different sources of evidence and establish relationships between events.

Alerts provide initial information regarding symptoms. Logs contain detailed execution records. Metrics reveal behavioral changes in the system. Distributed traces enable the tracking of requests across multiple services. Deployment records indicate recent changes that may be associated with the incident.

The main challenge lies in transforming these fragmented signals into a coherent explanation and a reliable corrective action.

2.3 Observability and Application Performance Monitoring (APM)

Observability represents the ability to understand the internal state of a system based on the signals generated during its operation. Unlike traditional monitoring, which focuses on verifying whether specific indicators exceed predefined thresholds, observability enables the investigation of unexpected behaviors and the identification of probable causes.

The three primary sources of observability are:

- **Logs:** detailed records of events that occurred during system execution;
- **Metrics:** quantitative values representing system behavior;
- **Traces:** records describing the path of a request across different components.

Application Performance Monitoring (APM) platforms combine these elements to provide operational visibility. They enable the identification of bottlenecks, performance changes, and incident investigation.

Despite the relevance of these tools, their traditional use depends heavily on human interpretation. Engineers must navigate across multiple systems, formulate hypotheses, and manually correlate information.

The integration between observability platforms and intelligent agents represents an opportunity to reduce this fragmentation; however, scientific evaluation is required to determine its actual benefits and limitations.

2.4 Model Context Protocol (MCP) and Context Architecture

The **Model Context Protocol (MCP)** emerges as a proposal for standardizing communication between artificial intelligence models and external tools. Its objective is to enable agents to access structured information and resources required for performing specific tasks.

Within the context of software engineering, this approach allows agents to connect with sources such as repositories, observability systems, documentation databases, and operational tools.

However, providing an agent with more information does not necessarily result in better decisions. The challenge lies in defining which information should be available, what level of access should be granted, and how to ensure that the provided data is reliable.

Context engineering addresses precisely this organization process: selecting relevant information, establishing priorities among data sources, controlling permissions, and maintaining decision traceability.

In a bug diagnosis environment, an agent does not need only to know that an error occurred. It needs to understand when the failure happened, which changes preceded the issue, which components were affected, and which constraints must be respected during the correction process.

2.5 Automated Patches and Correction Validation

Automated patch generation represents one of the most promising applications of artificial intelligence in software engineering. Agents can analyze failures, propose code modifications, and execute tests to verify potential corrections.

However, a syntactically valid correction does not necessarily represent an adequate solution. A patch may remove an error message without addressing the underlying cause, introduce new problems, or alter expected system behavior.

Therefore, agent evaluation should consider not only their ability to produce modifications but also the quality of those modifications.

Criteria such as patch acceptance, regression test execution, adherence to the original scope, and the quality of the supporting evidence become essential for scientific evaluation.

2.6 Related Work on Diagnosis Assisted by Language Models

Recent studies have investigated the use of language models in root cause analysis activities within cloud environments, evaluating the ability of these models to suggest probable causes and mitigation steps based on incident descriptions. Subsequent research extended this approach to agents capable of querying external sources during investigations.

In parallel, the literature on automated program repair has established criteria for evaluating automatically generated corrections, including the distinction between patches that satisfy available tests and patches that effectively resolve the underlying defect. Recent agent benchmarks have evaluated the resolution of real repository issues using predominantly static context, while reasoning-and-action architectures have established patterns based on alternating between inference and tool usage.

However, these research directions tend to evaluate different extremes of the process. Root cause analysis studies rarely require the production of a verifiable patch, while automated repair benchmarks rarely provide access to live operational telemetry from running systems.

The complete evaluation chain remains insufficiently explored: a scenario in which the same agent begins from an operational signal, queries observability data in read-only mode, and produces a correction submitted to validation.

3. RESEARCH GAP

Although previous studies have investigated applications of artificial intelligence in failure detection, code analysis, and development task automation, limitations remain regarding the evaluation of the complete process of AI-assisted incident resolution.

A significant portion of existing research focuses on isolated stages, such as defect localization or code generation. However, solving a real-world software problem involves a broader sequence:

alert detection → evidence collection → hypothesis formulation → root cause identification → patch implementation → correction validation

Success in one stage does not necessarily guarantee success in subsequent stages.

