D.aise: An Automated README Generator from Source Code Repositories

Arthur Alesi
Department of Informatics, PUC-Rio
Rio de Janeiro, Brazil
aalesi@inf.puc-rio.br

Gabriela Soares
Department of Informatics, PUC-Rio
Rio de Janeiro, Brazil
gsoares@aise.inf.puc-rio.br

Juliana Alves Pereira
Department of Informatics, PUC-Rio
Rio de Janeiro, Brazil
juliana@inf.puc-rio.br

Abstract

Software documentation is essential for software comprehension and maintainability, with README files serving as one of the main documentation artifacts in software repositories. However, maintaining clear and up-to-date README files is often neglected due to the effort required from developers. To address this problem, we developed D.aise, a tool for automated README generation and maintenance using Language Models (LMs). The platform integrates technologies such as Python, React, and Ollama, while also supporting external providers such as ChatGPT and Gemini through APIs. D.aise can analyze repositories, generate README content from repository structures and source files, and allow users to edit and apply the generated documentation directly to repositories. We evaluated the tool through a human-centered study in which participants assessed its usability and practical usefulness for documentation workflows.

Demo video: https://zenodo.org/records/20386248

Keywords

Documentation, Software Repositories, Language Models

1 Introduction

Large organizations often maintain dozens or hundreds of repositories [3, 8, 14]. Alongside source code, documentation plays an important role in software development and maintenance. In particular, the README file is commonly used as the main entry point for developers and users, containing information about installation, usage, architecture, and contribution guidelines [17]. Well-structured documentation facilitates onboarding, improves maintainability, and supports collaboration across distributed teams [1, 3]. However, maintaining repository documentation remains a persistent challenge. As software projects evolve through continuous development, refactoring, and feature updates, README files frequently become outdated, incomplete, or inconsistent with the current state of the project [1, 3, 19]. Additionally, in many cases, documentation is neglected or written without standardized practices, reducing its usefulness for both contributors and end users [1].

Recent advances in Language Models (LMs) have created new opportunities for automatically extracting repository information and generating software documentation [4, 5, 11, 13, 16]. In an informal and highly individualized manner, developers use models such as GPT and Gemini to assist in README creation and refinement. Different developers may use distinct prompts, workflows, and strategies. More experienced users may develop effective prompting techniques and validation methods, while less experienced developers may struggle to obtain satisfactory results due to limited familiarity with LM interaction practices [4]. This lack of centralized management and reusable prompt strategies can lead to inconsistencies, duplicate effort, and reduced collaboration efficiency.

To address these limitations, we propose D.aise a platform designed for README generation, update, and prompt benchmarking using LMs. The platform enables users to extract and organize repository information, manage reusable prompts, and experiment with different documentation-generation strategies within a unified environment. D.aise adopts a client-server architecture composed of a Python-based backend and a React frontend built with Next.js. Both layers communicate through a REST API. The backend is responsible for repository processing, integration with external APIs, prompt management, and communication with supported LM providers. The system supports local model execution via Ollama, cloud-based interaction with the OpenAI and Gemini APIs via user-provided authentication tokens. Additionally, D.aise integrates with the GitHub API to retrieve repository metadata and project information directly from remote repositories. D.aise can be deployed locally and used through a web-based interface. The frontend provides an interactive environment for repository selection, prompt creation, README editing, and comparative evaluation of generated outputs.

Unlike existing approaches that rely on isolated prompt usage [5, 13] or general-purpose AI (*e.g.* GitHub Copilot and Cursor), D.aise is specifically designed for structured README generation workflows. D.aise organizes the documentation process as a reproducible pipeline, where repository context, prompts, and generated outputs are explicitly stored, reused, and compared. This enables systematic comparison of different prompting strategies under controlled conditions, rather than ad hoc interactions typical of general-purpose tools. Moreover, its support for multiple LM providers enables consistent evaluation across models, isolated testing of individual models, and local self-hosted deployment without external service costs, facilitating more structured experimentation and refinement of documentation workflows.

The user-centered evaluation with 18 developers demonstrated that D.aise provides an intuitive and usable environment for automated README generation and maintenance. Participants successfully completed documentation-related tasks and reported positive perceptions regarding the platform's ease of use, usefulness, and integration of repository analysis with centralized prompt management, indicating its potential to support real-world software documentation workflows.

2 Background and Related Work

README files are a standard form of software documentation, typically written in Markdown (.md) and located at the root of a project repository. They describe a project's purpose, installation process,

usage instructions, dependencies, and contribution guidelines. Well-maintained README improve usability and collaboration, although keeping them synchronized with codebase evolution remains a recurring software maintenance challenge.

