



An LLM Assistant for Software Project Onboarding

Knud Ronau Larsen
Mid Sweden University
Östersund, Sweden
knudronau@gmail.com

Magnus Edvall
Mid Sweden University
Östersund, Sweden
maed2104@student.miun.se

Truong Ho-Quang
Mid Sweden University
Göteborg, Sweden
truong.ho-quang@miun.se

Felix Dobslaw
Mid Sweden University
Östersund, Sweden
felix.dobslaw@miun.se

Rodi Jolak
Mid Sweden University
Göteborg, Sweden
rodi.jolak@miun.se

ABSTRACT

New developers commonly join software teams for projects in the advanced stages of development. They frequently face barriers, especially in software comprehension, that impede their ability to make early contributions. To aid newcomers in understanding software projects, we have developed an LLM-based tool called SPAC-B that can answer software project-specific questions. In this study, a case study is conducted to investigate the accuracy of SPAC-B's answers to common developer questions when resolving issues on two open-source projects regarding relevance, completeness, and correctness. On a Likert scale (1-5), answers from SPAC-B scored a mean of 4.6 in relevance and 4.3 in completeness and correctness. An experiment with ten software developers is performed to further examine SPAC-B's ability to help newcomers formulate plans for resolving real-life open-source issues. Results show that the participants could make a better implementation plan with the use of SPAC-B and 8/10 participants found the tool to be helpful.

CCS CONCEPTS

• **General and reference** → **Evaluation; Experimentation; • Software and its engineering** → **Software implementation planning; Maintaining software; • Computing methodologies** → **Knowledge representation and reasoning; Natural language processing; • Information systems** → *Language models; Question answering.*

KEYWORDS

Large Language Models, Retrieval-augmented Generation, Software Development, Onboarding

ACM Reference Format:

Knud Ronau Larsen, Magnus Edvall, Truong Ho-Quang, Felix Dobslaw, and Rodi Jolak. 2025. An LLM Assistant for Software Project Onboarding. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25)*, June 23–28, 2025, Trondheim, Norway. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3696630.3728719>



This work is licensed under a Creative Commons Attribution 4.0 International License. *FSE Companion '25*, June 23–28, 2025, Trondheim, Norway
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1276-0/2025/06.
<https://doi.org/10.1145/3696630.3728719>

1 INTRODUCTION

In software projects, new developers often board in later stages when code bases have grown large over time. These newcomers¹ must quickly grasp the technical and technological aspects of the ongoing project [5]. According to Steinmacher et al. [20], newcomers in open-source software projects will often face a variety of challenges, including unclear, outdated, or lacking documentation, difficulty in finding an appropriate task, software architecture complexity, lack of expertise, and untimely or improper answers from other developers. Furthermore, Matturo et al. [13] have found that newcomers in commercial software projects express that the most frequently experienced difficulties are related to documentation and understanding the solution / knowing the product. These findings underline that understanding software projects is a core issue newcomers face in commercial and open-source development.

In both open-source and commercial development, these barriers have negative consequences. Newcomers must overcome the learning curve of understanding the software project, which impedes their ability to make early contributions. In commercial development, this causes a discrepancy between companies and newcomers, as companies expect contributions earlier than possible for the newcomer [16]. In open-source development, the barriers can ultimately cause newcomers to give up on a project [21]. This has significant consequences as it is vital for open-source software project communities to keep a continuous influx of newcomers and maintain the core developers to remain sustainable [15, 19].

In November 2022, OpenAI revolutionized the artificial intelligence (AI) industry when they released their chatbot ChatGPT to the public [18] and started the AI gold rush. Since then, Generative AI has shown the potential to improve software development productivity in code generation, test case generation, enhancing creativity, summarizing documentation, problem-solving, efficient development, maintaining legacy, and improving software quality. It is becoming increasingly utilized in software development, where large language model (LLM) based tools can serve as powerful assistants, offering a range of capabilities that support software development [6]. Although these results from this relatively young technology are valuable for developers, it has not yet been widely adopted to assist developers in software project understanding.

Under the right circumstances, large language models (LLM) such as OpenAI's ChatGPT have remarkable performance but are limited to the knowledge acquired during their training. This makes

¹Throughout, we write *newcomer* not to mean novice, but simply new to the project.

them impractical in knowledge-specific areas, such as questions about a specific software project. This paper aims to investigate this research gap to discover if Generative AI can assist newcomers' understanding of software projects.

In this study, we introduce Software Project Artificial Consultation Bot (SPAC-B), a tool powered by retrieval augmented generation (RAG), created to enhance an LLM with the knowledge of a given software project, thus allowing a newcomer to ask project-specific questions. SPAC-B aims to assist newcomers by enabling them to ask detailed questions about a specific software project, thereby improving their understanding and reducing the barriers to effective contribution.

The scope of this study is limited to the topic of using Generative AI to assist with understanding software projects, which are investigated through the capabilities of SPAC-B. The focal point of this investigation is centred around understanding software; therefore, LLM capabilities such as code generation, test case generation, improving software quality, and maintaining legacy code fall outside this study's scope. We use the two following research questions to construct our study:

RQ1 *How accurate are the answers SPAC-B delivers regarding a given software project/repository?*

This research question aims at assessing the reliability of SPAC-B from a developer's perspective by evaluating its answer accuracy.