An agent may correctly identify the affected service but incorrectly interpret the cause of the failure. Likewise, it may identify the root cause but generate an inappropriate or insufficiently tested modification.

Furthermore, there is still limited understanding of the specific impact of providing structured, read-only access to operational telemetry.

This study seeks to address this research gap by proposing an experimental evaluation comparing different approaches to contextual access during software failure diagnosis and correction activities.

3.1 Industrial Motivation and Preliminary Observations

Before the controlled experiment, exploratory observations were conducted in a production environment of a large financial technology company in the United States, using an artificial intelligence agent with read-only access to a commercial Application Performance Monitoring (APM) platform through the Model Context Protocol (MCP).

The investigation focused on a GraphQL API gateway responsible for customer-facing traffic in a production environment, within the context of GraphQL Denial-of-Service (DoS) mitigation and query complexity abuse prevention. The environment employed operation limits (**maximum aliases, maximum root fields, depth, and height**) and edge-level rate limiting.

During approximately 17 days of telemetry observation:

- typical traffic presented daily maximum values close to 3 aliases/root fields;
- before operation limits enforcement, peaks of 200 and 700 aliases/root fields were observed;
- after deployment of operation limits (**max aliases/root fields = 250**) with active enforcement, additional outliers of 100 and 50 were still detected;
- during the following six days, the daily maximum value stabilized again at 3;
- query depth never exceeded 10, remaining below the configured maximum depth of 15.

Two findings were particularly relevant for the experimental protocol.

First, telemetry completeness was uneven: aggregated query complexity metrics identified outliers that were not retained by the APM platform as correlatable samples, preventing the identification of the GraphQL operation, client identity, or authentication context. Rejected requests caused by operation limits (HTTP 400) also did not consistently increase the daily maximum values recorded by aggregated metrics, indicating potential undercounting.

Second, the MCP-assisted agent was able to correlate temporal windows, synthetic user agents, and deployments, generating hypotheses and discarding incorrect causal assumptions (for example, authentication or authorization incidents occurring within the same timeframe but mechanically unrelated to rate limiting or operation limits). Nevertheless, the final operational decision — maintaining conservative thresholds until telemetry quality improved — required human judgment due to incomplete evidence.

These observations do not represent experimental results from the proposed protocol. Instead, they provide empirical motivation for Research Questions RQ3–RQ5 and Hypotheses H3–H4: structured access to observability data may accelerate investigation, but it does not eliminate risks related to incomplete sampling, excessive confidence, and premature decisions based on insufficient evidence.

Table 1 summarizes the daily maximum values observed during the analyzed period. The days are presented relative to the beginning of data collection, and the values were preserved without transformation to maintain visibility into the relationship between typical traffic patterns and observed outliers.

Table 1. Daily maximum query complexity values (aliases/root fields, depth, and height) observed during an approximately seventeen-day preliminary industrial telemetry window.

Relative Day	Aliases / Root Fields	Depth	Height	Observation
D0	3	10	671	Beginning of query complexity telemetry collection
D1	200	10	671	Peak before operation limits enforcement
D2	50	10	671	—
D3	700	10	1400	Highest observed value; operation limits (250) deployed on the same day
D4–D7	3	10	671	Typical traffic pattern
D8	100	10	671	Post-enforcement outlier without retained APM sample
D9–D10	3	10	671	Typical traffic pattern
D11	50	10	671	Post-enforcement outlier without retained APM sample
D12–D17	3	10	659	Stable baseline (aliases/root fields = 3) for six days

The ratio between typical traffic and the highest observed value exceeded two orders of magnitude, characterizing a scenario in which average values provide limited information and analysis depends primarily on extreme distribution behavior — a pattern commonly associated with query complexity abuse and rate limit/operation limit tuning.

Throughout the observation window, only eleven one-minute intervals exceeded the conservative threshold initially proposed (aliases > 30), while query depth showed no occurrence close to the configured maximum depth.

The retained samples presented characteristics compatible with synthetic traffic, including payloads of approximately twelve and fifty-five kilobytes originating from command-line clients (curl), without associated authentication information. A subsequent attempt containing more than one thousand repeated aliases was rejected by the gateway with HTTP 400 (Bad Request) due to operation limit enforcement and, nevertheless, did not increase the maximum daily values recorded by aggregated metrics.

