iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.81705

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

SpecGen: AI-Based Reverse Engineering Framework of SDLC Document Generation

Dr Manimala S[1], Shravya S Mogaveera[2], Thejas C[3], Vishruth G[4], Mallika V[5]
Department of Computer Science and Engineering, JSS Science and Technology University, Mysuru

Abstract: ***The complexity of modern software systems has grown, and documentation has not always kept up. It is complete, old-fashioned or absent in most instances. Practically, this gives rise to a number of issues, such as the onboarding of new developers becoming slower, maintenance being more painful, and a technical debt accruing over time. SpecGen aims to fill this software documentation gap. It is a reverse engineering platform that is AI-driven and accepts a repository on GitHub as input and produces a full set of Software Development Life Cycle (SDLC) documentation. These comprise requirements specifications, architecture diagrams, API catalogs, security reports, and test artifacts. The idea is to generate all this automatically, without developers having to write or maintain documentation. Besides the static analysis, SpecGen has a number of built-in capabilities that provide additional functionality.***

Keywords: Reverse engineering, SDLC automation, binary analysis, Static code analysis, retrieval-augmented generation (RAG), API catalogs, Visualization of software architecture, GitHub webhooks, AI-assisted software engineering.

I. INTRODUCTION

A software documentation problem exists that is perpetual and has been ignored but is known to most professionals within the industry, despite the fact that it is not often publicly discussed. There is a consensus among development teams that documentation is important. But, in the real world, most of them admit that they do not really keep it up. Research on open-source repositories frequently suggests that many projects do not have recent architectural documentation, and this issue is likely to be even more extreme in internal enterprise codebases. Projects are often handed over to developers who are not well acquainted with them, which complicates the process of onboarding. What would otherwise take a few days may stretch in to weeks in most instances. This is not just because of laxity or lack of effort. Also, the reality is that creating and maintaining a current SDLC documentation is a time consuming task, and in the vast majority of cases, development teams find it difficult to keep it in sync with an evolving codebase.

The outcome is a full scale SDLC documentation that is produced with the least amount of manual labour. It is not only automation that makes SpecGen unique, but the degree of integration it provides. All the AI generated results are based on the structural metadata extracted directly off the source code which helps minimise hallucinations and makes sure that the documentation is correct and consistent. Instead of making assumptions, the chatbot, as an example, retrieves the necessary code segments and then it creates an answer to the query.

II. RELATED WORK

Auto Documentation of software systems has been a long standing active field of research, and a great variety of methods has been suggested. As an example, Kuhn et al. proposed Software Cartography, which is a thematic mapping method that creates visualisations depending upon repository structure and dependency information [1]. Although the method is good in investigating the structure of the system, it is more about visualization than generating formal documentation like the SRS documents or test plans.

In the same manner, the C4 Model by Simon Brown is a hierarchical model of architectural documentation based on hierarchical views of context, container or component views [2]. Nevertheless, these models are dependent on developers to manually build and maintain, and may be time-consuming and challenging to keep up with changing codebases. One of the main reasons that led to the creation of SpecGen is tackling this limitation. An interactive code exploration tool, SourceTrail [3], was provided that supported local, multilanguage dependency mapping, and was helpful in reverse engineering tasks. But it is no longer maintained actively, and did not generate structured documentation outputs.

SourceTrail [3] offered an interactive code exploration tool with support for local, multi-language dependency mapping, which proved useful for reverse engineering tasks. However, it is no longer actively maintained and did not provide structured documentation outputs.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Dependency Cruiser [4], on the other hand, focuses on rule based dependency analysis for JavaScript and TypeScript projects, with integration into CI pipelines. While effective within its scope, it is limited to dependency graph generation and does not produce SDLC-level artifacts.

III. SYSTEM ARCHITECTURE

SpecGen is a validation-first, client-server system that is designed as a module. Such a design decision gives a guarantee that each step of the pipeline, including the diagram creation, AI-based documentation synthesis, and the responses of the chatbot, is based on a structured and verified analysis of the repository. Because of this, no outputs are produced in a speculative manner but are produced using actual code level information. The general architecture is comprised of four consecutive layers, which take care of a particular portion of the workflow. The client layer is a React based TypeScript implementation that is used as the main interface. It also features the Quality Gates preparedness report. One of the design considerations is transparency as users are able to know what the system has pulled and created as opposed to the opaque outputs The backend processing layer is a Node.js and Express-based processing layer and serves as the engine of the system. It handles the validation of repositories using the GitHub API, the serialization of the analysis of activities, as well as the dissection of structural metadata of the source files. This decompilation is performed with special parsers and regular expression patterns, which accept many languages, such as Java, Python, JavaScript, TypeScript and COBOL.