Automated generation of README files using LMs is an emerging area with few published studies. ReadMeReady [5] is an LM-based tool that generates documentation for public and custom repositories, supporting fine-tunable open-source models as an alternative to proprietary APIs. Its evaluation using BLEU and BERTScore metrics found high semantic similarity between generated and original READMEs. Unlike our approach, ReadMeReady does not support README maintenance in response to ongoing commits.

A benchmark for evaluating code-to-README generation was introduced by Patched.codes [13], consisting of 198 open-source Python repositories. Their evaluation of multiple LMs found Gemini 1.5 Flash to be the strongest performer. Notably, this approach feeds the entire repository content into the model's context window, whereas our system selectively extracts structured metadata information — directory tree, dependency file, commit history — to stay within practical token limits and reduce noise.

GitHub Copilot[1] and Cursor[2] are AI-assisted development tools commonly used for code generation and project analysis. GitHub Copilot is typically integrated into IDEs such as Visual Studio Code, while Cursor is an IDE with native AI integration. Both tools allow developers to analyze files, interact through chat interfaces, and request code modifications. However, neither tool was designed for prompt benchmarking or structured evaluation of prompt strategies, and both primarily focus on general software development rather than README generation workflows. Additionally, they rely on internal prompts and orchestration mechanisms that are not fully visible to users, reducing transparency and reproducibility. Both platforms also depend on commercial subscription models. In contrast, our tool enables systematic exploration and evaluation of multiple prompting strategies while supporting integration with local models through tools such as Ollama.

The systematic evaluation of prompt engineering strategies for SE tasks has been explored in several studies. Dvivedi et al. [7] compared multiple LMs and prompting configurations for code documentation generation at the function level, finding that prompt design meaningfully influences output quality. To the best of our knowledge, no published study has proposed a dedicated prompt design for each stage of an automated README generation and maintenance pipeline — covering project analysis, creation from scratch, and commit-driven updates — and evaluated the resulting tool through a structured user study.

3 D.aise Tool

D.aise provides a unified platform that automates the generation and maintenance of README files through the integration with Language Models (LMs). The tool uses a repository as input and produces README documentation as output. To this end, D.aise exposes three LM-driven operations: *Analyze Project*, which extracts structured metadata from the repository structure; *Create* README, which generates a complete README from scratch; and *Update* README, which revises an existing document to reflect recent commit activity. These operations are backed by dedicated prompt templates used at runtime by the backend's agent layer. The following sections describe each component of the platform in detail.

D.aise is built on a Python/Flask backend and a Next.js frontend, and supports both cloud-hosted LMs via the Google Gemini API [10] and locally-hosted open-weight models via Ollama [12].

Repository Input. The first step in the D.aise workflow is the input of a target software repository. The platform supports two acquisition modes: (1) local repository selection and (2) remote import of GitHub repository via clone or GitHub REST API [9]. Once imported, GitHub repositories can be refreshed on demand, pulling the latest file tree without re-importing the project.

The user can configure their personal access token directly in the application header, accessible from any page. Once saved, the token is used transparently by the backend in all subsequent API calls that require authentication. Notice that two D.aise functionalities require authenticated access to the GitHub API: importing and updating private repositories and committing generated documentation back to a repository.

Project Analysis. This functionality instructs the language model to inspect the repository's file tree, dependency manifest, and any existing description, to extract structured metadata from the repository structure. It returns a JSON object with inferred metadata, such as project name, short description, primary programming language, framework, dependency information, and main entry-point file. The metadata values are populated on the project screen of D.aise, where the user can review and correct any inaccuracies before proceeding to documentation generation. Notice that the metadata are either filled in manually by the user or extracted through this functionality.

README Generation. This functionality targets repositories that do not yet have a README file. The user selects what project metadata from Table 1 to forward to the model. The backend uses GitPython [15] to traverse the repository's Git log and assemble the selected information, which is injected into the `create_readme` prompt template. The response is parsed to extract the generated Markdown, which is displayed in a result panel with two switchable views: a *Markdown* tab showing the raw text and a *Preview* tab rendering it as formatted HTML. Once the user is satisfied with the result, the document can be downloaded as a `.md` file, applied directly to the local filesystem, or committed to the remote GitHub repository via a modal that prompts for the user to enter with a commit title and commit message.

README Update. This functionality targets repositories whose README already exists but has drifted from the current state of the codebase. The user configures the update through a modal offering two strategies for defining the relevant commit window: (1) *since the last* README *modification*, which automatically selects all commits made after the README was last changed; or (2) a *custom date range* with explicit start and end dates. The user also selects what metadata from Table 1 to forward to the model. The backend uses GitPython [15] to traverse the repository's Git log and assemble the selected information, which is injected into the `update_readme` prompt template alongside the current README content. The result is displayed in a side-by-side comparison panel, where the left pane shows the *Current* README and the right pane shows the *Updated* README. Each pane offers independent *Markdown* and *Preview* tabs.

