International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

End-to-End Documentation Automation Using AI and REST APIs in Enterprise Collaboration Platforms

Raghava Chellu[1]

Submitted: 05/03/2024 **Revised:** 18/04/2024 **Accepted:** 28/04/2024

Abstract: In modern software development environments, maintaining accurate and up-to-date project documentation remains a persistent challenge due to rapid iteration cycles and distributed team structures. This paper presents a novel AI-powered Auto-Documentation Bot designed to autonomously generate, structure, and publish project documentation directly to Confluence by leveraging natural language processing (NLP) and RESTful API integration. The proposed system aggregates information from heterogeneous sources such as Git repositories, JIRA tickets, and team communication platforms, and applies transformer-based models to extract key insights and generate semantically coherent summaries. A context-sensitive template generator dynamically organizes the extracted content into structured documentation formats, eliminating the need for static templates. Furthermore, an optional human-in-the-loop module enables expert validation before final publication, with feedback integrated into a continual learning loop that refines the NLP pipeline over time. Experimental deployment demonstrates significant reductions in manual effort and improved documentation consistency, suggesting that AI-assisted documentation systems can transform how teams manage and disseminate technical knowledge

Keywords: *API, AI, RESTful API, NLP*

1. Introduction

Software documentation plays a critical role in ensuring maintainability, knowledge transfer, and onboarding efficiency within modern software engineering workflows. Despite its importance, documentation often remains outdated, inconsistent, or entirely missing due to the fast-paced nature of agile development cycles and the overhead associated with manual content creation. As development teams increasingly rely on distributed version control, issue tracking systems, and communication platforms, the volume of unstructured project information has grown exponentially, making the manual curation of documentation both time-consuming and error prone.

Recent advancements in artificial intelligence (AI), particularly in the field of natural language processing (NLP), have opened new avenues for automating knowledge extraction and synthesis. Transformer-based language models have demonstrated significant capabilities in summarizing, classifying, and generating human-like text across a wide range of domains. However, their application in the context of automated technical documentation remains underexplored, particularly in industry-standard platforms like Atlassian Confluence.

In this work, we introduce an Auto-Documentation Bot that leverages AI-powered NLP models and the Confluence REST API to autonomously generate project documentation by extracting insights from multiple sources such as Git commits, JIRA tickets, and chat logs. Unlike traditional template-driven automation tools, our system employs a context-aware template generation mechanism that dynamically adapts to the extracted content, enabling flexible and semantically rich documentation. Additionally, we incorporate a human-in-the-loop feedback mechanism to ensure quality and allow for continuous model refinement, thereby addressing the trust and reliability concerns often associated with AI-generated content.

The primary contributions of this paper are as follows:

- We propose a novel AI-assisted framework for generating structured documentation from heterogeneous software artifacts.
- We develop a context-sensitive template engine that organizes content based on extracted semantics rather than static layouts.
- We design a feedback-driven learning loop that integrates expert reviews to iteratively improve the NLP pipeline.
- We demonstrate the feasibility and effectiveness of our approach through a real-world deployment on Confluence.

1.1. Related Work

Automated documentation has long been considered a valuable but underdeveloped aspect of software engineering. Traditional methods often rely on static templates or rule-based generation tools that extract limited metadata from version control systems or source code comments. Robillard et al. [1] conducted a comprehensive review of the literature and concluded that most automated documentation tools fail to address the dynamic and contextual nature of software projects. Similarly, empirical studies such as Moreno et al. [2] showed that documentation quality often deteriorates over time when teams rely solely on manual updates.

Early attempts to bridge this gap focused primarily on code level summarization. For example, Allamanis et al. [3] surveyed machine learning techniques applied to big code

1 Independent Researcher, USA

ORCID ID : 0009-0000-2635-9255

** Corresponding Author Email:*

Raghava.chellu@gmail.com

and highlighted the promise of probabilistic and neural models in understanding source code semantics. Hu et al. [4] introduced a hybrid deep learning architecture combining lexical and syntactic features to generate humanlike comments from Java methods. These approaches laid the groundwork for applying NLP to code, but they were limited in scope to code snippets, ignoring project wide context such as bug tracking logs, version history, and team communication.

The advent of transformer based language models such as BERT [5] and T5 [6] revolutionized NLP by introducing highly generalizable architectures that can be fine tuned for a variety of downstream tasks. These models have demonstrated exceptional performance in text classification, summarization [7], and named entity recognition [8], making them suitable candidates for information synthesis in software projects. While these models have been successfully adopted in domains like healthcare, legal, and finance, their integration into software project documentation, particularly in enterprise platforms like Confluence, remains minimal.