RQ2 *Does using SPAC-B help newcomers formulate plans for solving issues in software projects?*

Addressing this research question involves having newcomers outline plans for real software development tasks with and without assistance from SPAC-B. This approach enables the evaluation of how developers interact with so-called Generative AI assistants, the assessment of the usability of SPAC-B, and the gathering of feedback on the tool from participants. Additionally, the experiment will provide initial insights into whether SPAC-B can enhance newcomers' understanding of software projects.

2 RELATED WORKS

Belszner et al. [1] investigate the prospects and challenges of adopting LLMs in software development. They discuss how LLMs can support classical phases of software development, including Requirements Engineering, System Design, Code Generation and Quality Assurance, Testing, and Verification. To follow up, they conducted a minor case study, testing ChatGPT and Bard in these aspects, where they observed interesting and promising but inconclusive results. Finally, they address the current challenges in adopting LLMs for software development and put forward three scenarios if the obstacles were to be alleviated: A Better Bat, A Game Changer, and An Entirely New Game.

Wong et al. [22] divide AI-assisted programming tasks into two main categories: *Generation*, which includes code generation, code completion, code translation, code refinement, and code summarization. *Understanding*, which includes defect detection and clone detection. Thus, while this study addresses code understanding, it does so from the perspective of AI-assistant tools and does not address how AI-assistant tools can assist developers in understanding software projects. Lastly, they address current challenges and opportunities in AI-assisted programming, which corresponds with the findings from Belzner et al. [1].

In 2023, Fan et al. [7] investigated the role of LLMs in software engineering. The authors surveyed LLM use cases in software engineering and the open problems relating to these use cases. The survey shows that LLMs usage are most thoroughly explored in the areas of *Software Testing*, *Documentation Generation*, *Software Analytics*, *Human Computer Interaction*, *Software Engineering Process*, and *SE Education*. While the survey is extensive, it overlooks the usage of LLMs to assist in understanding software projects.

Liang et al. [12] conducted a large-scale usage-oriented survey comprising 410 developers to understand their practices using AI-assistant tools. Participants reported several successful use cases, with the most prevalent being simple code generation, such as auto-completion to reduce keystrokes and implementing repetitive code or simple logic. However, a noteworthy finding is that numerous participants reported *Learning* as a successful use case, where participants utilized the tools to assist when exposed to new programming languages or libraries instead of online documentation or video tutorials. An additional noteworthy finding is that the second most desired improvement of AI assistants is for them to *have an additional understanding of code context*.

Samia Kabir et al. [8] conducted an empirical study analyzing ChatGPT's answers to Stack Overflow programming questions. Their results from manually analyzing these answers show that 52% contain incorrect information, 78% are inconsistent with human answers, 35% lack comprehensiveness, and 77% contain redundant, irrelevant, or unnecessary information. Despite this level of incorrectness, the user study conducted in this work shows that participants preferred ChatGPT answers 35% of the time. This indicates an impressive willingness to adapt to and use this new technology despite its subpar performance.

Nam et al. [14] built a plugin for the IDE VS Code called GILT that utilizes OpenAI's GPT-3.5-turbo to explain a highlighted section of code, provide details of API calls used in the code, explain key domain-specific terms, and provide usage examples for an API. A user study with 32 participants found that GILT aids in task completion more than web search. While the aim of the study is fundamentally similar to ours, there are key differences in the approaches. Unlike SPAC-B, GILT does not focus on user prompts and does not utilize RAG to find relevant context. GILT explains highlighted code and, therefore, focuses on code understanding and not understanding software projects as a whole.

It is evident from the summary of related works that AI-assisted software development has become a widely researched topic. However, it is equally apparent that research regarding using AI assistance for developer understanding is lacking, with only a few related works mentioning this topic. Similar to the work of Ross et al. [17], SPAC-B was developed to evaluate the effects of generative AI in software development, but in contrast, focus solely on its impact on assisting newcomers' understanding of software projects. Furthermore, SPAC-B is developed using RAG, which has been scarcely researched concerning software development assistance, as outlined in the first section.

3 RESEARCH METHODOLOGY

In this section, the methodology of our research is presented. Figure 1 presents each step in the methodology process.

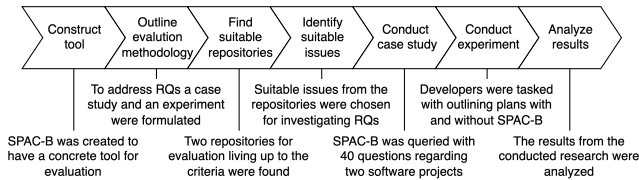


Figure 1: An overview of the overall research process.

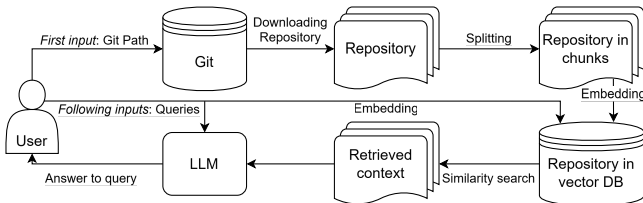


Figure 2: SPAC-B's logical processes and user interactions.

