International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

Generative AI-Assisted Root Cause Analysis and Intelligent Resolution Recommendation for Software Bugs

Sneha H S and Bhargava Krishna M.R

3rd Year BCA, BMS College of Commerce and Management, Bengaluru, India

Corresponding Author Email: snehahsshivakumar@gmail.com

Co-author Email: bhargavkrish037@gmail.com

Abstract - Software bugs are inevitable in modern software development, and identifying their root causes is often a manual, time-consuming, and experience-dependent activity. Traditional bug tracking and debugging practices rely heavily on developer expertise, which leads to delayed resolution, recurring defects, and higher maintenance costs. This paper presents a Generative AI-assisted approach for automated Root Cause Analysis (RCA) and intelligent resolution recommendation for software bugs. The proposed system accepts bug information such as descriptions, error messages, stack traces, severity levels, and related code snippets, and uses Generative AI to interpret the failure context. Based on this analysis, it identifies probable root causes, categorizes them, and generates clear, step-by-step resolution recommendations. The approach is integrated with a bug lifecycle and release management workflow, allowing defects to be tracked from reporting to closure while retaining RCA insights for future reference. By transforming unstructured bug reports into structured and actionable knowledge, the approach reduces manual debugging effort, shortens resolution time, and supports consistent decision-making across development teams. The study demonstrates the potential of Generative AI to improve software quality, maintainability, and release reliability. The solution can benefit students, developers, testers, and software organizations seeking an accessible and efficient method for understanding and resolving software defects.

Keywords: Generative AI, Root Cause Analysis, Software Bugs, Bug Lifecycle Management, Resolution Recommendation

1. INTRODUCTION

Every non-trivial software system contains defects, and a large share of maintenance effort goes into understanding why a defect occurred. Answering that question requires reading error messages, tracing execution, reproducing the failure, and drawing on experience with similar failures. This process is known as Root Cause Analysis (RCA).

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

In most small teams, and especially in student projects, RCA is informal and its reasoning is seldom written down. Once the bug is closed, the knowledge of why it occurred stays with one person, and the same category of defect reappears later.

Recent advances in Generative AI, and in Large Language Models (LLMs) in particular, offer a practical way to reduce this burden. LLMs are trained on very large corpora of source code, documentation, and technical discussion. They can read an error message and a stack trace, relate them to common failure patterns, and express a likely explanation and a repair strategy in natural language. When such a model is placed inside a bug-tracking workflow, and when its output is stored in a structured and reviewable form, it can act as a knowledgeable first reviewer for every incoming defect.

This paper presents a Generative AI-assisted approach to RCA and resolution recommendation, together with a working web-based bug lifecycle and release management platform in which the approach is embedded. The principal contributions are as follows.

- A structured pipeline that converts an unstructured bug report (description, reproduction steps, error message, stack trace, severity, code snippet) into a structured RCA record containing a root-cause category, a probable cause, a step-by-step resolution, and a preventive action.
- An integration design in which the generated analysis is attached to a bug lifecycle (New to Closed), reviewed by a human, and then promoted into a searchable knowledge base so that insight is retained across releases.
- A prototype implementation of the lifecycle, RCA, knowledge base, release, deployment-verification, and analytics modules using Flask, a relational database, Bootstrap, and Chart.js, with role-based access control.
- A human-in-the-loop and guardrail design, together with an evaluation methodology, intended to make AI-generated diagnoses trustworthy enough for everyday use by students and small teams.

2. BACKGROUND AND RELATED WORK

2.1 Bug reports and their quality

A bug report is the primary input to debugging. Zimmermann et al. showed, through a survey of developers and users, that the most helpful elements of a report are clear reproduction steps, stack traces, and observed versus expected behaviour, while the most common problems are missing or inaccurate information [1]. This observation motivates our design choice to collect these fields explicitly in the reporting form and to feed them, in a consistent layout, to the language model.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

2.2 Automated triage and classification

Anvik et al. recommended developers for new reports using text classification [2], Runeson et al. detected duplicate reports with natural language processing [3], and Lamkanfi et al. predicted bug severity from text [4]. These approaches route and prioritise defects, but do not explain why a defect occurred or how to repair it.