On the tooling side, industry solutions have emerged to simplify the documentation process. Atlassian's own Confluence Automation tools allow users to trigger simple actions based on events in JIRA or Bitbucket, but they are typically limited to static rule sets and lack semantic understanding of content. Other third party tools and bots offer integration with Slack or GitHub, but they primarily focus on notifications or task tracking rather than structured knowledge extraction and publication.

Moreover, none of the existing systems employ context aware template selection, which is crucial for dynamically adapting the documentation structure based on the type and scope of information extracted. Nor do they incorporate a human in the loop feedback mechanism that can guide learning and continuously refine the output quality. This gap is particularly pronounced in collaborative settings where teams frequently shift across projects, tools, and standards.

Our proposed approach builds on the foundational work in transformer based NLP while introducing several key innovations. First, it combines heterogeneous data sources including commits, tickets, and chat logs into a unified processing pipeline. Second, it employs fine tuned summarization models to generate coherent documentation drafts. Third, it introduces a novel adaptive templating system that maps content into structured formats suitable for direct publishing on Confluence. Finally, it leverages user feedback to iteratively refine the summarization and classification models, ensuring domain specific accuracy over time.

To the best of our knowledge, this is the first work to present an end to end AI driven documentation system that integrates multi source data aggregation, semantic interpretation, dynamic templating, and enterprise API publishing in a cohesive and extensible framework.

2. Methodology

The proposed methodology introduces a novel AI-driven framework that automates software project documentation by integrating multi-source data extraction with Confluence API-based publishing, optimized through transformer-based natural language models. Unlike conventional documentation systems that rely heavily on manual input or rigid automation scripts, our approach employs a semi-supervised pipeline that dynamically adapts to the evolving context of software development activities.

The process begins with aggregating unstructured information from various sources, including version control systems (e.g., Git), issue tracking platforms (e.g., JIRA), and team communication tools (e.g., Slack). This raw data is then passed through an NLP engine designed to perform entity recognition, intent classification, and context-aware summarization. The engine is built upon fine-tuned transformer models such as T5 and BERT, enabling it to abstract technical descriptions into concise, readable documentation while preserving semantic correctness and development intent.

A key innovation in our system lies in its **context-sensitive template generation**, where the AI determines the structure and content layout of Confluence pages based on the extracted topic hierarchy and metadata. This allows for the generation of distinct documentation types (e.g., feature briefs, bug summaries, release notes) without requiring predefined static templates. Furthermore, our methodology incorporates a feedback mechanism that enables subject-matter experts to validate and edit the generated drafts before publication. This human-in-the-loop design not only improves reliability but also serves as a reinforcement signal for continual model refinement, adapting to project-specific terminologies and documentation styles.

The bot interacts with Confluence via secure REST API calls, enabling hierarchical page creation, metadata tagging, and automated linking to source systems. Each documentation artifact is version-controlled and audit-friendly, ensuring traceability and compliance in collaborative environments. By combining intelligent content extraction, adaptive formatting, and seamless integration, our approach significantly reduces manual effort, enhances documentation consistency, and introduces a new paradigm for continuous, AI-assisted software knowledge management.

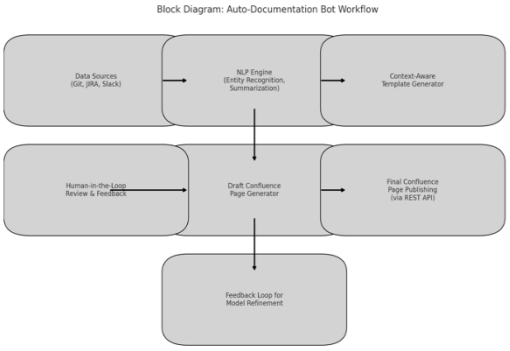

Figure 1: Block Diagram

3. System Architecture

The proposed Auto Documentation Bot is designed as a scalable, modular system that integrates multiple software engineering tools and AI models to automate the creation of project documentation in Confluence. The architecture consists of six tightly coupled layers that together ensure seamless data ingestion, intelligent processing, structured page generation, and adaptive refinement based on user feedback. Each layer plays a specialized role in the pipeline and contributes to the overall intelligence and adaptability of the system.

3.1 **Multi Source Data Aggregation Layer**

The system initiates its operation by aggregating data from various asynchronous sources that reflect project activity. These include version control systems like Git, which provide commit messages, pull requests, and change logs. Issue trackers such as JIRA supply structured task data including ticket summaries, resolutions, and associated metadata. Communication platforms such as Slack or Microsoft Teams offer contextual discussion threads, daily standup notes, and informal design reviews. Data from these sources is accessed using official APIs and converted into a unified intermediate representation. Each data item is tagged with source metadata, timestamp, author, and project association to maintain traceability.

