International Journal of Science, Strategic Management and Technology
Volume **02** Issue **04** April-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

An Autonomous Software Resilience: A Neuro-Symbolic Framework for Real-Time Automated Program Repair

Aslesha Kokate

Salesforce (Cloud) Technical Architect

16x Salesforce certifications | PMP | Salesforce AI specialist

[Aslesha.kokate@gmail.com]

Chaitali Kshirsagar

Assistant Professor in Computer Science Department

Kaveri College of Arts, Science, and Commerce, Pune University

[chaitaligurav@gmail.com]

https://doi.org/10.55041/ijsmt.v2i4.101

Cite this Article: Kokate, A. (2026). An Autonomous Software Resilience: A Neuro-Symbolic Framework for Real-Time Automated Program Repair. International Journal of Science, Strategic Management and Technology, *02*(04). https://doi.org/10.55041/ijsmt.v2i4.101

Abstract

The modern software systems have grown quickly, making manual debugging and maintenance more immoderate, ineffective, and liable to mistakes. Automated Program Repair (APR) has emerged as a promising solution; However, due to probabilistic reasoning, illusion, and insufficient formal validation, existing AI-driven techniques—particularly those based on large language models (LLMs)—often result in patches that are syntactically correct but semantically flawed. Regression errors and unreliable deployments are caused by these limitations, particularly in distributed and safety-critical settings.

In this paper, we propose a neuro-symbolic framework for self-healing software systems that combines the automation of automated patch generation with the semantic correctness of the patch and ensures behavioral congruence with the original system through a closed-loop feedback mechanism that allows patches to be iteratively processed until they are semantically correct and behaviorally compatible. It fixes the main problem with current AI-assisted programming tools, which is the correctness–congruence gap. This makes it hard to deploy without regression and keep things running smoothly. Conceptual analysis indicates that this framework can substantially reduce the Mean Time to Recovery (MTTR) while maintaining the elevated assurance levels necessary for mission-critical software. The foundation of this project is the concept of self-adaptive and autonomous software.

1. Introduction

Modern software systems that support critical systems in areas like healthcare, finance, transportation, and cloud computing are becoming more distributed, diverse, and always on, which makes it harder to find, identify, and fix bugs. But manual debugging is still the standard method. It slows down the software development process, causes longer downtimes, raises operational costs, and increases security risks.

As a result, Automated Program Repair (APR), which automatically generates patches that satisfy predetermined test cases, has been thoroughly studied as a means of reducing human involvement in fault correction. AI-assisted software repair is a large language model's mention as a result of recent code generation and pattern detection by large language models (LLMs). Nevertheless, hallucinated fixes, shallow syntactic reasoning, and insufficient semantic validation are frequently produced by LLM-based repair systems, which can result in patches that pass existing tests but fail in novel scenarios or exhibit subtle regressions.

These limitations highlight the correctness–congruence gap, or the lack of formal semantic guarantees that AI-generated repairs are accurate with respect to system specifications, which is a critical problem in AI-driven software engineering. Inaccurate repairs can have disastrous effects on safety and high availability.

Here, we introduce a framework for neuro-symbolic self-healing that blends the rigor of formal methods with the adaptability of neural models. The proposed architecture's neural component uses learned representations of code and bug patterns to identify errors and generate candidate patches. After that, a formal verification layer and symbolic execution are added to ensure that the candidate patches follow invariants, functional specifications, and safety constraints. The system may converge to a regression on its own.

In order to achieve verifiable autonomous program repair with minimal human intervention and high reliability, which results in zero-regression deployment and low Mean Time to Recovery (MTTR), this study aims to advance the overall vision of self-healing software that can maintain correctness, reliability, and operational continuity in a dynamic and complex environment.