2.3 Standard classification of anomalies

IEEE Std 1044 defines a standard classification of software anomalies, covering the recognition, investigation, action, and disposition of a defect [5]. Our RCA categories (requirement gap, design flaw, coding error, configuration issue, environment issue, third-party dependency, data issue, and testing gap) follow the same intent: to ensure that each defect is analysed along a common taxonomy so that trends can be counted and compared across projects and releases.

2.4 Large language models for code and incident analysis

Transformer-based language models [6] have been scaled into general-purpose systems that perform well on few-shot tasks [7], and models trained on source code can read and generate programs [8]. Prompting strategies that elicit intermediate reasoning steps have been shown to improve performance on multi-step problems [9]. Closest to the present work, Ahmed et al. studied the use of large language models to recommend root causes and mitigation steps for cloud incidents and reported that fine-tuned models produced useful recommendations when evaluated against incident data from a large service provider [10]. Retrieval-augmented generation, which supplies the model with relevant external documents at inference time, reduces unsupported statements and allows the model to use organisation-specific knowledge [11].

2.5 Gap addressed by this work

Existing studies concentrate on a single task such as triage, duplicate detection, or incident mitigation, while practical bug trackers store defects but do not help explain them. There is therefore a gap for an accessible tool that (i) interprets the failure context of an individual bug with Generative AI, (ii) expresses the result in a structured and consistent form, (iii) ties that result to the full lifecycle from report to release, and (iv) keeps the knowledge for reuse. This paper addresses that gap with a design aimed at students, small development teams, and academic projects.

3. PROBLEM STATEMENT AND OBJECTIVES

3.1 Problem statement

Root cause analysis is predominantly manual and experience-dependent. Conventional bug trackers record who reported a bug and what its status is, but not why it happened, so resolution is delayed, similar defects recur, and maintenance cost increases.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

3.2 Objectives

- To design a mechanism that accepts bug descriptions, error messages, stack traces, severity levels, and code snippets and interprets the failure context using Generative AI.
- To identify probable root causes and assign each to a defined category.
- To generate clear, step-by-step resolution recommendations and preventive actions.
- To integrate the analysis with bug lifecycle tracking and release management so that defects can be followed from reporting to closure and deployment.
- To retain validated RCA insights in a searchable knowledge base for future reference.
- To keep a human reviewer in control of every recommendation that enters the permanent record.

4. PROPOSED METHODOLOGY AND SYSTEM ARCHITECTURE

The proposed approach treats RCA as a pipeline with five stages: capture, contextualise, analyse, review, and retain. Figure 1 shows the architecture.

4.1 Capture

A tester reports a bug through a form collecting title, description, reproduction steps, severity, priority, and an optional attachment; error messages and stack traces are pasted into the description or attached. Structured capture matters because the quality of the analysis depends on the quality of the input [1].

4.2 Contextualise

Before calling the model, a context builder assembles a compact bundle: the report fields, the project name and technology stack, the relevant code snippet if one has been supplied, and, where available, the top few matching articles retrieved from the knowledge base. Sensitive values such as passwords, keys, and personal data are masked before the bundle leaves the application.

4.3 Analyse

The bundle is inserted into a prompt template that instructs the model to act as a software debugging assistant, to reason about the failure step by step, and to return its answer in a fixed JSON schema. The schema contains the root-cause category, the probable root cause, a confidence statement, the evidence in the report that supports the diagnosis, ordered resolution steps, a preventive action, and a list of verification checks. The model is explicitly told to say when information is insufficient rather than to guess.

4.4 Review

The parsed result is presented to the assigned developer as a suggestion. The developer may accept it, edit it, or reject it. Only an accepted or edited analysis is written to the permanent RCA record, and the record stores who approved it. This keeps responsibility with a person and prevents a confident but incorrect model output from silently becoming organisational knowledge.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

4.5 Retain

When a bug is fixed and tested, the validated RCA can be published as a knowledge base article linked to the originating bug. Later reports retrieve these articles as context, so each resolved defect makes future analyses more specific to the project.