[1] https://github.com/features/copilot
[2] https://cursor.com/

The same deployment features available during generation are offered once the user accepts the result.

Table 1: Available metadata used for prompt engineering

metadata	description
project name	Project name.
description	Project description.
language	Main programming language.
framework	Framework used in the project.
dependency file content	Dependency file content.
dependency file name	Dependency file name.
tree	Repository file tree.
commits (title and description)	Commit titles and descriptions.
commits (diff)	Commit diff content.
since last README modification	Collect commits since the last README modification.
start date	Start date for commit collection.
end date	End date for commit collection.

Prompt Lab. A central component of `D.aise` is the *Prompt Lab*, which treats prompt templates as configurable artifacts, allowing users to author, store, modify, and activate different prompting strategies at runtime without changes to the application code. The Prompt Lab supports multiple types of prompt corresponding to the three LM-based functionalities: *Project Analysis*, `README` *Generation* and `README` *Update*. Each prompt is authored as a free-form text template containing named placeholder tokens enclosed in double braces, which are resolved at runtime by substituting values from the current project metadata (see Table 1). The complete specification of all placeholders and the structural schema of each template are provided in our repository.

Users can save multiple prompts for each functionality to a persistent library and designate any saved prompt as the active one for its corresponding operation. It allows experimentation with alternative templates without losing prior working configurations.

Prompt Templates. D.aise ships with nine predefined prompt templates that cover four prompting strategies: Zero-Shot, Few-Shot, Role Prompting, and Chain-of-Thought. The full structural schema of each template is provided in our repository.

LM Configuration. `D.aise` is model-agnostic at the user level, exposing a configuration interface for selecting the LM provider, model used, and API credentials. The platform currently supports two providers with complementary characteristics.

Google Gemini [10] is a cloud-based provider accessed via a personal API key issued through the Google AI platform. As a managed service, it offloads computational overhead to Google's infrastructure and requires no local hardware setup. API access is subject to usage-based pricing; however, Google provides a free tier that allows students to obtain an API key and issue requests within a limited token quota, making it viable for small-scale experimentation without incurring costs.

Ollama [12] is a self-hosted inference server that enables the use of open-weight models locally, without relying on external cloud services. This integration is particularly relevant for users or organizations with privacy constraints that prevent sending repository content to third-party providers. Unlike cloud-based APIs, local deployment requires the user to manage hardware resources, but grants full control over model execution and data handling.

The D.aise Web Interface. The primary interaction surface of `D.aise` is a web-based frontend built with Next.js, TypeScript, and Tailwind CSS (see Figure 1). The interface is organized around four views accessible from a left-hand sidebar: *Projects* (6), *Project Workspace* (3), *Prompt Lab* (4), and *LM Configuration*. The *Project Workspace* is the central page of the application. Its left panel (7) renders an interactive, collapsible file tree for the selected repository. The right panel (3) presents the project metadata form alongside three primary action cards: *Analyze Project*, *Create* `README`, and *Update* `README`. When a generation task completes, the interface renders the result in a dedicated output panel (2). The panel also shows the option of directly committing the generated documentation to the GitHub repository.

4 Methodology

This study aims to validate the usability and perceived usefulness of `D.aise` by conducting a structured experiment designed to simulate realistic usage scenarios and evaluate how potential users interact with the platform in practice. Given that the core motivation of `D.aise` is to lower the friction of software documentation by automating `README` generation and updates within a unified, LM-powered interface, we also evaluated how effectively `D.aise` in comparison with GitHub Copilot.

To achieve this, the study was organized into four main steps: (1) Preparation, (2) Execution, (3) Performance Comparison and (4) Data Analysis. These steps include the design and execution of targeted tasks as well as the application of a survey grounded in the Technology Acceptance Model (TAM) [6]. It enables us to capture nuanced insights into participants' interaction patterns and the tool's effectiveness in supporting typical documentation workflows. We will detail each step in the following sections.

(Step 1) Preparation. In the preparation phase, we formulated the structure of the experiment, defining the three categories of tasks used in our protocol [6]: *filtering*, *basic*, and *assimilation*. *Filtering tasks* were designed to evaluate whether participants had the minimum level of familiarity with the domain and the interface necessary to proceed with the study; these tasks involved basic identification of elements in the interface, such as locating specific fields or reading pre-populated project metadata. *Basic tasks* required participants to perform isolated operations, such as importing a repository, running LM analysis, or generating a `README` from scratch. *Assimilation tasks*, in turn, involved more complex challenges in which participants had to correlate multiple functionalities, such as evaluating the differences between an updated version of a `README`. The full list of experimental tasks is presented in Table 2. The tasks cover `D.aise`'s core functionalities such as repository import, LM-powered project analysis, `README` generation and update, and *Prompt Lab* interaction.