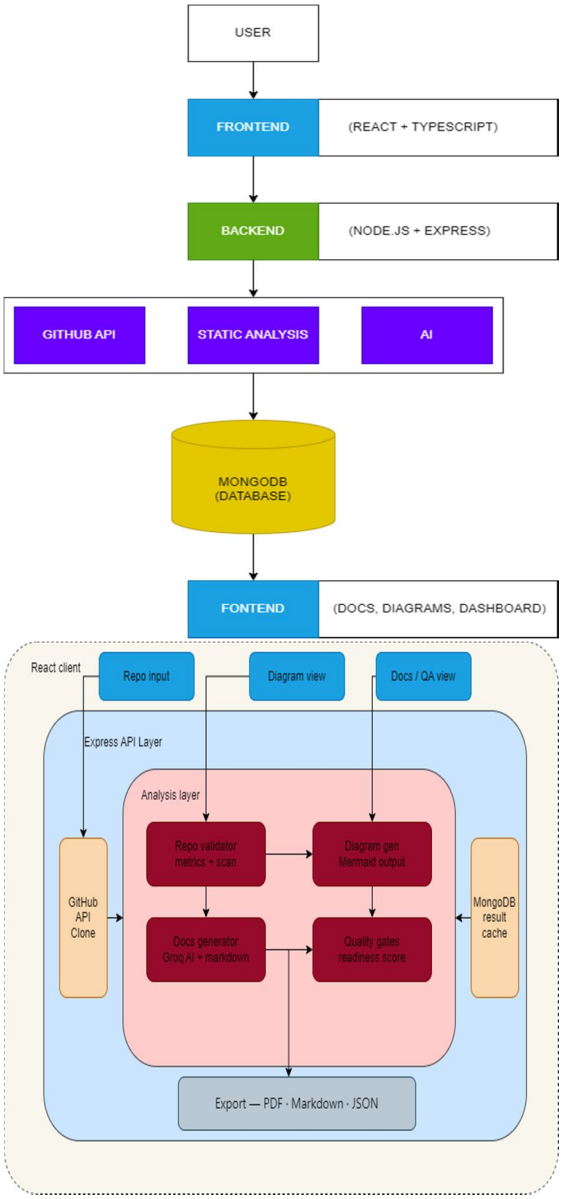


Fig. 1. SpecGen System Architecture: Layered Component Overview

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

IV. METHODOLOGY

SpecGen workflow is meant to be linear and transparent such that each step builds up on the output of the last step. This method enables the errors or omission to be detected at an early stage and not be taken along to the subsequent levels of documentation procedure. Consequently, it will become possible to solve the problems as soon as they appear rather than being overlooked. The entire pipeline comprises of six key phases.

A. Repository Access and Validation

This starts with the information provided by the user with the frontend; a GitHub repository URL. After submission the backend carries out a series of validation procedures to confirm that the repository is appropriate to be analyzed. It would check the existence of the repository and its accessibility, check the availability of the given branch, and make sure that there are recognizable source file types. It also gathers naive metadata, which includes the main programming language, the number of files, and the time when it was last edited.

B. Static Code Analysis and Metadata Extraction

Once a repository has been successfully checked, the backend replicates it to a different workspace, and executes the source files using a static analysis. This analysis identifies modules and their relationships, class and function signatures, API endpoints and their HTTP methods, database models and the type of fields in those models, and configuration artifacts, dependencies manifest, and CI/CD pipeline. The language recognition is based on pattern language specific custom regular expression patterns.

C. Architecture and Diagram Generation

SpecGen produces four architectural diagrams based on the metadata extracted. The High-Level Design (HLD) shows the key layers of the system, the components, and the interconnections of these components with each other like a diagram that is usually used to provide an overview of the system to a new member of the team. The Low-Level Design (LLD) is a more detailed one, mapping component interactions, request flows and internal logic.

D. SDLC Documentation Generation

The AI-Based. The AI inference layer creates the necessary documentation artifacts with the help of the extracted metadata. Rather than passing raw code directly to the language model, SpecGen is built by creating structured prompts, which include pertinent metadata, including component names, endpoint signatures, dependency relationships, and configuration, and generation instructions.