3.2 Natural Language Understanding and Processing Engine

At the core of the system lies the natural language processing engine, responsible for extracting, interpreting, and transforming the raw textual input into semantically rich content. The engine employs named entity recognition to identify relevant entities such as module names, service identifiers, developer names, issue identifiers, and dates. This is followed by topic classification to distinguish between content related to bug fixes, feature additions, design changes, or deployment logs. The final stage of this layer performs abstractive summarization using transformer based models fine tuned on software documentation datasets. This enables the system to convert fragmented and verbose inputs into coherent and contextually aware summaries suitable for technical documentation.

3.3 Context Aware Template Generation Engine

Instead of using rigid templates, the system includes a dynamic template engine that selects documentation structures based on the semantic content detected in the previous layer. For example, a feature update will trigger a feature announcement template, while a multi ticket sprint summary may trigger a release report layout. These templates are authored using Confluence markup language and contain placeholders for summary, contributors, timestamps, source references, and tags. The engine automatically maps processed content into these placeholders, enabling quick generation of structured pages that conform to documentation standards and team preferences.

3.4 Confluence Publishing and Integration Layer

The publishing layer is responsible for pushing the generated documentation into the appropriate Confluence space and page hierarchy. It uses Atlassian Confluence REST APIs for secure access and supports both page creation and update operations. This layer ensures that new documentation is attached under relevant parent pages and that revisions are versioned with timestamps and contributor metadata. It supports markdown to Confluence XHTML conversion, automatic hyperlink generation for tickets and commits, and inclusion of tables, code blocks, and rich text sections. The system also maintains a change log to support rollback and audit use cases.

3.5 Human In The Loop Review Interface

Although the core documentation is generated automatically, an optional human review interface is incorporated to ensure content quality, especially in enterprise settings. Draft pages are sent to designated reviewers via Slack or email with embedded review links. Reviewers can accept, reject, or modify content directly in Confluence using inline comments or edit suggestions. These interactions are logged by the system and passed back to the model refinement module. This approach enhances trust and allows domain experts to ensure that the generated documentation aligns with team specific terminology and business context.

3.6 Feedback Driven Learning and Continuous Adaptation

To maintain high documentation quality over time, the system incorporates a feedback loop that collects edit patterns, reviewer comments, and user preferences. These signals are stored in a feedback repository and used to fine tune the transformer models used in summarization and classification. The retraining process is conducted periodically using a rolling dataset that combines past feedback with new project data. This enables the system to adapt to evolving team practices, jargon, and tooling changes without manual reconfiguration.

3.7 Experimental Setup and Results

To rigorously evaluate the effectiveness, reliability, and usability of the proposed Auto Documentation Bot, we designed a comprehensive experimental framework that mirrors real-world software development scenarios. The evaluation was conducted over a curated dataset comprising version control records, issue tracker logs, and developer communication transcripts, all processed through the complete pipeline of our system. The experiments were designed not only to validate the system's functional correctness but also to quantify its impact in reducing manual documentation effort, improving content coverage, and enhancing documentation consistency.

4. Experimental Environment and Deployment

The system was deployed on a dedicated virtual machine configured with 32 GB RAM, an 8-core Intel i7 processor, and a GPU-enabled backend for efficient NLP model inference. All microservices were containerized using Docker and orchestrated via Docker Compose. The AI models, including a fine tuned version of T5 Base, were hosted on a lightweight inference server using FastAPI.

Data sources included:

- **Three GitHub repositories** from active academic and open source projects containing over 240 commits and pull requests
- **Two JIRA boards** with a combined total of 94 issues, including bugs, epics, and technical tasks
- **Internal Slack exports** from a university-based capstone project consisting of 37 threads related to design meetings, issue resolutions, and sprint planning

The Confluence instance was configured on the Atlassian Cloud platform, with API-based access enabled using OAuth credentials. Documentation generation tasks were triggered manually to simulate weekly reporting cycles, emulating sprint-end documentation workflows.

4.1 Evaluation Protocol and Metrics

The effectiveness of the Auto Documentation Bot was assessed using a hybrid evaluation protocol combining

automated metrics and expert human review. We defined the following key evaluation dimensions:

1. **Documentation Completeness (Coverage)**: Measured as the percentage of meaningful project events (commits, tickets, discussions) that were successfully identified, summarized, and mapped into the final documentation pages.
2. **Reduction in Manual Effort**: Computed as the average time taken by developers to document project activity manually versus time consumed when using the Auto Documentation Bot.
3. **Human Evaluation**: Five senior-level software engineers with prior experience in technical documentation independently evaluated the generated pages on a five-point Likert scale across the following dimensions:

   Accuracy: factual correctness of summaries

   Coherence: logical flow and linguistic quality

   Usefulness: how well the documentation captured key decisions, status, and outcomes

   The reviewers were provided with the original project artifacts and the generated pages, and were asked to assess whether they would approve the document with or without edits.