Participants were selected to represent individuals with professional experience with software development, reflecting the target audience of `D.aise`. Sessions were scheduled individually. An Informed Consent Form (ICF) was prepared to ensure that participants were fully aware of the nature and objectives of the study. This document detailed the objectives of the research, ensured anonymity, and clarified that screen and audio recordings would be used exclusively for research purposes.

Prior to the main sessions, two pilot studies were conducted to validate the task script and questionnaire. In this pilot section, we identified improvements in task formulations and session duration. The pilot sessions also allowed us to refine the environment setup procedure, including the selection of new repositories to be analyzed and creation of a checklist for preparing the environment

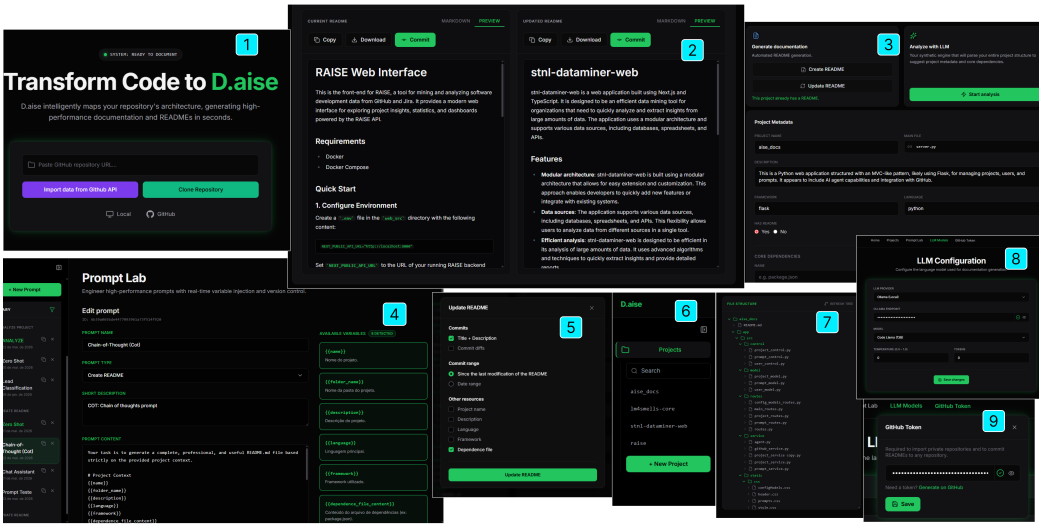

Figure 1: Screenshot showing the home page with the repository selection screen from our web interface.

Table 2: Structure of the experimental task script.

Filtering Tasks (FT)	
FT1	Which projects are available in the tool?
FT2	Access the `stnl-dataminer-web` project. What framework and programming language are used in this project?
FT3	How many prompt templates of type "Create README" are available in the tool?
FT4	Which LM provider is configured in the tool?
FT5	Which model is configured in the tool?
Basic Tasks (BT)	
BT1	Configure the tool to use the Code Llama (13B) model.
BT2	Does Prompt Lab currently have any template set as the default for README creation? Which one?
BT3	Duplicate the existing "Chain-of-Thought" template of type "Create README" and rename the copy to "Prompt Test". Then modify its content to use the "Role Prompt" technique, specifying that the LLM should behave as a README file creation specialist.
BT4	Import the `lm4smells-core` repository into the tool using the GitHub API import option.
BT5	For the `lm4smells-core` repository you just imported, run the project analysis with the LM and verify whether the framework and programming language were automatically updated after the analysis.
BT6	For the `stnl-dataminer-web` project, create a new README by selecting only the project name and tree information.
Assimilation Tasks (AT)	
AT1	Access the "Chain-of-Thought (CoT)" template of type "Create README". Identify what is missing, make the necessary changes, and save it.
AT2	You are a developer who has just joined a new team and has been assigned to work on the `D.aise` repository. The repository has no documentation, and you need to understand how to configure and run the project on your machine. Using the tool, generate a README for this repository and commit the generated README. Describe the steps you followed and how you evaluate the generated README.
AT3	Use the tool's README update feature to generate an updated version of the documentation for the `stnl-dataminer-web` repository. Update it based on the changes made since the last README update. Analyze the result and describe what changed compared to the previous README.

for the next experiment, *e.g.* changing the setup of the active LM model and prompt, and removing new features introduced by the participants. Thus, ensuring consistent and reproducible conditions across all participant sessions.