3.2 Taxonomy of Observed Telemetry Gaps

The challenges encountered during the assisted investigation were not caused by the absence of tools, but rather by the way telemetry data was structured. Table 2 organizes these gaps into reusable categories that will serve as the basis for the controlled context manipulation proposed in Research Question RQ3.

Table 2. Observed telemetry gaps and their implications for agent-assisted investigation.

Telemetry Gap	Observed Manifestation	Implication for the Agent
Partial sampling	Outliers present in metrics without retained execution samples	Inability to attribute the event to a specific operation or client
Rejected request undercounting	Rejected requests do not consistently increase aggregated maximum values	Underestimation of the actual phenomenon frequency
Absence of identity dimensions	Metrics without operation or client labels	Hypotheses can be formulated but cannot be verified
Absence of request content in samples	Trace without the GraphQL operation document	Probable cause cannot be confirmed
Aggregation granularity	Daily maximum values hide per-minute event frequency	Frequency is underrepresented during analysis
Weak caller identity	Lack of authentication and client identification	Attribution becomes limited to heuristics, such as user-agent analysis

This taxonomy is relevant to the experimental design because it enables telemetry degradation to be performed systematically rather than arbitrarily. Each category can be activated or disabled within a failure scenario, producing comparable experimental conditions across different executions.

3.3 Observed Assisted Investigation Workflow

The preliminary investigation followed a recurring sequence that was later adopted as a reference model for protocol instrumentation:

aggregated signal → temporal refinement of the series → correlation with execution samples → verification of caller identity → correlation with change records → formulation and elimination of hypotheses → explicit declaration of evidence gaps → recommendation submitted for human decision-making

Two stages of this workflow are frequently absent from evaluations of agents focused exclusively on code generation.

The first is the explicit elimination of competing hypotheses, observed when incidents occurring within the same timeframe but with different mechanisms were correctly separated from the investigated change.

The second is the declaration of insufficient evidence, which preceded the recommendation to maintain the existing configuration rather than modify it.

It should also be noted that telemetry access occurred through a commercial APM platform integrated via MCP. Platforms in this category, such as Datadog, New Relic, and Dynatrace, provide equivalent programmatic interfaces; therefore, the proposed protocol is not dependent on a specific vendor.

4. RESEARCH OBJECTIVES

4.1 General Objective

To experimentally evaluate whether structured, read-only access to observability data through the Model Context Protocol influences the performance of artificial intelligence agents in identifying root causes and producing verifiable patches for software failures.

4.2 Specific Objectives

Compare different strategies for accessing operational context during bug investigations;

- Evaluate the impact of MCP on the time required to identify root causes;
- Measure the quality of patches produced by intelligent agents;
- Analyze the relationship between telemetry completeness and agent confidence;
- Identify risks associated with the use of agents connected to operational tools;
- Develop a reproducible protocol for future research on AI agents applied to software engineering.

5. RESEARCH QUESTIONS AND HYPOTHESES

This research is guided by questions aimed at understanding not only whether artificial intelligence agents can assist in failure resolution, but also how structured access to operational context influences their decisions.

The objective is not solely to evaluate the final outcome, but to analyze the entire reasoning process followed by the agent during investigation: which information sources were consulted, which hypotheses were formulated, which evidence supported decisions, and how the correction was validated.

5.1 Research Questions

RQ1 – Does structured telemetry access through MCP reduce the time required to correctly identify the root cause of a failure?

This question investigates whether reducing fragmentation of operational information can accelerate diagnosis when compared with traditional approaches based on static data or manual investigation.

RQ2 – Do agents with access to MCP-enabled tools produce patches with higher acceptance and test validation rates?

This question evaluates whether greater contextual availability contributes to technically superior corrections, considering criteria such as patch acceptance, regression test execution, and absence of side effects.

RQ3 – How do telemetry quality and completeness influence agent performance and confidence?

This question seeks to understand whether agents behave differently when provided with complete, incomplete, or potentially ambiguous information.

This question is empirically motivated by cases in which production metrics identified outliers, but the APM platform did not retain correlatable samples — a situation in which telemetry completeness directly limits diagnosis and may induce unjustified confidence.

RQ4 – Which information sources provide the greatest contribution to accurate diagnoses?