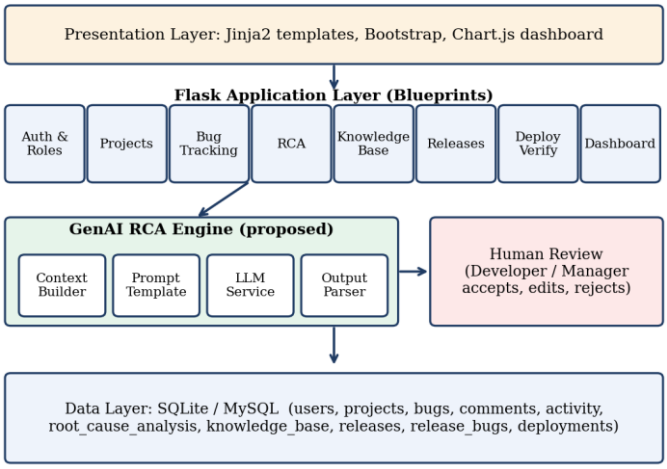


Figure 1. Layered architecture of the proposed system

5. IMPLEMENTATION OF THE LIFECYCLE PLATFORM

A prototype of the platform, named the Smart Bug Lifecycle, Root Cause Analysis and Release Management System, was implemented as a server-rendered web application. This section describes the technology choices and the modules implemented.

5.1 Technology stack

The application is written in Python using Flask [12], with each functional area as a blueprint. Pages use Jinja2 templates and Bootstrap, and the dashboard uses Chart.js. SQLite is the default database so the system runs on a student laptop, and an equivalent MySQL schema is provided. Passwords are stored as salted hashes.

5.2 Data model

The schema has eleven related tables: users, projects, project_members, bugs, bug_comments, bug_activity, root_cause_analysis, knowledge_base, releases, release_bugs, and deployments. A bug stores its title, description, reproduction steps, severity, priority, status, reporter, assignee, and attachment, while an RCA record stores the bug reference, category, root cause, preventive action, and technical recommendation.

5.3 User management and access control

Four roles are supported: Administrators manage users and projects, Managers create projects and plan releases, Developers are assigned bugs and record RCA, and Testers report and verify bugs. A login-required decorator protects every route, and role checks restrict sensitive actions.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

5.4 Bug reporting, assignment, and tracking

The bug module implements the lifecycle New, Assigned, In Progress, Fixed, Testing, and Closed. A bug may be assigned to a developer, and each status change, assignment, and comment is written to the activity log through a common logging function. This produces a complete history that can later be audited or used to compute resolution times.

5.5 Root cause analysis module

From any bug page an authorised user can record an RCA with a category from a controlled list of nine values (Table 1), a root cause, a preventive action, and a technical recommendation. A fixed vocabulary lets the dashboard and the language model describe causes in the same terms.

Table 1. RCA categories and example causes

Category	Typical example
Requirement Gap	Behaviour not specified for an edge case such as an empty search
Design Flaw	Missing validation layer between the interface and the data store
Coding Error	Off-by-one error, unhandled null value, wrong comparison operator
Configuration Issue	Incorrect environment variable or database connection string
Environment Issue	Different library version in production than in development
Third-Party Dependency	Breaking change in an external package or service
Data Issue	Malformed or legacy records that violate assumptions in the code
Testing Gap	Defect passed because no test covered the failing path
Other	Cause that does not fit the above

5.6 Knowledge base

The knowledge base stores articles with resolution steps, optionally linked to the originating bug, and supports keyword search. It is also the retrieval source for the contextualise stage (Section 4.2).

5.7 Release and patch management

Releases carry a version label and a status (Planned, In Progress, Released, or Rolled Back). Fixed bugs are attached to a release, giving traceability from a user-visible failure to the version that corrected it.

5.8 Deployment verification

Each staging or production deployment is recorded and issued a unique verification link with a QR code, which a field auditor can open without logging in to confirm the deployment.