(Step 2) Execution. This step was designed to explore how developers interact with and evaluate `D.aise` through hands-on activities with the tool. The execution phase of the study was organized into three distinct stages:

(i) Participant Onboarding and Consent. Each session began with a formal introduction to the study. After introduction, participants were presented with the ICF and participation was conditioned on reading and agreeing with the ICF.

(ii) `D.aise` *Introduction and Task Execution.* Following consent, participants were shown a short introductory video providing an overview of `D.aise`'s interface and core functionalities. Then, participants began the experimental session, which involved completing the practical tasks described in Table 2 by interacting with the `D.aise` web interface. Tasks were presented one at a time by the researcher, and participants were instructed to follow the think-aloud protocol [18], verbalizing their reasoning and impressions while performing the tasks. All sessions were conducted in person and recorded using OBS Studio, capturing both screen activity and audio for subsequent transcription and qualitative analysis.

(iii) Post-Experiment Survey Questionnaire. After completing all tasks and providing verbal feedback, participants filled out a structured questionnaire designed to collect additional insights. This instrument combined scaled items grounded in the Technology Acceptance Model (TAM) [6] to evaluate perceived ease of use, perceived usefulness, and overall impressions of the tool. The structure of the questionnaire is presented in Table 3. It is composed of ten TAM-based questions (Q1–Q10), each rated on a 5-point Likert scale. Note that each question was followed by an optional open-ended question. It also included characterization questions to describe participant profiles and a final question for general feedback.

(Step 3) Performance Comparison. To further assess the performance of `D.aise`, we conducted a comparative evaluation against GitHub Copilot using the GPT-4.1 model. The goal of this stage was to analyze how different these approaches support automated `README` generation and maintenance under comparable conditions. The comparison focused on the generation and update of `README` files from source code repositories. For fairness, both tools were evaluated using the same set of repositories. In the GitHub Copilot condition, we used the GitHub Agentic Workflow configured with GPT-4.1 and used a predefined Rule Prompt designed to standardize the documentation generation process. For the `D.aise` condition, we used the platform configured with the Gemini-2.5-Flash model.

We analyzed the generated `README` files according to four quality dimensions: clarity, completeness, conciseness, and correctness

Table 3: Structure of the survey questionnaire

ID	Section	Question
Evaluation of the Tool		
Q1	TAM	Importing a GitHub repository was a simple and straightforward process.
Q2	TAM	The tool's Projects screen presents the repository information in a clear and understandable way.
Q3	TAM	The LM Analysis feature allowed me to identify the project characteristics without much effort.
Q4	TAM	The README generation process in the tool was clear and understandable.
Q5	TAM	The preview of the README generated by the tool was clear and allowed me to evaluate the result easily.
Q6	TAM	Navigating and editing templates in Prompt Lab was intuitive.
Q7	TAM	The tool makes it easier to create documentation for software repositories.
Q8	TAM	I was able to import, analyze, and generate documentation for a repository without major difficulties.
Q9	TAM	The tool contributes positively to productivity in software project documentation.
Q10	TAM	I would use this tool to generate or update the documentation for my projects.
Participant Characterization		
Q11	Charact.	What is your area of expertise?
Q12	Charact.	How many years of professional experience do you have in your field?
Q13	Charact.	What is your level of knowledge about GitHub?
Q14	Charact.	In which contexts have you used GitHub?
Q15	Charact.	Have you ever used any documentation generation tool?
Final Question		
Q16	Final	What do you think about this study and the tool?

(see [2] for the complete evaluation protocol and assessment criteria). We used five internal open-source projects developed and maintained by our research team, allowing us to leverage domain knowledge and technical familiarity to more accurately assess the quality and consistency of the generated documentation. In total, we selected sixty case study of Pull Requests (PRs) for manual labeling of quality. The complete information about the projects and the prompt used are available in our repository.

(Step 4) Data Analysis. This study was designed to evaluate `D.aise`'s usability in practice, to understand how users perceive the tool, and to assess the quality of the documentation it generates. To evaluate `D.aise`'s usability as experienced by participants, we analyzed the responses collected from 18 participants during the experimental sessions, including task execution observations, TAM-based questionnaire results, and qualitative feedback obtained through think-aloud interactions and post-session discussions. To evaluate `D.aise`'s performance, we used five open-source projects hosted on GitHub. We formulated three research questions (RQs):

RQ_1: How usable is `D.aise` *when performing tasks of varying complexity?* This question investigates how participants performed while executing the experimental tasks structured in Table 2. We analyzed participants' performance metrics, specifically task completion and time spent per section. In addition, a qualitative analysis was conducted based on the video recordings captured during the experiment sessions.