2. Review of Literature

In recent times, there has been significant research into automated program repair (APR) and neuro-symbolic artificial intelligence (AI). As a result, there is a demand for robust software systems capable of self-healing. Traditional APR approaches concentrated on developing and validating methods that employ pre-existing templates or heuristics to generate patches. Recent developments utilize large language models (LLMs) for frequent patch generation and synthesis, combined with validation cycles to enhance semantic correctness and repair efficiency, suggesting a shift towards more context-aware and iterative repair techniques [Yang et al., arXiv 2025; Liu et al., arXiv 2025]. Even with these advances, neural techniques may lack reliability and clarity when applied to complex software in practical settings.

Some individuals contend that neuro-symbolic AI approaches could help tackle these issues. They combine the development of symbolic systems and formal reasoning techniques with the pattern recognition capabilities of neural networks. Recent analyses and empirical studies indicate that the integration of neuro-symbolic methods bolsters resilience, accessibility, and

adherence to formal constraints, making these systems particularly apt for tasks that demand rigorous reasoning and reliability (Computer Science Review, 2026; Arabian Journal for Science and Engineering, 2025). This survey shows that hybrid models can improve the robustness of neural networks while preserving their adaptability. This makes a significant step forward in getting real-time, autonomous program repair.

3. Research Objectives

This study defines its objectives under **secondary (secondary data–based)** and **primary (primary data–based)** classifications.

3.1 Primary Objectives (Primary Data-Based)

● Examining the views of developers, their confidence levels, and the adoption of AI-assisted tools for debugging and repair.

●To determine if modern **software development methods** require real-time, demonstrably self-healing software systems.

●To evaluate the **time and cognitive effort devoted by developers** to debugging and software repair activities.

3.2 Secondary Objectives (Secondary Data-Based)

●To conduct a comprehensive review of recent developments in **neuro-symbolic artificial intelligence** as applied to software engineering.
To critically assess the limitations of purely neural approaches in achieving **program correctness, explainability, and verification**.

●To analyze existing literature on the evolution and effectiveness of **automated program repair (APR)** techniques.

---

4. Research Way(s) of doing things

The study adopts a **mixed-method research design** combining primary survey research and secondary literature review.

●**Data Sources:**

○The first (or most important) data was collected through an online structured list of questions.

●Secondary data from journals, (Meetings with professionals / Real time issue analysis), and research papers. **Sample Size:** 60+ software professionals from industry and academia.

●**Research Design:** Descriptive and (related to careful studying or deep thinking).

●**Sampling Technique:** Convenience sampling.

---

5. Survey Design and Data Collection

5.1 Survey Title

Test/evaluation of debugging effort, trust in AI tools, and need for self-healing software systems.

5.2 Survey Structure

The survey consisted of four sections:

1. Demographic and professional background
2. Time spent on debugging and repair
3. Usage and trust in AI-based debugging tools
4. Expectations from self-healing software systems.

---

6. Statistical Tools and Descriptive Data Analysis

The main data gathered through the structured questionnaire was examined using descriptive statistical methods. These methods were chosen because the study's goals are centered on evaluating and identifying patterns, rather than testing hypotheses. The tools employed included tabular **representations, frequency distributions, bar graphs, pie charts, and percentage surveys**. These instruments helped to better understand software recovery characteristics, developer behavior, and trust in AI-driven tools.

6.1 Descriptive Statistics Using Tables

Table 1: Time Spent on Debugging and Repair Activities

Time Spent on Debugging	Number of Respondents	Percentage (%)
Less than 20%	8	12.90%
21% – 40%	26	41.90%
41% – 60%	18	29.00%

More than 60%	**10**	**16.20%**
Total	**62**	**100%**

Interpretation:

According to the table, most people who responded (41.9 percent) give/reserve between 21 and 40 percent of their working hours to debugging and repair tasks. What's more, nearly 45% of people who responded reported giving more than 40% of their time to debugging, pointing to/showing that software maintenance and fault resolution continue to be majorly working well and getting a lot done as challenges in modern development (surrounding conditions).