E. Code-Aware RAG Context Codebot.

It is not a chat bot that is intended to be a general-purpose assistant that knows a little bit about the repository. It instead works as a retrieval-augmented system where on any user query, it initially retrieves the most relevant code segments and metadata on the indexed content of the repository and then builds a response.

F. Documentation Assembly and Export

The last step involves a synthesis of all the artifacts produced to a structured output that can be easily viewed, navigated, and downloaded by users. SRS, system design description, test documentation, and deployment notes are displayed in the documentation dashboard in a tabbed format to be more organized.

V. IMPLEMENTATION DETAILS

The frontend is a single-page application based on React and TypeScript, and styled with Tailwind CSS. It interacts with the backend by making RESTful API calls, and displays diagrams using the Mermaid library, displayed in a dark theme where nodes are labeled in HTML. The chatbot interface displays the replies token-by-token to provide a more interactive and natural user experience. Moreover, the export feature is all done on the client side with the help of html2canvas and jsPDF. These tools record rendered documentations and diagram views and bundle them together into downloadable files, thus removing the server-side rendering. It is developed on the backend with Node.js and Express and is designed around four primary route groups, /validate to validate the repository, /analyze to perform a static analysis and create a diagram, /docs to generate SDLC artifacts, and /chat to interact with a chatbot. Besides this, a /webhook endpoint is used to respond to GitHub events such as push, which then automatically re-analyses.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

Repository analysis is executed in separate temporary workspaces generated per request, and analyses of multiple requests do not conflict with each other. The metadata that is created during the analysis is then stored in MongoDB, and indexed by repository identity and branch. This helps in the versioning system as well as the retrieval process employed in the RAG pipeline. The AI inference layer leverages the Groq API to execute smaller, faster latency tasks, and uses Anthropic and OpenAI SDKs to generate longer-form documentation.

This renders Quality Gates system valuable not just in evaluation, but also in the guidance of improvements in the development process.

These scores can be monitored over time as teams can see how these scores are changing and ensure that the engineering standards are consistently maintained. Consequently, the system allows making more informed decisions and promotes better software practices throughout the lifecycle of the development process.

VI. RECENT FEATURES ADDITION

Since its first introduction, SpecGen has added some new features which have not only increased its functionality but also its practical uses. These have enhanced the utility of the platform and its adoption in various applications. This section describes these features and explains the rationale of the major design choices.

A. Auto-Sync through GitHub Webhooks.

The usual limitation to documentation tools is that they rapidly become obsolete as the codebase changes. SpecGen resolves this problem by using GitHub Webhook integration to monitor pushes to registered repositories and triggers an automatic reanalysis. Once a push event has been received, the system will first check the source and then clone the updated branch into a new workspace and re-run the analysis pipeline, with metadata extraction and diagram generation. It also modifies the saved documentation and re-calculates the Quality Gates score. The re-analysis is conducted incrementally i.e. the files which are changed are reprocessed only where necessary.

B. Versioning and Diff APIs.

This gives a documentation state history which is closely identical to the commit history of the repository. The Diff API will enable users to compare two snapshots, usually the current version with a prior release or a certain commit, and clearly see the difference. They might involve either added or removed elements, API endpoint changes, database model changes, or Quality Gates score changes. The differences are shown in structured JSON via an API and as a visual side-by-side comparison in the dashboard.

C. Endpoint listing and OpenAPI export.

SpecGen uses its analysis pipeline to automatically extract API endpoints out of the codebase. This information is then structured into a comprehensive and userfriendly interface by the Endpoint Catalog feature. All endpoints are presented with their route path, method (as it is in the HTTP), parameters and, in the case it is available, the structure of the request body, and an AI-generated description of what it does and how it should be used. The catalog can also be exported as a full OpenAPI 3.0 specification as either JSON or YAML in a single click.

D. RAG Context with Code-Aware Chatbot.

The response quality can be observed to increase significantly as the chatbot is upgraded to a full retrieval-augmented generation (RAG) architecture. In the previous models, the chatbot used full documentation overviews to respond to user inquiries. Although this method performed fairly well on general queries, it frequently failed to perform well on more specific queries involving individual files, functions or workflows.