RQ_2: How do users perceive the usefulness of D.aise? This question captures participants' subjective evaluation of `D.aise` based on the structured post-session questionnaire (Table 3), grounded in TAM [6], together with open-ended responses collected for each question (Q1-Q10) and qualitative observations during task execution.

We aggregated the closed-ended responses from the survey questionnaire and calculated the average score for each question to identify general trends in participants' perceptions of the platform. These averages were then visualized in a bar chart mapping the average score of each item to its corresponding question identifier, providing a concise summary of the distribution of user feedback across the evaluated dimensions of `D.aise`. Additionally, the open-ended questions of the survey was used to capture participants' subjective impressions and suggestions for improvement.

RQ_3: How does `D.aise` *compare with GitHub Copilot in supporting automated* `README` *generation and maintenance?* This question investigates the differences between `D.aise` and GitHub Copilot regarding the quality of the generated documentation. For GitHub Copilot, we used the GitHub Agentic Workflow with GPT-4.1 model and a predefined Rule Prompt, while `D.aise` was configured with the Gemini-2.5-Flash model. We evaluated the resulting `README` files according to four quality dimensions: clarity, completeness, conciseness, and correctness.

5 Results and Discussion

We discuss the results of our research questions (RQs) as follows.

RQ_1: Evaluation Through Task Execution. Table 4 shows that most participants were able to successfully complete all experimental tasks, including filtering (FT), basic interactions (BT), and more cognitively demanding assimilation tasks (AT). FT and BT presented the highest completion consistency among participants, while AT required longer execution times and introduced slightly greater variability in performance due to their higher cognitive demands. The observed errors were typically related to misunderstanding interface feedback, overlooking intermediate steps, or uncertainty regarding how generated modifications should be applied to the repository. In a few cases, participants hesitated when switching between different sections of the platform. Despite this increase in complexity, most participants were still able to complete the tasks successfully, reinforcing the overall usability of the tool. This suggests that the tool supports an effective end-to-end workflow, allowing users to navigate repository information, generate `README` content, and interact with LM-based features without major barriers, even as task complexity increases.

RQ_2: Perceived Usefulness and User Experience. The results in Table 5 indicate a generally positive perception of `D.aise` regarding usability and applicability in `README` generation and update tasks. Participants provided constructive suggestions regarding workflow integration and flexibility. Some users indicated that the experience could be further improved through integration with existing development environments, such as IDE extensions or desktop applications, reducing context switching during development activities. Additional feedback revealed opportunities for improvement related to interface clarity and advanced interactions. A few participants reported difficulties understanding the Prompt Lab menu and navigating certain interface elements during more complex tasks. Users also highlighted the practical value of `D.aise` in reducing documentation effort through features such as prompt standardization, automated repository information extraction, and reductions in repetitive manual work, particularly in scenarios involving tight deadlines or large documentation workloads.

RQ_3: Comparative Evaluation with GitHub Copilot. The results indicate that both `D.aise` and GitHub Copilot were capable of generating useful `README` files, achieving similar average scores across the evaluated quality dimensions for five repositories and a total of sixty case studies (PRs). As illustrated in Figure 2, both approaches obtained comparable results for clarity, conciseness, and correctness. However, qualitative observations revealed important practical differences between both approaches. GitHub Copilot operates as a largely black-box solution in which the internal orchestration of prompts, repository inspection, and context selection

Table 4: Participant Performance Data.