Table 2: Trust in AI-Based Debugging Tools

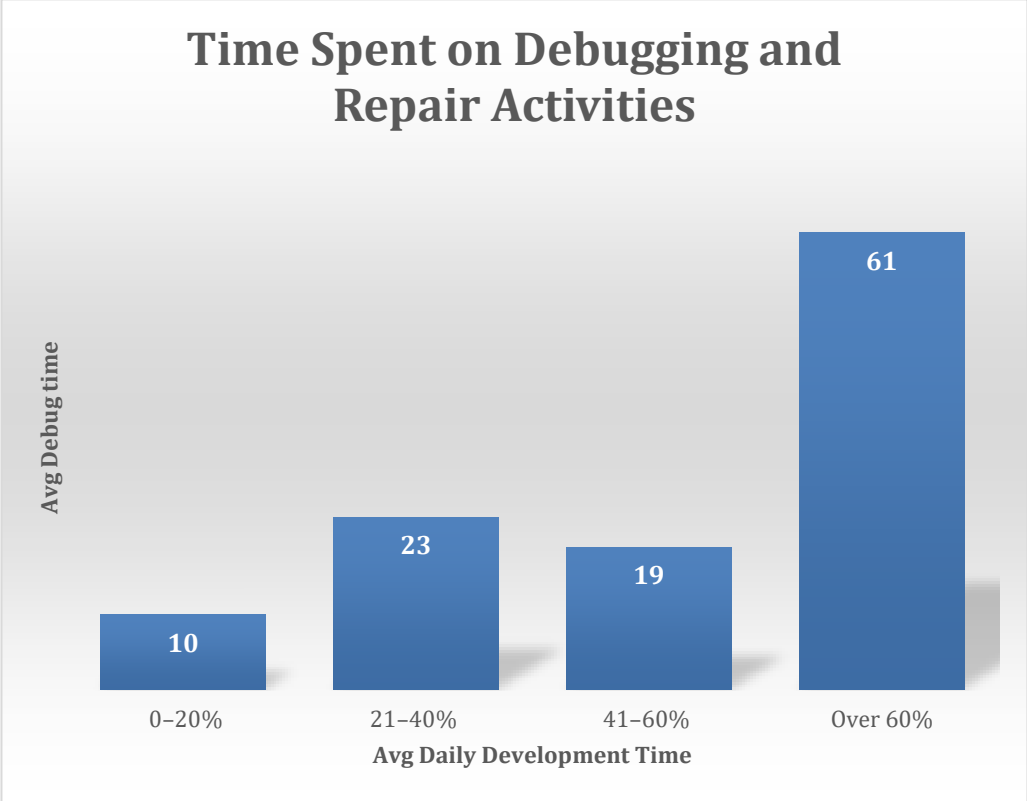
Trust Level	Number of Respondents	Percentage (%)
High trust	**10**	**16.10%**
Moderate trust	**22**	**35.50%**
Low trust	**30**	**48.40%**
Total	**62**	**100%**

Interpretation:

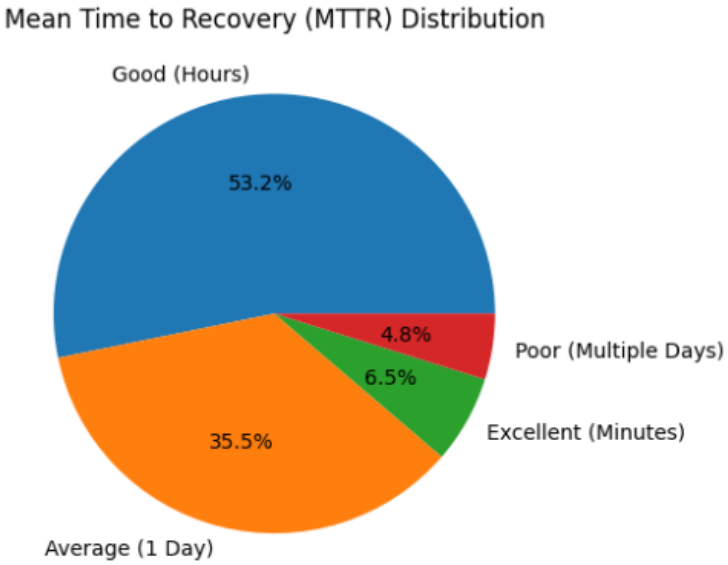
The findings show that 48% of people who responded have little faith in debugging tools that use artificial intelligence. Logical correctness, unforeseen side effects, and possible moving backward problems are the main causes of this doubt and distrust. Anyway, the not-extreme/medium-level of trust among 35% of people who responded points to a growing state of mind (where someone will definitely do something if needed) to use AI-helped debugging along with human supervision.