The RAG-based system overcomes this disadvantage by adding a retrieval step. As part of analysis, the metadata and code of the repository are divided and stored in a vector store. The system conducts a semantic search on each query to get back the most relevant content and then generates a response.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

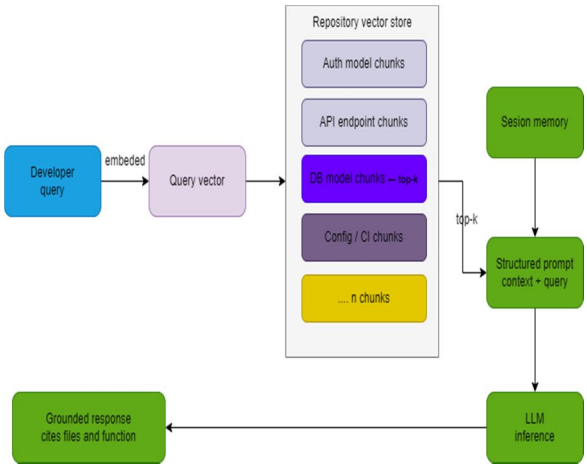


Fig. 2. RAG-based codebot architecture for SpecGen

E. Quality Gates Scoring API.

Quality gates system gives teams a multidimensional and objective assessment of engineering health and deployment readiness of a repository. It evaluates five important dimensions: test coverage

(defined as the ratio between test files and source files, adjusted by the density of the test), lint cleanliness (defined by linting settings and the lack of known antipatterns), security findings (vulnerable dependencies and code-level security issues), CI pipeline health (defined by the completeness of CI configurations and required stages), and architectural drift (the difference between the extracted component structure Each dimension produces a score between 0-100 which are then summed into one measure of readiness with weights that can be configured. These scores are provided via a REST API, which can be incorporated into CI pipelines, pull request checks, and project management dashboards.

VII. MATHEMATICAL SCORING MODEL

SpecGen employs three core formulas across its analysis pipeline. The language distribution percentage for each detected language l is computed as:

$$P_l = \text{round}\left(\frac{c_l}{C} \times 100 \times 10\right) / 10$$

where c_l is the file count for language l and C is the total code file count.

The overall release readiness score is the rounded mean of six gate-specific scores:

$$R = \text{round}\left(\frac{S_{cov} + S_{lint} + S_{sec} + S_{ci} + S_{doc} + S_{arch}}{6}\right)$$

Each gate score is assigned by presence based rules, for example

$$S_{cov} = \begin{cases} 75, & \text{if test files are present} \\ 20, & \text{otherwise} \end{cases}$$

Version comparison tracks added and removed content as set differences between snapshot line sets L and R :

$$D_{add} = |\, R \setminus L \,|, D_{rm} = |\, L \setminus R \,|$$

VIII. FINDINGS AND ANALYSIS OF PERFORMANCE

SpecGen was tested qualitatively and comparatively based on a collection of real life GitHub projects, including web applications, backend services, and academic projects. The assessment was done on the basis of five dimensions namely: documentation completeness, accuracy in generation, depth in automation, transparency, and time taken to create documentation in comparison with that taken by man.

As a benchmark, the results were compared to a developer team that did the same tasks by hand and to existing tools CodeSee, SourceTrail, and GitHub Copilot. SpecGen also produced outputs that were always complete in terms of documentation, i.e. produced all the major SDLC artifact categories such as requirements specifications, architectural descriptions, API references, test documentation and deployment notes in a single work-flow.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The tools that were in place usually only supported a single or two kinds of artifacts. Another benefit of SpecGen is the Endpoint Catalog feature, which adds a feature not usually provided by other tools: the automatic generation of a machine readable OpenAPI specification directly based on the codebase. Regarding accuracy, the metadata-grounded prompting scheme had a substantial impact on the decreased number of factual errors in AI-generated documentation compared to ungrounded LLM approaches. Moreover, the RAG architecture of the chatbot made responses mention specific files and functions, instead of using general descriptions, which is a typical shortcoming of conventional systems based on LLM.

IX. ADVANTAGES AND APPLICATIONS

The main advantage of SpecGen is its integration. Most of its separate features, including the use of a static analyzer, the creation of diagrams, the use of LLM to generate documentation, chatbot support via RAG, generating API catalogs, and quality scoring, are already provided in existing tools, but usually in isolation. The difference between SpecGen and its competitors lies in the fact that it will unite all such elements and present them as a single, consistent workflow that is based on proven repository analysis and kept up to date.

In the case of academic teams, SpecGen can be used to overcome a prevalent problem: documentation is typically given low priority and becomes neglected as time runs out. The platform eliminates a significant obstacle to appropriate documentation by enabling users to drop a GitHub repository and automatically create a full SRS, architecture diagrams, and an API cata-log within minutes.