3.1 Tool construction

The source code of SPAC-B is publicly available on GitHub [10]. It requires an API key from OpenAI² to run. Figure 2 illustrates how SPAC-B functions from a user's perspective. After inputting a Git repository path, SPAC-B loads the source code utilizing GitLoader[3]. It splits the downloaded data into 1600 character chunks with 200 character overlap using RecursiveCharacterTextSplitter[4], where *separators* is a list of language syntax from most common programming languages. It embeds these with text-embedding-3-large³, which as of writing is the best performing embedding model, and stores it in a FAISS[2] vector database. The relevant context is pulled out of the vector database through a similarity search when the user enters a query, which is also embedded with text-embedding-3-large. The query and context are provided to the LLM, which outputs an answer displayed for the user. The GUI consists of a large text area, where responses are presented, with two text input fields below for users to input queries and repository path. It also has components for users to choose the branch, LLM temperature, and whether the question is related to testing, all marked with blue arrows in Figure 3. Any questions can be asked, as SPAC-B is implemented to use the 32 most relevant chunks, a number chosen to increase chances of finding the correct context without risking confusing the model and stay within GPT-4 Turbo's context window.³ The performance of SPAC-B heavily relies on the underlying LLM regarding both response quality and response time, as the response time impact of SPAC-B's RAG implementation is negligible.

3.2 Select suitable open source repositories for the study

The repositories are selected from GitHub using following criteria:

- Size - Should be small to medium.
- Status - Should be active with recent contributions.
- Popularity - Should be a popular project based on the number of stars in GitHub.

²<https://openai.com/blog/openai-api>

³<https://platform.openai.com/docs/models/overview>

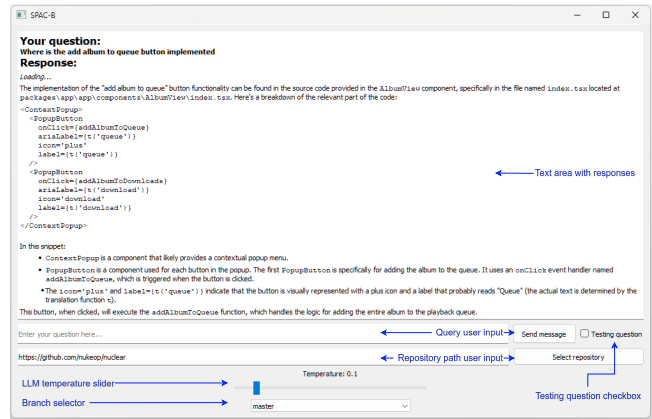


Figure 3: The graphical interface for SPAC-B.

- Number of contributors - Should be developed by a large pool of contributors.
- Primary programming language - Should be a popular language such as Java, JavaScript/TypeScript, or Python.
- No academic/study projects.
- Popular project domain - Should be easy to understand, to ease the need of researching the project domain, e.g. a computer game, and not something technical such as a framework.

The two last criteria are essential for the repository chosen for the experiment addressing RQ2 to limit the effort spent by participants on understanding the project domain and allow them to focus on solving the issue. Specifically, issues labelled as a **good first issue** are preferred, indicating they are appropriate for newcomers. Furthermore, recently closed issues are prioritized in this study, as the accepted pull requests offer an outline of the correct implementation, which are used as a baseline when evaluating both SPAC-B answers in RQ1 and participant plans in RQ2.

Based on the list of criteria, the two selected repositories are Nuclear⁴, utilized for both the case study addressing RQ1 and the experiment addressing RQ2, and JabRef⁵, which is used exclusively for the case study. Both repositories in this study are found on Up-For-Grabs⁶, a website dedicated to open-source projects looking for new contributors. Nuclear is a desktop music player for streaming from free sources. JabRef is a reference management software that uses BibTeX and BibLaTeX as its native formats. Nuclear is written in TypeScript and JavaScript as the main programming languages whereas JabRef is primarily written in Java.

3.3 Case study: Evaluating accuracy (RQ1)

To evaluate the accuracy of SPAC-B, a case study is conducted in which SPAC-B's responses to 40 queries were analyzed through three metrics: *relevance*, *completeness*, and *correctness*.

3.3.1 Data collection. Five closed issues were chosen from each of the two open-source repositories. All selected issues are related to either implementing a new feature or improving on a current

⁴<https://github.com/nukeop/nuclear>

⁵<https://github.com/JabRef/jabref>

⁶<https://up-for-grabs.net>

feature. SPAC-B was queried with four questions per issue, adding up to 40 questions. The questions were formed by the authors from a newcomer’s perspective working on developing and maintaining an application. For each issue, the same structure of questions was asked: two questions regarding the location of the parts relevant to the issue (feature location) and two questions on the implementation of the desired issue. For the latter two questions, the first does not rely on any of the information gathered from the first two questions to investigate SPAC-B’s implementation plans when queried from the point of view of a complete newcomer. The second question utilizes the information collected from the former responses to create an informed question to evaluate if this impacts proposed implementation plans. Below is an exhibit of the questions following this structure that were asked regarding one of the chosen issues⁷:

- *Where is the button to clear queue implemented?*
- *Where is the list with queued tracks implemented?*
- *How can I implement a button below “save as playlist” to add the queue to an existing playlist?*
- *How can I implement a Dropdown in QueueMenuMore/index.tsx that adds queueItems to an existing playlist?*

3.3.2 Grading schema creation. The responses SPAC-B produced are evaluated on relevance, completeness, and correctness and graded in each metric. These evaluation metrics were inspired by Samia Kabir et al.’s [8] work and updated to fit this case study. The authors jointly created this grading schema and graded the data collected from this case study.