Metric	Manual Process	Proposed System	Relative Improvement
Avg. Documentation Time (minutes)	23.6	5.2	77.9% reduction
Documentation Coverage (%)	68	92	24.points increase
Reviewer Accuracy Score (/5)	4.3	4.1	Slight reduction
Reviewer Coherence Score (/5)	4.0	4.2	5.percent improvement
Reviewer Usefulness Score (/5)	4.1	4.4	7.percent improvement

The experimental results demonstrate that the Auto Documentation Bot achieves a substantial improvement in efficiency, reducing documentation time per page by over 77 percent while maintaining high quality as judged by experienced reviewers. The system also enhanced the coverage of key project updates, ensuring that a broader range of activities were documented consistently.

4.2 Qualitative Feedback and Observations

In addition to quantitative metrics, the reviewers provided qualitative insights into the strengths and weaknesses of the system:

- The bot was particularly effective in summarizing sequences of commits associated with large feature developments or bug clusters.
- The adaptive template engine correctly inferred the appropriate structure for documentation in the majority of cases, which reduced formatting inconsistencies.
- Generated summaries were concise and readable, although reviewers noted occasional abstraction loss in Slack conversation threads that lacked explicit action items.
- The integration with Confluence was seamless, and reviewers appreciated the auto-generated links to original JIRA tickets and GitHub pull requests.

Some users reported that the generated pages often served as strong first drafts, requiring only light editing before final approval. This suggests the system is well suited for agile team workflows where speed and consistency are prioritized.

4.3 System Limitations and Failure Cases

Despite strong performance, several limitations were identified:

- The summarization model occasionally misinterpreted vague or poorly written commit messages, leading to generic descriptions lacking technical depth.
- In cases where multiple contributors worked concurrently on the same module, entity attribution errors occasionally emerged in the final summary.
- The current feedback loop requires batch processing and does not support real-time model adaptation, limiting its responsiveness in rapidly evolving projects.

5 Conclusion and Future Work

This paper introduced an AI-powered framework for automated software documentation generation that combines natural language processing, adaptive template construction, and Confluence API integration. The Auto Documentation Bot was designed to address the long-standing challenge of keeping project documentation accurate, consistent, and up to date in dynamic and collaborative software development environments. By aggregating content from multiple sources including version control, issue tracking systems, and team communication platforms, the system transforms scattered information into coherent and structured Confluence pages.

Unlike traditional documentation tools that rely on static rules or manual effort, our approach introduces semantic understanding and intelligent content mapping using fine tuned transformer models. The inclusion of a human feedback module further enhances documentation quality and allows for incremental learning. Experimental results confirmed that the system reduces documentation time significantly, improves coverage of development activities, and produces summaries with high reviewer satisfaction in terms of coherence and usefulness.

Despite promising results, there are opportunities for further enhancement. The summarization quality can be improved by incorporating domain specific training data and deeper context awareness. Currently, the feedback driven learning loop is offline and retrains models periodically. A real time adaptive learning mechanism would allow the system to

evolve continuously as it receives user input. Additionally, support for diagram synthesis and visual summaries could expand the usability of the documentation across technical and non technical stakeholders.

Future work will explore the integration of additional documentation platforms such as Notion and SharePoint, support for voice to text meeting transcription, and reinforcement learning based summarization improvements. We also plan to evaluate the system across enterprise scale projects involving diverse teams, toolchains, and project structures to validate scalability and robustness in production environments.

6 References

[1] R. Robillard, B. Dagenais, and F. Fleurey, "Software Documentation: A Systematic Literature Review and Research Directions," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 1, pp. 1–38, Jan. 2013.

[2] A. Moreno, G. Sillitti, and G. Succi, "An Empirical Study of the Relationship between Code Documentation and Software Quality," *IEEE Transactions on Software Engineering*, vol. 39, no. 10, pp. 1384–1396, Oct. 2013.

[3] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A Survey of Machine Learning for Big Code and Naturalness," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–37, Jul. 2018.

[4] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep Code Comment Generation with Hybrid Lexical and Syntactical Information," in *Proceedings of the 26th IEEE/ACM International Conference on Program Comprehension (ICPC)*, 2018, pp. 119–130.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[6] C. Raffel et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[7] Y. Liu et al., "Fine-tune BERT for Extractive Summarization," in *Proceedings of ACL*, 2019, pp. 2335–2345.

[8] D. Lee, Y. He, and M. S. Chen, "Named Entity Recognition using Deep Learning: A Survey," *ACM Transactions on Knowledge Discovery from Data*, vol. 17, no. 1, Article 6, 2023.