This research question aims to identify the relative influence of different information sources, including:

- logs;
- metrics;
- traces;
- recent code changes;
- deployment records;
- technical documentation.

In a preliminary industrial investigation assisted by MCP, the combination of temporal metrics, spans/traces, user-agent information, and deployment records proved more decisive than isolated logs, which, in part of the analysis, did not provide usable evidence.

RQ5 – What risks remain when agents have read-only access to observability data?

Even without permission to directly modify systems, agents may incorrectly interpret available information or suggest inappropriate actions. This question investigates limitations related to confidence, excessive data collection, and incorrect decision-making.

A specific risk observed during MCP-assisted investigations is the recommendation of operational actions (for example, tightening rate limits or operation limits) based on incomplete aggregated signals, even when access remains strictly read-only.

5.2 Research Hypotheses

The hypotheses presented in this study represent propositions to be experimentally tested. None of them are considered validated before execution of the proposed protocol.

H1 – Structured access to operational context can reduce diagnosis time

The primary hypothesis is that agents connected to organized observability sources may identify correct root causes within shorter investigation periods due to reduced manual searching across fragmented information sources.

H2 – Availability of richer context can improve the quality of evidence used by agents

This hypothesis evaluates whether agents with structured access demonstrate greater ability to justify their decisions using relevant operational signals.

H3 – A larger amount of information does not automatically guarantee better patches

This hypothesis considers that excessive data availability or conflicting information may negatively affect decision-making, even when agents have access to additional sources.

In preliminary industrial investigations, the availability of additional metrics without correlatable execution samples was insufficient to safely support changes to rate limits or operation limits. This reinforces that telemetry volume does not automatically translate into decision-making quality.

H4 – Agent confidence may become inaccurately elevated when evidence is incomplete

An agent may generate a seemingly consistent explanation even when the available evidence is insufficient. This study aims to measure such behavior.

Preliminary observations suggest this risk occurs when aggregated maximum values (for example, query complexity peaks or rate-limit events) are treated as sufficient proof of abuse, despite the absence of retained traces, client identity information, or the GraphQL operation document.

6. EXPERIMENTAL METHODOLOGY

6.1 General Study Design

This work proposes a controlled experiment to compare different strategies for contextual access during software failure diagnosis and correction activities.

The study will be conducted using a reproducible environment based on reference applications with microservices architectures, enabling the creation of controlled failure scenarios and standardized evidence collection.

Each experimental execution will begin from a predefined configuration containing:

- infrastructure used;
- artificial intelligence model;
- instructions provided to the agent;
- computational resource limits;
- time constraints;
- available tools.

The standardization of these elements aims to reduce external interference and enable comparisons between different experimental conditions.

6.2 Experimental Environment

The test environment will be based on the **OpenTelemetry Demo**, a reference application composed of multiple instrumented services designed for the generation and collection of observability data.

The environment will enable:

- controlled workload generation;
- creation of known failures;
- metrics collection;
- log storage;
- distributed tracing;
- service behavior analysis.

The adoption of a standardized environment allows other researchers to reproduce the experiment in future studies.

6.3 Experimental Conditions

The study will be divided into three main experimental conditions.

Condition A – Investigation Using Static Information

In this condition, the agent will receive:

- source code;
- available documentation;
- previously collected exports of operational information.

The agent will not have dynamic access to observability tools.

The objective is to represent scenarios in which information is provided through traditional approaches.

Condition B – Manual Investigation Supported by APM

In this condition, the process will include access to traditional Application Performance Monitoring tools.

The investigator will be able to manually query:

- metrics;
- logs;
- traces;
- execution information.

After the initial analysis, the agent will be used to assist in generating the correction.

This condition represents a hybrid workflow combining human analysis and artificial intelligence assistance.

Condition C – MCP-Assisted Investigation

In this condition, the agent will have direct access to tools integrated through the Model Context Protocol.

Access will be restricted to read-only mode, allowing structured queries without the ability to modify the environment.

Available sources may include:

- distributed traces;
- logs;
- metrics;
- alerts;
- deployment information.

The objective is to evaluate whether a structured context architecture changes the agent's ability to investigate and correct failures.