Relevance was evaluated based on the amount of relevant information in the response to the query, graded on a scale from 1 (completely irrelevant) to 5 (completely relevant).

Completeness was evaluated based on the amount of helpful information included in the response, graded on a scale from 1 (contains no useful information) to 5 (consists of all useful information).

Correctness was evaluated based on the amount of correct information present in the response, graded on a scale from 1 (entirely incorrect) to 5 (entirely correct). The correctness of implementation-related questions was compared to the implementations submitted in the merged pull requests.

3.4 Experiment: Evaluating impact (RQ2)

To evaluate the impact of SPAC-B, an experiment was conducted in which ten developers had to outline implementation plans with and without SPAC-B, where the plans were graded in *feature location* and *implementation* and the interaction with SPAC-B was analyzed.

3.4.1 Task design. To investigate RQ2, an experiment was designed where participants were presented with an open-source software project and asked to create a plan for implementing two issues, one with SPAC-B and one without SPAC-B. The ideal complexity of these issues was where most, but not all, participants could outline a plan given a 30-minute time frame. Furthermore, the desired outcome of implementing the issue should be easy to understand, as participants should not be hindered due to difficulties understanding the product or the desired feature. To conduct the experiment,

⁷<https://github.com/nukeop/nuclear/issues/1577>

Table 1: Experiment grouping

Experiment grouping	SPAC-B on 1st task	SPAC-B on 2nd task
Starts with issue 1	Group 1	Group 2
Starts with issue 2	Group 3	Group 4

ten software developers with at least one year of professional experience were recruited.

The experiments were conducted with the following structure:

Step 1: Before the experiment started, the participants were sent an introductory video shortly explaining the purpose of the study, the functionality of the open-source project, how to use SPAC-B and the structure and time frame of the experiment.

Step 2: At the experiment’s start, participants were asked to provide basic information about their experience and give informed consent per the guidelines from Amy J. Ko et al. [9].

Step 3: The participants were provided with a document explaining their first task and informed if they would have SPAC-B available. They then spent 30 minutes creating a plan for how they would implement the issue. Specifically, they were tasked with identifying the files that needed to be modified and what needed to be implemented in each file, including a simple outline of the logic, if possible. The participants were asked to think aloud to provide insight into their planning strategies.

Step 4: The participants were provided with a document explaining their second task, which was completed similarly to step 3. Participants who did not have SPAC-B available in step 3, had SPAC-B available for this task.

Step 5: The participants were asked to complete a questionnaire concerning their experience using SPAC-B.

Since the two tasks were not necessarily completely equal in difficulty, half of the participants used SPAC-B for one task, and the other half used SPAC-B for the other task. Furthermore, the order in which the two tasks are done could also affect the experiment’s outcome, so the participants were divided into four groups to mitigate validity threats, as shown in Table 1. To ensure a working environment for all participants, the experiments were run on our computer, which the participants accessed through TeamViewer⁸.

3.4.2 Project and issues. Nuclear’s interface is similar to popular commercial music players such as Spotify and Apple Music, making it ideal for this experiment, as participants are expected to understand the program without trouble.

Issue 1 is to add the functionality of adding an album to a playlist. From an album, users could previously only add individual songs to a playlist. The issue has been resolved and merged into the master branch. Therefore, it was necessary to fork a previous version of the master branch used for the experiment. The merged plan serves as a reference for correctly implementing the issue.

Issue 2 is to add a sleep timer to the music player. Users should be able to set a timer in the settings, after which the music should stop playing. This issue has a solution that was never merged into the master branch due to feedback from the creator of Nuclear. After contacting the creator, a reference plan for correct implementation was created.

⁸<https://www.teamviewer.com/>

3.4.3 Grading schema and process. To assess the impact of SPAC-B, participants' implementation plans were graded on two metrics: how well they identified the files that needed to be modified to implement the issue and how well their proposed changes corresponded with the reference plan. Both metrics are graded on a scale from 1 to 5. Table 2 the criteria for achieving each score. The authors jointly created this grading schema and graded the data collected from these experiments.

3.4.4 Facilitating the experiments. As experiment facilitators, two of the authors ensured that all participants had viewed the experiment introduction presentation and understood the content. The facilitators asked for consent to record the meeting and aided the participants in connecting to the experiment computer, getting oriented, reading and understanding the task. Finally, the facilitators asked if they had any questions before the experiment.

When an experiment began, the facilitators set a timer for 30 minutes and reminded the participants when they had 15, 5, and 1 minute left to complete the task. The participants often had some technical questions about the tool, which the facilitators would answer without going into the content of the task itself.

To evaluate the experience of using SPAC-B in a practical scenario, the think-aloud method was used throughout the experiments to get insights into the participants' thought processes, decision-making strategies, and overall interactions with the system. When the facilitators noticed that a participant was not following the think-aloud method, they would remind them to voice their thought process aloud.

The experiments were recorded and notes were taken for analysis, describing how the experiments progressed, with a focus on interaction with SPAC-B. In addition, participants were asked to fill out a questionnaire regarding SPAC-B after the experiments. This contains five open-ended questions: *How would you describe your query strategy?*, *What was your impression of the answers SPAC-B provided?*, *What did you expect from SPAC-B?*, *How would you like to have SPAC-B improved?*, and *What are your overall thoughts about using SPAC-B?*.