Part.	Filtering Tasks						Basic Tasks							Assimilation Tasks			
	FT1	FT2	FT3	FT4	FT5	Time	BT1	BT2	BT3	BT4	BT5	BT6	Time	AT1	AT2	AT3	Time
P1	✓	✓	✓	✓	✓	2 min	✓	✓	✓	✓	✓	✓	9 min	✓	✓	✓	11 min
P2	✓	✓	✓	✓	✓	3 min	✓	✓	✓	✓	✓	✓	8 min	✓	✓	✓	6 min
P3	✓	✓	✓	✓	✓	3 min	✓	✓	✓	✓	✓	✓	9 min	✗	✓	✓	7 min
P4	✓	✓	✗	✓	✓	2 min	✓	✗	✓	✓	✓	✓	8 min	✓	✓	✓	12 min
P5	✓	✗	✗	✓	✓	4 min	✓	✗	✓	✗	✓	✓	9 min	✗	✓	✓	8 min
P6	✓	✓	✓	✓	✓	3 min	✓	✓	✓	✓	✓	✓	7 min	✓	✓	✓	20 min
P7	✓	✗	✓	✓	✓	5 min	✓	✗	✓	✓	✓	✓	11 min	✓	✓	✓	10 min
P8	✓	✓	✓	✓	✓	2 min	✓	✓	✓	✓	✓	✓	8 min	✓	✓	✓	13 min
P9	✓	✓	✓	✓	✓	1.5 min	✓	✓	✓	✓	✓	✓	5 min	✓	✓	✓	5 min
P10	✓	✓	✓	✓	✓	2 min	✓	✓	✓	✓	✓	✓	10 min	✗	✓	✓	9 min
P11	✓	✓	✓	✓	✓	1.5 min	✓	✓	✓	✓	✓	✓	7 min	✗	✓	✓	14 min
P12	✓	✓	✓	✓	✓	2 min	✓	✓	✓	✓	✓	✗	9 min	✗	✓	✓	13 min
P13	✓	✓	✓	✓	✓	2 min	✓	✓	✓	✓	✓	✓	23 min	✓	✓	✓	5 min
P14	✓	✓	✓	✓	✓	2 min	✓	✓	✗	✓	✓	✓	6 min	✓	✓	✓	8 min
P15	✓	✓	✓	✓	✓	3 min	✓	✓	✓	✓	✓	✓	9 min	✓	✓	✓	5 min
P16	✓	✓	✓	✓	✓	3 min	✓	✗	✓	✓	✓	✓	7 min	✓	✓	✓	7 min
P17	✓	✓	✓	✓	✓	3 min	✓	✓	✓	✓	✓	✓	7 min	✓	✓	✓	11 min
P18	✓	✓	✓	✓	✓	2 min	✓	✓	✓	✓	✓	✓	6 min	✓	✓	✓	9 min

Table 5: Participants' responses in the usability study.

#P	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
P1	5	5	5	5	5	5	5	5	5	5
P2	3	3	3	3	3	3	3	3	3	3
P3	5	5	5	5	5	2	4	5	4	4
P4	5	5	5	5	5	3	5	5	5	5
P5	4	4	5	5	5	4	5	4	5	5
P6	5	4	5	5	5	5	5	5	5	5
P7	5	4	5	5	5	4	5	5	5	5
P8	5	5	4	4	5	5	5	5	5	5
P9	5	3	3	5	5	5	3	4	3	2
P10	5	5	5	5	5	2	5	5	5	2
P11	5	4	5	5	3	5	5	5	4	5
P12	5	4	3	5	5	4	4	5	4	5
P13	5	5	5	5	5	5	5	5	3	2
P14	5	4	5	4	5	4	5	5	5	4
P15	5	5	5	4	5	5	5	5	5	5
P16	5	5	5	5	2	5	5	5	5	5
P17	5	5	5	5	5	4	4	5	4	5
P18	5	5	5	5	5	3	5	5	5	5

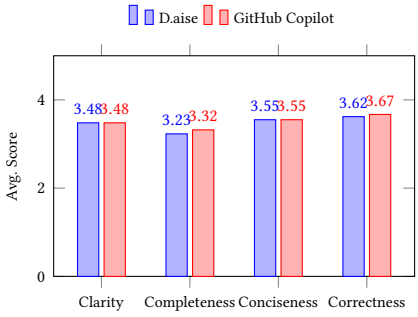

Figure 2: Comparison between `D.aise` and GitHub Copilot.

is not exposed to developers. During the experiments, we observed that changes introduced by updates in the GitHub Agentic Workflow pipeline occasionally affected the stability and reproducibility of the documentation generation process. In some cases, modifications in the workflow behavior produced inconsistent outputs or broke previously functioning configurations, limiting the level of control researchers and developers have over the generation pipeline. This lack of transparency makes it difficult to isolate the impact of prompt engineering decisions, reproduce experiments consistently, or customize intermediate stages of the documentation workflow. In contrast, `D.aise` was designed as a controllable and extensible documentation environment in which the repository analysis process, prompt composition strategy, LM configuration, and generated artifacts can be directly inspected and modified by users. This flexibility is particularly important for research scenarios involving prompt experimentation, model comparison, and controlled evaluations. As next step, we plan to evaluate other models and prompts.

6 Conclusion and Future Work

This paper introduced `D.aise`, a platform for generation and prompt-based experimentation that integrates repository data extraction and management capabilities. The system allows users to access and clone GitHub repositories directly. Overall, `D.aise` aims to assist developers in both generating and updating files and exploring different strategies of prompt engineering.

As future work, the platform can be extended to support additional software hosting services beyond GitHub, such as GitLab and Azure DevOps, increasing its applicability in multi-platform development environments. This would enhance its flexibility and enable broader comparative studies across different repository ecosystems.