6.4 Experimental Control

To ensure comparability between conditions, the following elements will remain constant:

- artificial intelligence model used by the agent;
- baseline instruction (prompt);
- analyzed repository;
- failure scenarios;
- time limit;
- token budget;
- evaluation criteria.

Differences exclusively related to the method of contextual access will be recorded as part of the experimental treatment.

6.5 Failure Catalog and Experimental Procedure

The experimental evaluation will use a set of predefined failure scenarios, enabling comparison of agent performance across different diagnosis and correction contexts.

The failure catalog will be designed to represent common problems found in modern distributed software environments, including failures related to code, configuration, dependencies, and service communication.

Each experimental scenario will contain a definition previously established by the researchers, including:

- expected system behavior;
- observed behavior after failure introduction;
- affected component;
- root cause;
- available evidence;
- expected correction;
- possible modifications that only eliminate symptoms.

Preparing these elements in advance will allow the evaluation not only of whether the agent generates a modification, but also whether the modification corresponds to the actual cause of the problem.

The scenarios may include examples such as:

- inadequate timeout configuration;
- incompatible retry policies;
- loss of context propagation between services;
- feature flag divergence;
- cache mechanism inconsistencies;
- incompatible data schema changes;
- improperly updated dependencies;
- problems caused by recent deployment changes.

In addition to these classical examples, the catalog will incorporate scenarios motivated by preliminary industrial observations in a GraphQL API gateway operating in a large-scale financial technology environment in the United States, involving rate limiting, operation limits, and query complexity. These scenarios were investigated using an MCP-assisted agent and a commercial APM platform.

The scenarios include:

- **Query complexity spike** identified through aggregated metrics, but without retained correlatable samples in the APM system. The oracle requires the agent to declare insufficient evidence rather than invent a client identity or GraphQL operation;
- **Synthetic traffic identification** (for example, unauthenticated curl requests) versus organic traffic, requiring correct classification before any recommendation involving rate limits or operation limits;

- **Requests rejected by operation limits (HTTP 400)** that do not consistently increase aggregated daily metrics, representing undercounting caused by incomplete telemetry;
- **False temporal correlation** between a rate limiting/operation limits deployment and a mechanically unrelated incident occurring during the same period, such as authentication or authorization failures;
- **Premature decision to tighten rate limits or operation limits** based on a low organic traffic baseline and outliers without reliable client identity information, classified as a configuration false fix;
- **Mitigation that shifts rather than resolves the symptom:** a rate limit is bypassed through request serialization (for example, reducing concurrency in the client or test runner), eliminating visible HTTP 429 Too Many Requests errors but exposing latent timeouts and causing cascading failures in dependent components. The oracle classifies the initial adjustment as an incomplete correction;
- **Divergence between configured rate limits/operation limits and actual traffic behavior**, in which observed metrics remain significantly below thresholds and the agent should recommend maintaining the current configuration rather than premature tightening.

These scenarios share a desirable characteristic for the experimental protocol: they contain a technically feasible, executable, and testable modification that nevertheless does not represent the actual root cause of the problem.

The evaluation oracle will be prepared before experimental execution. This oracle will serve as a reference to determine:

- whether the root cause was correctly identified;
- whether the produced patch resolves the actual problem;
- whether the modification introduces new risks;
- whether available tests are sufficient to validate the correction.

6.6 Experimental Execution Procedure

Each experiment will follow a standardized sequence.

Initially, the agent will receive an initial incident description containing information similar to that available in real operational environments, such as alerts or reports of unexpected behavior.

From this point onward, all actions performed during the investigation will be recorded.

The process will be divided into the following stages:

Stage 1 – Initial Problem Identification

The moment at which the agent receives the initial alert and begins the analysis will be recorded.

The following aspects will be observed:

- initial hypotheses formulated;
- information sources consulted;
- time until the first possible explanation is generated.

Stage 2 – Investigation and Evidence Collection

During this stage, the following elements will be recorded:

- analyzed files;
- consulted tools;
- executed queries;
- information used in the decision-making process.

The purpose is to evaluate not only the final outcome, but also the quality of the investigation path followed by the agent.

Stage 3 – Root Cause Identification

The moment when the agent presents a hypothesis considered correct will be recorded as the main investigation milestone.

This stage will enable measurement of the time required to reach an appropriate explanation.