3.4.5 Analysis. Due to the small sample size of the experiment, the analysis mainly relied on the participants' answers in the questionnaire and the recordings. These allowed us to get a better understanding of the participant's thoughts about the tool and why certain participants found the tool more helpful than others. The participants' implementation plans were gathered into a document and analyzed according to Table 2. Each plan was evaluated based on its similarity to the reference plan outlined in the project and issues section. The scores of each plan were then summarized in a table, where the mean score and standard deviation were calculated.

4 RESULTS & DISCUSSION

In this section, the results of the conducted research are presented, followed by in-depth discussion.

4.1 Case study: Evaluating accuracy

As presented in the methodology section, a case study of 40 queries over 10 issues has been performed. Each response has been analyzed in three metrics: relevance, completeness, and correctness, and graded on a scale from 1 to 5. All queries, responses, and gradings are available in a complete schema at Zenodo [11].

The distribution of Scores for each metric in the responses received for all questions is shown in Table 3. On average, the responses scored 4.6 in relevance, 4.3 in completeness, and 4.3 in correctness, with a mean of 4.4 across all metrics.

The mean score of the two types of questions, feature location and implementation, along with all standard deviations, is displayed in Table 3. Specifically, SPAC-B's response to feature location questions scored a mean value of 4.7 across all metrics, 0.6 higher than SPAC-B's mean score towards implementation questions. An independent two-sample t-test shows this difference is significant, $t(118) = 3.13$, $p = 0.002$.

There are multiple reasons SPAC-B could perform better in feature location tasks than implementation questions. Firstly, implementing a new feature is generally a more complicated task than locating an existing feature in a software project. Secondly, there might be many different ways of implementing a new feature, but the grading of implementation responses was based on the reference plans (fixes for the closed issues). Furthermore, the correctness and completeness have much to do with the practices commonly used in software projects to achieve a certain quality level. There might be specific design patterns and principles that development should follow. This documentation data might not be found in the similarity search, and the LLM might, therefore, not adhere to these patterns and principles.

The implementation plans SPAC-B created from uninformed queries scored, on average, 3.8 across all metrics. The informed queries, where information from the first SPAC-B responses was utilized, gave implementation plans that, on average, scored 4.4 across all metrics, resulting in an improvement of 0.6 compared to plans created from uninformed queries. This indicates that SPAC-B can produce accurate implementation plans if queried appropriately. This underlines the importance of proper prompting techniques when using LLM-based tools.

As LLMs have been improving rapidly, the accuracy is expected to improve simply by modifying the API call in SPAC-B when new models are available. New models could be provided with more context and generate more sophisticated code for the implementation questions. As the training data of LLMs grows more extensive, they will have more knowledge about typical software architecture, design patterns, and standard practices. Therefore, it may produce higher-quality solutions.

4.2 Experiments: Evaluating SPAC-B's impact

From the experiments where SPAC-B was utilized in a practical scenario, 20 implementation plans were submitted for the two issues from the 10 participants. These plans were graded per the scoring scale in Table 2, resulting in 40 scores. All experiments were recorded for later analysis, along with the full notes describing the progress of the experiments, which can be found at Zenodo [11].

4.2.1 Plan evaluations. The assessments of the participants' plans are presented in Table 4, along with short descriptions of how the experiments progressed and how the participants assessed SPAC-B. On average, plans made by participants without assistance from SPAC-B scored 2.1 in feature location and 1.7 in implementation, a combined mean score of 1.9. Plans made by participants with the assistance of SPAC-B on average scored 2.8 in feature location and

Table 2: Grading scale for participants' proposed implementation plans

Metric	File identification	Implementation
Score 1	No correct files were identified	Nothing is correct
Score 2	One correctly identified file, multiple mistakes	Severely lacking implementation plan
Score 3	One identified file with no mistakes, or at least two with multiple mistakes	Correct but incomplete, vague, or with several mistakes.
Score 4	One missing file or one mistake	Correct answer with minor mistakes
Score 5	All the correct files and only correct files were identified	Everything is correct

Table 3: Case Study Mean Scores and Standard Deviation (SD)

Metric	Feat. Loc.		Impl.		Total	
	Mean	SD	Mean	SD	Mean	SD
Relevance	4.65	0.83	4.55	0.99	4.60	0.90
Completeness	4.60	1.08	4.00	0.88	4.30	1.02
Correctness	4.70	0.88	3.85	0.66	4.28	0.88
Total	4.65	0.96	4.13	0.84	4.39	0.93

2.6 in implementation, resulting in a combined mean of 2.7, a mean value of **0.8 higher with assistance from SPAC-B than without**. The implementation plans were most affected by the assistance of SPAC-B with an average increase of **0.9**.

The plans from participants who reported no prior experience with AI tools for programming scored **0.7** higher on average when SPAC-B was available. For participants with AI experience, the improvement was slightly higher at **0.9** when assisted by SPAC-B.

4.2.2 The different approaches with and without SPAC-B. The participants essentially had similar approaches when completing their tasks without assistance from SPAC-B. They tried to identify where the feature should be added to the application's GUI. They then tried to identify the surrounding components in the editor by simply looking through the repository's folders or searching for the surrounding components' names. However, this approach was time-consuming and tedious as the file structure was difficult for the participants to navigate, and the search functionality had drawbacks, such as leading to language files and finding numerous entities with the same name. Only two out of ten participants succeeded with this approach and found the appropriate files for handling logic by following this lead.