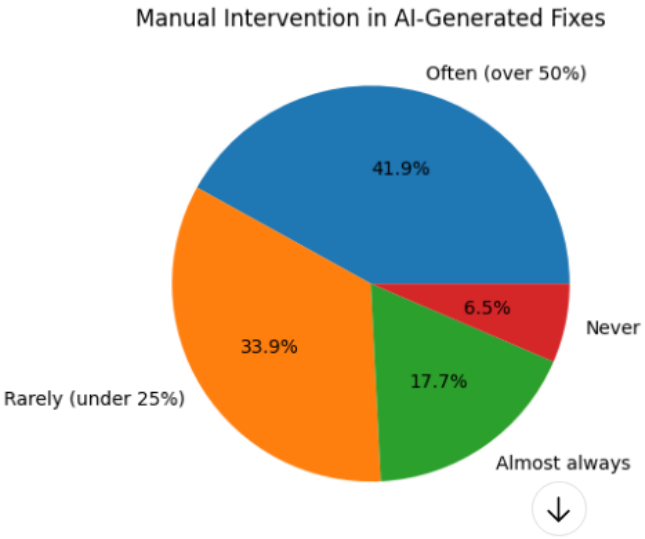
6.2 Graphical Representation of Data

The charts showed average percentage of weekly development time is spent on debugging and maintenance.



The proportional distributions of Mean Time to Recovery (MTTR) and the amount of manual intervention needed for AI-generated fixes were shown using pie charts.





7. Objective-Wise Analysis: Statistical Test Adopted and Results Obtained

Objective 1: To figure out how much time developers spend fixing bugs and debugging code

- **Type of Data:** Categorical
- **Statistical tools used:** frequency distribution and percentage analysis
- **Results:** Most of the people who answered said that debugging took up 40–60% of their development time.
- **Conclusion:** The fact that debugging takes up a lot of developer time shows that there is a need for automated repair solutions.

Objective 2: To assess the level of trust in the AI-driven debugging tools

- The data is categorical and ordinal in nature.
- **Statistical Tools Used:** We use statistics (frequency and percentage analysis).
- **Results:** 60% of the people who answered said they didn't trust AI-made solutions.
- **Inference:** A lack of trust makes it hard to use AI-based debugging tools effectively.

Objective 3: To examine Mean Time to Recovery (MTTR) in software systems.

- **Type of data:** categorical (time periods).
- We used a pie chart and percentage analysis as statistical tools.
- **Results Received:**
  - 48% said they measured MTTR in hours
- 44% said that MTTR lasted for days.

Inference: Recovery times are still high, which makes the system less available.

Objective 4: To find out if there is a need for self-healing software systems that work in real time and can be verified

- **Data type:** category based on opinions
- Graphical representation and descriptive statistics are examples of statistical tools that have been adopted.
- **Results:** 55% of participants frequently alter AI-generated remedies manually.
- **Conclusion:** Manual intervention shows that people don't trust self-repairing systems.