5.9 Dashboard and analytics

The dashboard presents bug counts by status and severity, top resolvers, and a fourteen-day report trend. Because RCA categories use a controlled vocabulary, a category-wise chart shows where most defects originate and where preventive effort should go.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

6. THE GENERATIVE AI RCA AND RECOMMENDATION ENGINE

The engine is the component that turns the platform from a record-keeping tool into an assistant. It is designed as a stateless service with a narrow interface: it receives a bug identifier, builds the context, calls a language model, validates the answer, and returns a structured suggestion to the RCA screen. Because the engine is isolated behind this interface, the model provider can be replaced or upgraded without touching the lifecycle modules.

6.1 Input representation

The inputs are the bug description and reproduction steps (symptom and trigger), the error message and stack trace (failure type and location), severity and priority (urgency), an optional code snippet (suspect logic), and similar knowledge base articles (project-specific precedent).

6.2 Prompt design

The prompt has four parts: a role statement (a careful senior debugging assistant), the task description, the context bundle, and the output schema. The model must reason from evidence in the report to the most probable cause, choose a category from the controlled list, give numbered resolution steps, cite the supporting line of the error or trace, and state when information is insufficient. Stepwise reasoning is consistent with evidence that it improves multi-step inference [9].

6.3 Structured output

The engine requests a single JSON object with the fields category, root_cause, evidence, confidence, resolution_steps, preventive_action, and verification_checks. A parser validates the category and required fields, retries once on failure, and otherwise falls back to a blank manual form. This structure lets the result be stored in the root_cause_analysis table and counted in the dashboard.

6.4 Retrieval of prior knowledge

Following the retrieval-augmented generation principle [11], the context builder searches the knowledge base for articles matching the exception and module names of the new bug and adds the best matches to the prompt. This anchors the model on fixes that have worked in the same code base.

6.5 Guardrails and human oversight

Language models can produce fluent but incorrect explanations. The design therefore includes several controls.

- **Suggestion, not decision.** Output is labelled as AI-generated and is stored only after a developer accepts or edits it.
- **Evidence requirement.** A diagnosis that does not point to supporting text in the report is flagged as low confidence.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

- **Data protection.** Credentials, tokens, and personal data are masked before any text is sent to an external service, and teams with strict confidentiality needs can point the engine to a locally hosted model.
- **Safe fixes.** Resolution steps are advisory text. The engine never edits source code or runs commands.
- **Auditability.** The prompt version, model name, and reviewer are stored with every accepted analysis, and the activity log records the action.

7. ILLUSTRATIVE SCENARIO

To show how the pieces fit together, consider the following scenario. It is constructed for illustration and is not an experimental result.

A tester working on a book search application reports a bug titled "Search page crashes when the query is empty". The report describes the steps (open the search page, press Search with no text), marks the severity as High, and pastes an error message indicating that a string method was called on a null value, along with the three stack frames from the application code.

On the RCA screen the developer requests an AI suggestion. The engine classifies the cause as a Coding Error with a secondary Requirement Gap, explains that the handler passes the raw query to a string function, cites the relevant trace line, and recommends validating input at the start of the handler, handling empty queries gracefully, adding a unit test, and agreeing the intended behaviour in the requirements. The developer edits one step, accepts it, and the record is saved. After release, the bug is published as a knowledge base article, which a later report about a blank filter field retrieves.

8. EVALUATION METHODOLOGY

The quality of an AI-assisted RCA tool cannot be judged by fluency alone. The following methodology is proposed for evaluating the engine. Quantitative results from this evaluation are left for future work and are not claimed in this paper.

8.1 Dataset

A test set of defects is assembled from three sources: bugs recorded in the platform during student projects, resolved issues from open-source repositories whose fixes and discussion are public, and synthetic defects seeded into small applications by the authors. For each defect the ground-truth root cause category, the true cause, and the accepted fix are recorded by at least two reviewers, and disagreements are resolved by discussion.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

8.2 Metrics

Table 2. Proposed evaluation metrics

Metric	Definition
Category accuracy	Proportion of defects for which the suggested category equals the reviewer category
Root-cause correctness	Reviewer rating of the suggested cause as correct, partially correct, or incorrect
Resolution usefulness	Reviewer rating on a five-point scale of whether the steps would lead to a correct fix
Time to diagnosis	Time taken by developers to reach a correct diagnosis with and without the tool
Recurrence rate	Share of new bugs whose category and module repeat an earlier closed bug
Schema validity	Proportion of model responses that pass parsing and validation on the first attempt
Acceptance rate	Share of suggestions accepted unchanged, edited, or rejected by developers

8.3 Experimental design

Two conditions are compared: participants diagnose defects with the platform alone (control), or with AI suggestions on the RCA screen (treatment), with order counterbalanced. Three engine variants are compared: report only, report plus code snippet, and report, snippet, and retrieved articles, to isolate each input's contribution.