When participants had SPAC-B available, most also started by trying to locate the relevant feature in the GUI. They then asked SPAC-B where the surrounding GUI components were implemented and where their logic was implemented instead of searching through the IDE. Generally, the participants had more success with this approach, especially since SPAC-B often answered with pointers to the logic, resulting in participants identifying the files quicker, thus having more time to plan implementation logic. Some participants continued by asking for an implementation plan for the desired feature, thus utilizing SPAC-B to outline the desired implementation.

Participants P1 and P2, who had previously reported using AI, such as ChatGPT and CoPilot, differed in their approach. They started by asking elaborate questions to SPAC-B to see if that could create the whole plan for them. P1 received a misleading response,

as it was based on context from testing files instead of source code, but trusted it without verifying it, resulting in a plan based solely on misleading information. P2 got an essentially correct response, looked through the code to verify the answer, and asked follow-up questions to gather more information, resulting in a much better plan than without SPAC-B. P6 mistook a playlist for an album in the application and asked SPAC-B about the wrong components, an obstacle that was not anticipated. This resulted in confusion and nonsensical answers from the participants.

Apart from P1 and P2, P3, P5 and P8 also reported using AI for programming, while P4, P6, P7, P9 and P10 reported no experience with AI assistance.

4.2.3 The factors impacting participant's scores & performance. It is noticeable that there are significant differences in participant's proficiency in utilizing SPAC-B. As participants were asked about their previous experience with AI tools for programming, it was evaluated if this impacted participants' proficiency utilizing SPAC-B. 5 out of 10 participants reported having at least some experience with AI tools, all of which asked implementation questions. Comparatively, of the 5 without experience with AI tools, only one participant asked a single implementation question. Thus, it was observed that "*all participants with previous AI experience asked implementation questions, and only one participant without previous experience did*". Regardless of which factors affected the proficiency in utilizing SPAC-B, the differences could have been mitigated with a more extensive introduction to SPAC-B and its capabilities. However, it was decided to provide a minimal introduction, showcasing only one feature location-related question, to evaluate how participants would interact with SPAC-B after this minimal presentation. **The conclusion is that AI assistant tools require a thorough introduction to assist all users.**

Three participants, P6, P7, and P8, proposed plans with assistance from SPAC-B that scored **1** (low) in both feature location and implementation. As previously mentioned, P6's challenge largely revolved around mistaking a playlist for an album, thus asking SPAC-B the wrong questions. P8's challenges stemmed from overshooting the complexity of the tasks, assuming the implementation required extensive back and front-end modifications. This resulted in "*suboptimal queries*" focused on getting assistance to an entirely over-engineered plan. Conversely, P7 started by asking reasonable questions and essentially received the correct answers but continued with some poorly phrased implementation questions, which caused deception. The fourth poorest SPAC-B-assisted plan came from P1, who asked SPAC-B to create the whole implementation plan, tweaked the prompt, and asked again. This revealed that there

Table 4: Overview of scores the participants’ answers along with observation during the experiment and assessments of SPAC-B. Participant (P), Feature Location (FL), Implementation (I), Without SPAC-B assistance (W/o), With SPAC-B assistance (W/)

P	Grp	W/o		W/		Significant observations from the experiment	Assessment of SPAC-B	
		FL	I	FL	I			
1	2	1	1	2	2	Used IDE’s search function to but could not locate the correct files.	Trusted SPAC-B full implementation answer without verifying.	It is helpful for newcomers in large software projects.
2	4	1	1	5	4	Spent the entire time between a test file and using Google for looking up things.	Queried a full implementation. Asked follow-up questions and tried to verify.	It can highlight the process of making a feature change in all its steps.
3	1	1	2	4	3	Used the IDE’s search function to look for a settings file but did not find the right file.	Queried SPAC-B with increasing granularity as the participant learned about the project.	Was pleasantly surprised with its ability to reason about the particular project.
4	3	5	3	3	3	Displayed very efficient use of IDE’s search function to find all the relevant files.	Queried SPAC-B to locate code, looking for specific functions or labels.	It is not helpful compared to the search functionality in the IDE.
5	2	5	3	3	3	Found all relevant files by browsing the file structure and using search.	Noted that a slight modification to query gives a more helpful answer.	Not sure how it differs from services like CoPilot.
6	4	2	1	1	1	Unsuccessfully tried to utilize a debugger and resorted to searching in the IDE.	Mixed up album and playlist, and thus queries SPAC-B about the wrong button.	Was impressed with the meaningful answers and ease of communicating.
7	1	1	1	1	1	Browsed files in IDE but struggled to find FL. delivered an answer too vague to properly interpret	Ignored useful SPAC-B responses and instead followed a misleading answer due to missing information in the prompt.	It looks promising and provides potential solutions.
8	3	2	2	1	1	Assumed task is much more complicated it is, outlines a complex and almost entirely wrong plan.	Queried SPAC-B for backend timer implementation and fronted implementation without pointing to desired location	It might deliver too much information which people without sufficient background knowledge might not take in.
9	1	2	2	5	5	Searched for settings files and figured out which one requires modification, but could not create plan to implement logic.	Quickly located one of the relevant files without SPAC-B and found the other after querying SPAC-B.	It was very helpful and very good at finding the relevant code.
senic	4	1	1	3	3	Had trouble utilizing the search functionality to locate relevant files, and ended up browsing through the file structure feeling overwhelmed.	Queried SPAC-B many times, receiving responses of varying degrees of helpfulness. Spent a lot of time looking through the suggestions from SPAC-B.	It gave general information about the project which helped to familiarize with the code-base.
Mean		2.1	1.7	2.8	2.6			
SD		1.6	0.8	1.6	1.4			

could be placed *"too much faith in AI tools"*, as the response was misleading as the supposed GUI file had *"test"* in the name.