Stage 4 – Patch Generation

After identifying the probable cause, the agent will be required to propose a system modification.

The following aspects will be evaluated:

- alignment with the identified problem;
- modification size;
- potential impact;
- clarity of the presented justification.

Stage 5 – Validation

The correction will be submitted to predefined validation mechanisms.

The evaluation will consider:

- execution of automated tests;
- patch acceptance;
- absence of regressions;
- compatibility with expected system behavior.

7. METRICS AND ANALYSIS PLAN

The experimental evaluation will be based on quantitative and qualitative metrics, enabling the analysis of different dimensions of agent performance.

The primary metric will be:

7.1 Time to Correct Root Cause Identification

This metric represents the interval between the beginning of the investigation and the correct identi-

fication of the problem origin.

This measure enables the evaluation of whether different forms of contextual access influence diagnostic speed.

7.2 Quality of Generated Patches

Patch quality will be evaluated considering:

- patch acceptance;
- test approval;
- need for additional modifications;
- ability to resolve the original cause.

A modification that removes a visible error but preserves the fundamental cause will be classified as an incomplete correction.

7.3 False Fix Rate

One of the central objectives of this study is identifying situations in which the agent produces an apparently valid solution that does not resolve the actual problem.

This indicator is important because agents may generate technically plausible responses without necessarily understanding the underlying cause of an incident.

7.4 Decision Provenance

The origin of the information used by the agent will be analyzed.

The research will observe:

- which tools were consulted;
- which evidence supported the hypothesis;
- which information was ignored;
- whether the decision was adequately justified.

Traceability is essential for evaluating reliability in intelligent systems.

7.5 Human Intervention

The amount and type of human intervention required throughout the process will be measured.

The evaluation will consider:

- clarification requests;
- redirection of the investigation;
- external validations;
- manual patch modifications.

7.6 Operational Cost

Factors associated with computational cost will also be evaluated, including:

- number of queries performed;
- token consumption;
- processing time;
- volume of accessed information.

The objective is to determine whether potential performance improvements justify the additional cost associated with agent utilization.

7.7 Evidence Completeness and Declared Inconclusiveness

Preliminary industrial observations indicated that a significant portion of investigations may conclude without sufficient evidence for causal attribution, even when structured telemetry access is available. Therefore, the proposed protocol introduces two additional metrics.

- **Evidence Completeness Index:** The proportion between the evidence elements required by the failure oracle (for example, operation identification, client identity, timestamp, and associated change) and the elements effectively retrieved by the agent during investigation.

- **Declared Inconclusiveness Rate:** The proportion of executions in which the agent explicitly recognizes insufficient available evidence instead of presenting an unsupported root cause.

Unlike other metrics, declared inconclusiveness is not interpreted as agent failure. In scenarios involving incomplete telemetry, recognizing the absence of sufficient evidence represents the correct behavior. Treating such behavior as an error would encourage precisely the inappropriate confidence described in hypothesis H4.

7.8 Statistical Analysis Plan

The experimental protocol will employ statistical analysis based on comparisons between experimental conditions.

The analysis will consider:

- experimental condition as a fixed effect;
- individual task as a random effect;
- repeated experiments to reduce the influence of occasional variations.

In addition to average values, result distributions will be presented, allowing the identification of extreme behaviors.

This approach is important because an average improvement may conceal critical situations, such as increased false fixes or severe failures in specific incident categories.

Positive, neutral, and negative results will all be considered relevant for scientific evaluation.

8. SECURITY MODEL FOR MCP USAGE

The integration between artificial intelligence agents and operational tools requires control mechanisms to prevent risks associated with inappropriate access to information.

In this study, tools connected through MCP will operate exclusively in read-only mode.

Each tool must have:

- a defined purpose;
- a known query schema;
- response limits;
- authorized services;
- protected sensitive fields;
- complete logging of performed calls.

The credentials used will be synthetic, temporary, and restricted to the experimental environment.

All queries must record:

- agent identification;
- request timestamp;
- tool utilized;
- relationship with the experimental task.

Inventory of Read-Only Tool Classes

Based on preliminary industrial observations, the protocol will adopt the minimum inventory of read-only tool classes described in Table 3. This classification is relevant because each class presents a distinct interpretative risk, not merely an access risk.