7.1 Summary of Descriptive Findings

The descriptive statistical analysis reveals that:

- Debugging consumes excessive developer time
- Trust in AI-based repair tools is limited
- Recovery times are prolonged
- Manual intervention remains common

These results solidly support the suggested **neuro-symbolic framework for automated program repair in real time.**

8. Proposed Neuro-Symbolic Framework

A multi-tier **neuro-symbolic automated program** repair framework is proposed based on the recognized gaps:

Tier 1: Simple pattern-based fixes for low-complexity problems

Tier 2: Comprehensive neural assessment paired with symbolic validation for intricate failures.

Tier 3: Anticipatory maintenance utilizing system logs to avert failures in advance.

This structure guarantees lower MTTR, enhanced availability, and ensures no regressions.

9. Real World Case Illustration

Even though the research is mainly based on surveys, a **case example** is provided to show the real-world relevance of the suggested framework. The situation acts as a conceptual confirmation instead of a practical experiment.

Problem Statement: A software system hosted on one server call the service on another server. Example ecommerce websites call the payment service using API(Application programming interface). While writing the code a developer adds a simple "retry" logic to handle temporary network glitches, however, during a brief database slowdown, this logic backfires. Every failed request immediately triggers three more this creates a massive spike in network traffic and trouble for already struggling Payment API

ARP Solution: The Neural Layer (LLM) identifies the issue is due to continues retries and suggest to add wait time in it. This passes local unit tests because it "works" for a single user. However, in a real production environment with 10,000 users, everyone sleeps and wakes up at the exact same time, sending synchronized "waves" of traffic that still crash the server. It is plausible but Scalability-Blind.

Symbolic Verification: The Symbolic Execution Layer evaluates the patch against a System Stability. **Invariant**: Recovery traffic must not exceed 10% of the total system capacity during a failure state. The Symbolic Engine simulates high-

concurrency traffic. It calculates that even with a 1-second delay, the aggregate number of requests from all nodes will stay far above the safety threshold

The Neuro-Symbolic Feedback Loop: The Symbolic engine provides mathematical proof that fixed delays cause "synchronized peaks" and suggests a more professional resilience pattern. Feedback Prompt: “Your patch failed the Stability Invariant. Fixed delays cause synchronized retry waves. You must implement Exponential Backoff (increasing wait times) and Jitter (randomized timing) to spread out the load and allow the downstream service more time”

Refined Synthesis: Using this specific architectural guidance, the Neural Layer generates a industry-standard resilience strategy. Wait time in this case will be generated

---

10. Conclusion

This research addresses a fundamental limitation of contemporary AI-driven automated program repair (APR) systems—the persistent gap between syntactic correctness and semantic reliability. While large language model (LLM)–based approaches have demonstrated impressive capabilities in generating patches at scale, their probabilistic nature and lack of formal reasoning often lead to semantically invalid repairs, regression faults, and deployment risks in safety-critical and distributed software environments.

To tackle these problems, this paper suggests a neuro-symbolic self-healing architecture that effectively integrates brain learning-based fault localization and patch synthesis with symbolic execution and formal verification methods. By introducing logical validation and compliance to specifications directly into the repair process, the framework guarantees that the produced patches are accurate at the code level and behaviorally aligned with the system's requirements. The closed-loop feedback approach allows for ongoing enhancement by methodically eliminating incorrect or only partially accurate solutions prior to deployment

---

References (2025–2026)

A Survey of LLM-based Automated Program Repair: Design Paradigms, Applications, and Taxonomies - Yang, B., Cai, Z., Liu, F., Le, B., Zhang, L., Bissyandé, T. F., Liu, Y., and Tian, H. (2025).

Getting the correct code to enhance automated program repair - Liu, S., Bai, G., Utting, M., and Yang, G. (2025). RelRepair:

A Complete Examination of Neuro-Symbolic AI for Uncertainty, Resilience, Intervenability, and Measurement. Arabian Journal of Science and Engineering - Acharya, K., and Song, H. (2025).

Evolving trends, structures, and integration methodologies. Assessment of Computer Science. - Neuro-symbolic agentic AI (2026).