P4 outlined an adequate plan when assisted by SPAC-B, but a better one which scored higher in file identification without assistance, and reported not finding the tool very helpful. Two factors affected this: P4 stated that *"some answers were misleading"* as they mistakenly pointed to testing files and, therefore, had to spend time double-checking every answer from SPAC-B. After this response, SPAC-B was updated to version 0.2.0, where participants could exclude testing files. Additionally, P4 solely queried SPAC-B with feature location questions and was impressed when an implementation query was demonstrated after the experiment. It can, therefore, be inferred that P4 would have performed better with SPAC-B assistance if version 0.2.0 of SPAC-B had been available and if SPAC-B had been thoroughly introduced.

Like P4, P5 outlined a decent plan with SPAC-B but a better one without. P5 based most of the SPAC-B-assisted plan on an elaborate implementation query, which was helpful but vague. With two minutes remaining, the query was tweaked, which resulted in a much better response, but P5 had *"insufficient time to use the information"*. Again, this underlines *"the importance of proper prompting techniques"*, as found in the case study, when comparing informed and uninformed questions.

4.2.4 SPAC-B assessments. The participants reported greatly varying expectations of SPAC-B, ranging from not knowing what to expect (P6) to expecting it to provide a well-structured prompt if given an informative one (P1). They reported similarly different query strategies, such as *"Be specific and provide all necessary information in one prompt."* (P9), *"Query strategy to treat the AI as I would*

normally the programmer who made the project." (P3), and *"First, I wanted to see if it could give me all the steps, so basically asking the task question. Then afterwards, asking for the specifics for each of its steps."* (P2). 8 out of 10 participants reported that the answers SPAC-B provided were helpful with P2 explaining: *"Very much. I would otherwise have had no idea where to start."* Conversely, P4 states that not all answers were useful. When asked for their overall thoughts about using SPAC-B, their responses followed a similarly positive pattern as summarized in Table 4. P4 preferred searching in the IDE for files due to a misleading answer from SPAC-B, and while P5 stated that SPAC-B seems to be time-saving but is uncertain how it differs from a service such as CoPilot. P8 thought SPAC-B might deliver too much information that people without sufficient background knowledge might not take in. The remaining participants expressed they found SPAC-B helpful and/or were impressed with its capabilities. P3 is a lead developer responsible for onboarding new employees and thought tools like SPAC-B could be helpful.

To properly assess where participants found SPAC-B lacking, they were asked how they would like SPAC-B improved. The most sought-after improvement was speed, as participants would like a faster response time (P1, P2, P3, P7, P9, P10). P1, P2, P3, P9, and P10 further desired overall improvements to the UX/UI of SPAC-B. P1 and P5 would like SPAC-B to be integrated into the IDE to see the suggested modifications and files highlighted. P8 and P10 suggested that conversational memory should be added to SPAC-B since it would help with follow-up questions. P4 specifically asked for a lower ranking of testing files as these caused misleading answers. Following this feedback, SPAC-B was updated to v.0.2.0 with a toggle button, letting the user omit testing files if desired.

5 THREATS TO VALIDITY

Internal validity: How queries are phrased for the case study impacts the level of detail in the tool’s responses. Due to the limited number of participants, there is a risk of non-representative interaction from the developers in the study. Gradings might further be biased by SPAC-B’s or by participants’ answers. This was mitigated by developing an explicit grading schema, available at at Zenodo[11], based on accepted solutions from within the projects.

External validity: The participants have varying levels of programming experience and there are not enough data points to draw conclusions based on the groups. Further, the indexing in SPAC-B is implemented to handle the 20 most popular programming languages and languages of similar syntax. Additionally, it handles documentation and metadata languages such as Markdown and LaTeX. **SPAC-B is, therefore, capable of assisting with almost any software projects**, as GPT-4 is also trained in these popular languages. However, GPT-4 is a black box model, and variations in response quality for different languages may exist.

Construct validity: The participants had to connect to our PC to conduct the experiment through TeamViewer, i.e. worked in an environment they were not used to with added latency, which might impact their overall experience. Out of the tested tools, TeamViewer had least latency issues. Also, the two tasks chosen for the experiments might not be completely comparable. This might, to a certain extent, influence the score. However, this was mitigated by choosing two similar but non-overlapping tasks. Furthermore, as presented in the methodology section, the participants were divided into four experiment groups, thus eliminating any possible discrepancies.