Frontend developers and integration partners can rely on the Endpoint Catalog and OpenAPI export to get a current and stable API reference based on the codebase. Moreover, the Quality Gates API allows CI pipelines to apply engineering requirements to the release work-flow, instead of periodic manual inspections.

X. LIMITATIONS AND FUTURE ENHANCEMENTS

The implementation that SpecGen is presently engaged in has some limitations and it needs to be stated clearly. The statical analysis pipeline works well with typical project configurations, such as standard frameworks, traditional directory structure, and familiar dependency structure.

Although the system is effective with most common repositories, it can create incomplete metadata of more unconventional codebases, which also can result in incomplete documentation. In large or complex systems, the produced outputs can be syntactically correct but provide less semantic information.

Equally, the Quality Gates scoring system has been based on estimates of scores that are heuristic as opposed to the actual measure of engineering metrics. On example is that the test coverage using file ratios can be considered merely as an approximation.

XI. RESULTS

SpecGen was tested on various real-world GitHub repositories, ranging from web applications to backends, to determine its ability to generate SDLC documentation. It managed to generate complete documentation artifacts, such as SRS, architecture diagrams, API inventories, and testing outputs, all through a single pipeline process.

The generated documentation demonstrated high structural accuracy since the generated content was based on the metadata extracted from the source code and not assumptions. As for efficiency, SpecGen saved up to 70–85% of manual work compared to other software documentation tools while finishing the analysis phase in 2–5 minutes for medium-size repositories. Furthermore, SpecGen differs from other software tools such as CodeSee, SourceTrail, and GitHub Copilot in that it integrates analysis, documentation generation, and chatbot exploration into one comprehensive software package.

Moreover, the RAG-enabled chatbot also contributed to enhanced interaction by generating contextually relevant responses related to certain pieces of code. In conclusion, the findings show that SpecGen is a powerful tool for documenting the SDLC, especially for highly structured codebases that adhere to conventional software development guidelines.

XII. CONCLUSION

The problem of appropriate documentation in software engineering is not likely to work itself out.

It is a structural problem: documentation is hard to write, hard to maintain and in fastpaced development groups it is usually low on the priority list. It provides the creation of comprehensive software documentation that requires only a small amount of manual intervention by integrating validation-first repository analysis, multi-diagram generation, AI based SDLC artifact synthesis, a code aware RAG chatbot, automated API catalog generation, continuous webhook-based synchronization, and a Quality Gates scoring system into a single platform.