Table 3. Read-only MCP tool classes, investigative functions, and associated interpretative risks.

Tool Class	Investigation Function	Primary Risk
Capability discovery	Enumerate available tools and schemas before performing queries	Unnecessary expansion of the context surface
Aggregated metrics queries	Obtain time series by service and environment with controlled resampling	Aggregation may hide actual frequency and caller identity
Execution sample retrieval (spans)	Locate requests correlated with a specific time window	Partial sampling may generate absence of interpretable evidence
Full trace retrieval	Inspect attributes of a specific request	Exposure of sensitive fields contained in traces
Event and deployment queries	Correlate recent changes with symptom onset	Spurious temporal correlation between independent events
Visualization generation	Summarize time series for human inspection	Biased interpretation depending on selected scale and time window

The experimental record should store, for each call, the tool class, time window parameters, amount of returned data, and whether the result was effectively used in the final hypothesis.

This record enables the distinction between queries that generated relevant evidence and queries that merely consumed computational resources.

8.1 Risks and Limitations of MCP Usage in Software Engineering Environments

Although structured access to context may represent an evolution in the integration between intelligent agents and operational systems, its adoption also introduces challenges that must be considered.

One of the primary risks is related to the incorrect interpretation of available information. Access to larger amounts of data does not automatically guarantee better decisions. An

agent may receive multiple signals but assign inappropriate importance to specific evidence or overlook factors relevant to the actual cause of the problem.

Another risk involves excessive confidence in agent-generated responses. A technically plausible explanation may appear correct even when it is based on incomplete evidence.

Therefore, this study considers transparency of the decision-making process to be as important as the final outcome. It is insufficient to evaluate whether an agent reaches a solution; it is necessary to understand which information sources were used and how the conclusion was constructed.

Information security risks must also be considered. Even within a read-only model, access to operational data may reveal sensitive information regarding system architecture, behavioral patterns, or internal processes.

Therefore, the adoption of agents connected to external tools should include:

- strict permission control;
- environment isolation;
- information anonymization;
- query traceability;
- human validation for critical decisions.

This research does not assume that MCP eliminates these challenges. Instead, it aims to evaluate how a controlled architecture can mitigate some of the problems associated with fragmented context.

9. VALIDITY, ETHICS, AND REPRODUCIBILITY

9.1 Experimental Validity

As with any experimental study, this research presents limitations related to the ability to fully represent real-world production environments.

A controlled environment enables greater precision in measuring results; however, it does not completely reproduce factors commonly found in organizations, such as:

- operational pressure;
- informal team knowledge;
- legacy systems;
- external dependencies;
- undocumented historical decisions.

To reduce these limitations, the study will employ multiple failure scenarios, different problem categories, and controlled variations in contextual availability.

Furthermore, results should be interpreted considering that an experimental benchmark represents an approximation of reality rather than a complete replacement for experiences conducted in production environments.

9.2 Ethical Considerations

The research will be conducted exclusively using controlled environments and synthetic data.

When preliminary industrial observations are mentioned, they will be presented only in aggregated and anonymized form, without constituting part of the experimental dataset.

The following will not be used:

- private corporate data;
- real credentials;
- user information;
- proprietary source code;
- confidential incident records.

Industrial observations, when referenced, are presented in aggregated and anonymized form, without identification of the organization, users, credentials, or proprietary artifacts.

The controlled experiment remains restricted to synthetic environments.

The objective is to evaluate methodologies and architectures, not to analyze specific organizations or professionals.

If human participation is included in future stages of the experiment, participants must receive clear information regarding the research objectives, procedures, and treatment of collected data.

9.3 Reproducibility and Open Science

One of the central principles of this study is enabling other researchers to reproduce and evaluate its results.

For this purpose, the final version of the research intends to make the following artifacts available:

- experimental infrastructure;
- environment configuration;
- failure catalog;
- MCP tool manifests;
- agent instructions;
- sanitized experimental records;
- statistical analysis scripts.

The publication of these artifacts will allow different artificial intelligence models, observability tools, and development environments to be evaluated using a common methodology.

This approach contributes to a more transparent and comparable scientific process.

10. EXPECTED RESULTS

This section will only be completed after the full execution of the experiment.