9. DISCUSSION

9.1 Expected benefits

Students and junior developers receive an explanation of the failure and a path to a fix, testers gain a consistent vocabulary for describing defects, and managers gain dashboard visibility into which categories of defect dominate. Organisations also retain knowledge that would otherwise leave with a team member, and the link between bugs, RCA records, and releases improves traceability when deciding whether a version is ready to ship.

9.2 Limitations

- **Dependence on input quality.** Vague reports lead to vague analyses. The engine can ask for more information but cannot create it.
- **Hallucination risk.** The model may propose a cause that is plausible but wrong. The review step reduces but does not eliminate this risk, particularly if reviewers over-trust fluent text.
- **Limited view of the code base.** The model sees only the snippet and trace provided and may miss interactions between distant modules.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

9.3 Ethical and privacy considerations

Bug reports and logs can contain personal data, credentials, and proprietary code. Masking, the option of a local model, and explicit labelling of AI-generated content address part of this risk, and responsibility for any fix rests with the developer who accepts it. The interface also shows the evidence behind each diagnosis so that students learn to follow the reasoning rather than over-rely on the tool.

10. CONCLUSION AND FUTURE WORK

This paper presented a Generative AI-assisted approach to root cause analysis and intelligent resolution recommendation for software bugs, embedded in a bug lifecycle and release management platform. The approach turns an unstructured bug report into a structured, reviewable analysis consisting of a categorised root cause, step-by-step resolution, and preventive action, and it retains validated analyses in a searchable knowledge base for later reuse. A working prototype of the lifecycle, RCA, knowledge base, release, deployment-verification, and dashboard modules was described, together with the design of a modular language-model engine, its guardrails, and a methodology for its evaluation.

The work shows that Generative AI is most useful in bug management when it is treated as a collaborator inside an existing workflow, with structured output, human approval, and a path from individual fixes to shared knowledge, rather than as a stand-alone question-answering tool. Such an arrangement can reduce manual debugging effort, support consistent decisions across a team, and contribute to better software quality, maintainability, and release reliability.

Future work includes the following directions: carrying out the proposed evaluation and reporting quantitative results; adding semantic retrieval with embeddings; supporting repository-level context so that the engine can inspect recent commits and related files; automatic duplicate detection at report time; integration with continuous integration pipelines to attach test failures directly to bugs; support for locally hosted open models for privacy-sensitive settings; and the study of how AI-assisted RCA affects the learning of debugging skills among students.

REFERENCES

[1] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?", IEEE Transactions on Software Engineering, vol. 36, no. 5, pp. 618-643, 2010.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?", in Proc. 28th International Conference on Software Engineering (ICSE), 2006, pp. 361-370.

[3] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing", in Proc. 29th International Conference on Software Engineering (ICSE), 2007, pp. 499-510.

[4] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug", in Proc. 7th IEEE Working Conference on Mining Software Repositories (MSR), 2010, pp. 1-10.

[5] IEEE Std 1044-2009, IEEE Standard Classification for Software Anomalies, IEEE, 2010.

[6] A. Vaswani et al., "Attention is all you need", in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.

[7] T. Brown et al., "Language models are few-shot learners", in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020.

[8] M. Chen et al., "Evaluating large language models trained on code", arXiv preprint arXiv:2107.03374, 2021.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

[9] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models", in Advances in Neural Information Processing Systems (NeurIPS), vol. 35, 2022.

[10] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan, "Recommending root-cause and mitigation steps for cloud incidents using large language models", in Proc. 45th International Conference on Software Engineering (ICSE), 2023.

[11] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks", in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020.

[12] M. Grinberg, Flask Web Development, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)