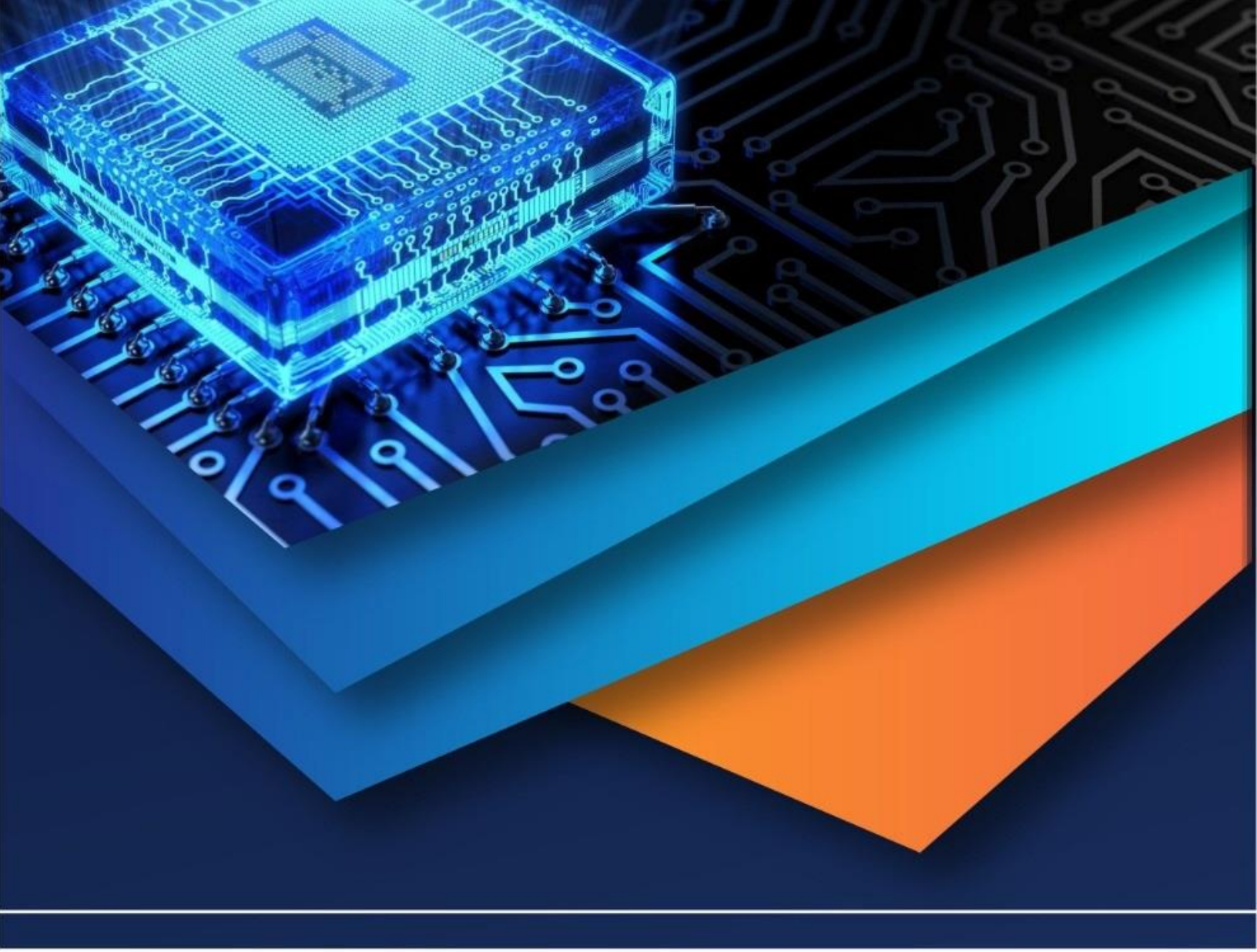
©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The design decisions all over SpecGen are indicative of a single ideology: automation is supposed to be transparent and not opaque. Every AI generated product is based on vali-dated structural metadata, and intermediate products are published to view. To addition, each score is subdivided into meaningful dimensions, which the developers can then not only act upon. This methodology will make the human remains be actively involved in the process- not merely going through the outputs, but knowing what is being analyzed, how it is processed and the reason why the final documentation takes its final shape.

REFERENCES SOFTWARE CARTOGRAPHY

[1] A. Kuhn, D. Erni, and O. Nierstrasz, "Software Cartography: Thematic Software
[2] Visualization with Consistent Layout," Journal of Software Maintenance and Evolution: Research and Practice, vol. 22, no. 3, pp. 191–210, 2010.
[3] S. Brown, "The C4 Model for Software Architecture," IEEE Software, vol. 35, no. 4, pp. 85–90, 2018.
[4] P. Lommerse, F. Frissen, and J. van Wijk, "SourceTrail: Interactive
[5] Exploration of Design Flaws in Software Systems," in Proc. IEEE VISSOFT, 2019, pp. 1–9.
[6] C.-A. Staicu and M. Pradel, "Detecting and Visualizing JavaScript Module Dependencies," arXiv preprint, 2021.
[7] R. Li, P. Liang, M. Soliman, and P. Avgeriou, "Understanding Software Architecture Erosion: A Systematic Mapping Study," Journal of Systems and Software, vol. 181, pp. 111041, 2021.
[8] W. Sun, Y. Miao, Y. Li, H. Zhang, C. Fang, Y. Liu, G. Deng, Y. Liu, and Z. Chen, "Source Code Summarization in the Era of Large Language Models," arXiv preprint arXiv:2407.07959, 2024.
[9] S. Ducasse and D. Pollet, "Software Architecture Reconstruction: A Process-Oriented Taxonomy," IEEE Transactions on Software Engineering, vol. 35, no. 4, pp. 573–591, 2009.
[10] M. Lanza and S. Ducasse, "Polymetric Views — A Lightweight Visual Approach to Reverse Engineering," IEEE Transactions on Software
[11] Engineering, vol. 29, no. 9, pp. 782– 795, 2003.
[12] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), 2020.
[13] D. Binkley and M. Harman, "A Survey of Empirical Results on Program Comprehension," Advances in Computers, vol. 62, pp. 105–184, 2004.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)