6 CONCLUSIONS

In this study, we investigated whether Generative AI can assist newcomers in understanding software projects through the RAG-based tool SPAC-B. A case study has investigated SPAC-B’s accuracy, and an experiment in which participants created plans for open-source software issues with and without SPAC-B has investigated SPAC-B’s impact. The most important findings are:

- SPAC-B provides accurate responses, which score an average of 4.4 across metrics.
- The implementation plans improved from 1.9 to 2.7 on average when assisted by SPAC-B, showing a tendency that SPAC-B is impactful.
- 8 out of 10 participants found SPAC-B helpful.

SPAC-B provides the user with accurate responses. The mean scores achieved in the case study were 4.7 (std. 1.0) for feature location and 4.1 (std. 0.8) for implementation questions. The implementation plans from the participants in our experiment improved from 1.9 to 2.7 when assisted by SPAC-B. These results indicate that Generative AI can be used to help with newcomers’ understanding of software projects.

While the tool is still a prototype, 8 out of 10 participants expressed that SPAC-B provided helpful answers. Similarly, their responses followed a positive pattern when asked for their overall thoughts about using SPAC-B. However, multiple participants would like to have improved some aspects of the tool, such as UX/UI, IDE integration, conversational memory, and response time, the latter of which is largely dependant on the API call response time.

Future work will consider more extensive studies with a larger and more diverse set of participants, as well as projects of varying sizes and complexities, including commercial ones. Additionally, research that tests design capabilities in providing advice on documentation-related questions would help to better understand the generalizability of this approach.

REFERENCES

- [1] Lenz Belzner, Thomas Gabor, and Martin Wirsing. 2023. Large language model assisted software engineering: prospects, challenges, and a case study. In *International Conference on Bridging the Gap between AI and Reality*. 355–374.
- [2] LangChain Community. 2024. FAISS. https://api.python.langchain.com/en/latest/vectorstores/langchain_community.vectorstores.faiss.FAISS.html.
- [3] LangChain Community. 2024. GitLoader. https://api.python.langchain.com/en/latest/document_loaders/langchain_community.document_loaders.git.GitLoader.html.
- [4] LangChain Community. 2024. RecursiveCharacterTextSplitter. https://python.langchain.com/api_reference/text_splitters/character/langchain_text_splitters.character.RecursiveCharacterTextSplitter.html.
- [5] Barthélemy Dagenais, Harold Ossher, Rachel KE Bellamy, Martin P Robillard, and Jacqueline P De Vries. 2010. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—Volume 1*. 275–284.
- [6] Christof Ebert and Panos Louridas. 2023. Generative AI for software practitioners. *IEEE Software* 40, 4 (2023), 30–38.
- [7] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.
- [8] Samia Kabir, David N Udo-Imeh, Bonan Kou, and Tianyi Zhang. 2024. Is stack overflow obsolete? an empirical study of the characteristics of chatgpt answers to stack overflow questions. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–17.
- [9] Amy J Ko, Thomas D LaToza, and Margaret M Burnett. 2015. A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering* 20 (2015), 110–141.
- [10] Knud Ronau Larsen and Magnus Edvall. 2024. SPAC-B Repository. <https://github.com/KnudRonau/SPAC-B>.
- [11] Knud Ronau Larsen and Magnus Edvall. 2024. SPAC-B Research Paper Documents. <https://doi.org/10.5281/ZENODO.13622439>.
- [12] Jenny T Liang, Chenyang Yang, and Brad A Myers. 2024. A large-scale survey on the usability of AI programming assistants: Successes and challenges. In *Proceedings of the 46th IEEE/ACM Intl. Conference on Software Engineering*. 1–13.
- [13] Gerardo Matturro, Karina Barrella, and Patricia Benitez. 2017. Difficulties of newcomers joining software projects already in execution. In *2017 International Conf. on Computational Science and Computational Intelligence*. IEEE, 993–998.
- [14] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.
- [15] Israr Qureshi and Yulin Fang. 2011. Socialization in open source software projects: A growth mixture modeling approach. *Organizational Research Methods* 14, 1 (2011), 208–238.
- [16] Ayushi Rastogi, Suresh Thummalapenta, Thomas Zimmermann, Nachiappan Nagappan, and Jacek Czerwonka. 2017. Ramp-up journey of new hires: Do strategic practices of software companies influence productivity?. In *Proceedings of the 10th Innovations in Software Engineering Conference*. 107–111.
- [17] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. 2023. The programmer’s assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. 491–514.
- [18] Jürgen Rudolph, Shannon Tan, and Samson Tan. 2023. War of the chatbots: Bard, Bing Chat, ChatGPT, Ernie and beyond. The new AI gold rush and its impact on higher education. *Journal of Applied Learning and Teaching* 6, 1 (2023).
- [19] Igor Steinmacher, Marco Aurélio Gerosa, and David Redmiles. 2014. Attracting, onboarding, and retaining newcomer developers in open source software projects. In *Workshop on Global Software Development in a CSCW Perspective*, Vol. 16. 20.
- [20] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 59 (2015), 67–85.
- [21] Igor Steinmacher, Igor Wiese, Ana Paula Chaves, and Marco Aurélio Gerosa. 2013. Why do newcomers abandon open source software projects?. In *2013 6th Intl. Workshop on Cooperative and Human Aspects of Software Engineering*. 25–32.
- [22] Man-Fai Wong, Shangxin Guo, Ching-Nam Hang, Siu-Wai Ho, and Chee-Wei Tan. 2023. Natural language generation and understanding of big code for ai-assisted programming: A review. *Entropy* 25, 6 (2023), 888.