No performance, accuracy, or quality results are assumed at this stage.

The industrial observations described in Section 3.1 do not replace the results of the controlled experiment; they serve exclusively as empirical motivation and as a basis for designing failure scenarios.

The study aims to produce evidence regarding the potential effects of structured telemetry access on the process of AI-assisted failure resolution.

Future analyses should consider different possible outcomes:

- if agents with MCP demonstrate improved performance, it will be necessary to determine in which scenarios such improvements occur and what associated costs are involved;
- if results are equivalent across conditions, this may indicate that structured context access does not provide significant advantages in certain scenarios;
- if negative results are observed, these findings will also be relevant for understanding limitations and risks associated with the approach.

The research also intends to analyze situations in which speed and quality exhibit different behaviors.

An agent may reach a response more quickly but generate a higher number of false fixes. Similarly, it may provide higher diagnostic quality while requiring greater operational cost.

Therefore, this study considers efficiency as a multidimensional concept.

11. EXPECTED SCIENTIFIC CONTRIBUTION

The main expected contribution of this work is the creation of an experimental protocol capable of evaluating artificial intelligence agents in complete software diagnosis and correction activities.

Unlike evaluations focused exclusively on code generation, this research proposes analyzing the entire lifecycle:

incident signal → investigation → hypothesis → root cause → patch → validation

This approach enables the observation not only of code generation capabilities but also of the operational reasoning capabilities of intelligent agents.

Furthermore, the study contributes to the discussion regarding **context engineering applied to artificial intelligence**.

The research is based on the premise that intelligent agents do not depend solely on increasingly advanced models, but also on the quality of the information architecture made available to them.

Therefore, the expected scientific contribution is related to the creation of methods capable of measuring how context, observability, and external tools influence intelligent systems.

12. CONCLUSION

The evolution of artificial intelligence agents introduces new possibilities for software engineering, particularly in complex activities such as failure diagnosis and correction.

However, the ability to generate code or produce responses is not sufficient to guarantee reliable solutions. Real-world systems require contextual understanding, evidence-based analysis, and rigorous validation.

This paper presented an experimental protocol designed to investigate whether structured, read-only access to telemetry through the **Model Context Protocol** can assist intelligent agents in software bug resolution.

The research does not assume prior results but proposes a methodology capable of measuring the benefits, limitations, and risks associated with this approach.

Preliminary industrial observations, presented in anonymized form, provide empirical motivation for evaluating not only diagnostic speed but also telemetry completeness, evidence provenance, and the risk of premature decisions based on incomplete signals.

By connecting observability, artificial intelligence agents, and open science practices, this study seeks to contribute to a more precise understanding of the role of context architecture in the future of software engineering.

The central question remains open: it is not sufficient to determine whether agents can generate corrections. It is necessary to understand under which conditions they can produce corrections that are reliable, verifiable, and sustainable.

REFERENCES

Beyer, B.; Jones, C.; Petoff, J.; Murphy, N. R. *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media, 2016.

Beyer, B.; Murphy, N. R.; Rensin, D. K.; Kawahara, K.; Thorne, S. *The Site Reliability Workbook: Practical Ways to Implement SRE.* O'Reilly Media, 2018.

Google Cloud. *DORA Research: Accelerate State of DevOps Report.* Google Cloud, multiple years.

OpenTelemetry Authors. *OpenTelemetry Documentation and Specification.* Cloud Native Computing Foundation.

Zhang, C.; Yang, J.; et al. *Large Language Models for Software Engineering: A Systematic Literature Review.* ACM/IEEE Software Engineering Research Literature, 2023.

Chen, M.; Tworek, J.; Jun, H.; et al. *Evaluating Large Language Models Trained on Code.* arXiv, 2021.

Pearl, J. *Causality: Models, Reasoning, and Inference.* Cambridge University Press, 2009.

Sculley, D.; et al. *Hidden Technical Debt in Machine Learning Systems.* Advances in Neural Information Processing Systems, 2015.

Author's biography

Felipe O. B. Souza is an independent researcher in the fields of Software Engineering, Artificial Intelligence, and Distributed Systems. His research interests include intelligent agents, observability, software development process automation, failure diagnosis, and the application of artificial intelligence models in complex computing environments.