Another possible extension is the support for generating additional types of software documentation, such as changelogs and release notes. This would expand the scope of the tool beyond generation, enabling more comprehensive documentation support throughout the software lifecycle. Together, these improvements would expand `D.aise`'s scope from generation to a more general platform for software documentation support. Future work also includes more comprehensive comparisons between `D.aise` and AI-assisted software development tools, such as GitHub Copilot, as well as emerging tool- and agent-based systems.

Artifact Availability

The source code for `D.aise` is available at github.com/aisepucrio/D.aise and https://doi.org/10.5281/zenodo.21464226 under the MIT license.

Acknowledgments

This research was partially funded by the Brazilian funding agencies CAPES/COFECUB (Grant 88881.879016/2023-01 and Ma1036/24), CNPq (Grant 406089/2025-6), FAPESP (Grant 2023/00811-0), and FAPEMIG (Grant APQ-01488-24). We also acknowledge the Brazilian company Stone[3] for their financial support.

[3]https://www.stone.com.br/

References

[1] 2019. Software Documentation Issues Unveiled. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)* (Montreal, QC, Canada). IEEE. doi:10.1109/ICSE.2019.00122

[2] Arthur Alesi, Gabriela Soares, and Juliana Alves Pereira. 2026. *D.aise: An Automated README Generator from Source Code Repositories.* doi:10.5281/zenodo.21464226

[3] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2024. Why People Contribute Software Documentation. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE '24).* ACM. doi:10.1145/3641822.3641881

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS 2020).* 1877–1901.

[5] Sayak Chakrabarty and Souradip Pal. 2025. ReadmeReady: Free and Customizable Code Documentation with LLMs - A Fine-Tuning Approach. *Journal of Open Source Software* 10, 108 (April 2025), 7489. doi:10.21105/joss.07489

[6] Fred D Davis. 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly* 13, 3 (1989), 319–340.

[7] Shubhang Shekhar Dvivedi, Vyshnav Vijay, Sai Leela Rahul Pujari, Shoumik Lodh, and Dhruv Kumar. 2024. A comparative analysis of large language models for code documentation generation. In *Proceedings of the 1st ACM international conference on AI-powered software.* 65–73.

[8] Csaba Nagy Mario Linares-Vásquez Laura Moreno Gabriele Bavota MicheleLanza David C. Shepherd Emad Aghajani. 2020. Software Documentation: The Practitioners' Perspective. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE).* ACM. doi:10.1145/3377811.3380405

[9] GitHub. 2024. GitHub REST API Documentation. https://docs.github.com/en/rest. Accessed: 2025.

[10] Google LLC. 2024. Gemini API Documentation. https://ai.google.dev/docs. Accessed: 2024.

[11] Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2024. Pre-Trained Language Models for Text Generation: A Survey. *Comput. Surveys* 56, 9, Article 230 (2024), 39 pages. doi:10.1145/3649449

[12] Ollama. 2024. Ollama: Get Up and Running with Large Language Models Locally. https://ollama.com. Accessed: 2024.

[13] Patched.codes. 2024. Generate README Eval: A Benchmark for Evaluating LLMs on README Generation. https://huggingface.co/datasets/patched-codes/generate-readme-eval. Dataset and benchmark hosted on Hugging Face. Accessed: 2025.

[14] Jatinderkumar R. Saini et al. 2015. Significance of Software Documentation in Software Development Process. In *Proceedings of the International Conference on Advanced Computing and Communication Technologies.*

[15] Sebastian Thiel et al. 2024. GitPython: A Python Library for Git Repository Access. https://github.com/gitpython-developers/GitPython. Accessed: 2024.

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* (2023). doi:10.48550/arXiv.2302.13971

[17] Tianlei Wang, Shaowei Wang, and Tse-Hsun (Peter) Chen. 2023. Study the Correlation Between the Readme File of GitHub Projects and Their Popularity. *Journal of Systems and Software* (2023). doi:10.1016/j.jss.2023.111806

[18] Michael D. Wolcott and Nikki G. Lobczowski. 2021. Using Cognitive Interviews and Think-Aloud Protocols to Understand Thought Processes. *Currents in Pharmacy Teaching and Learning* (2021). doi:10.1016/j.cptl.2020.09.005

[19] Junji Zhi, Vahid Garousi-Yusifoğlu, Bo Sun, Golara Garousi, Shawn Shahnewaz, and Guenther Ruhe. 2015. Cost, benefits and quality of software development documentation. *Journal of Systems and Software* (2015). doi:10.1016/j.jss.2014.09.042