Meet C4SE: Your New Collaborator for Software Engineering Tasks

Gabriele De Vito, Stefano Lambiase, Fabio Palomba, Filomena Ferrucci
Software Engineering (SeSa) Lab – University of Salerno, Salerno, Italy
gabriele.devito@gmail.com, slambiase@unisa.it, fpalomba@unisa.it, fferrucci@unisa.it

***Abstract*—The software industry has rapidly increased in complexity and scale, leading to challenges in managing information and tasks among developer teams, often resulting in inefficiencies, misunderstandings, and delays. Moreover, the increasing search for automated tasks led to the extensive adoption of chatbots—a.k.a. conversational agents—for software development purposes. However, despite their undoubted positive contributions, practitioners started to identify numerous issues deriving from their adoption, both technical and social, first of which, the uselessness of the provided support due to the bot's lack of full working context. To address such a limitation, we propose C4SE, a chatbot designed to assist software engineers and managers in performing several tasks. The idea behind the bot is to collect information from the different tasks that could be useful for others to provide better support and tailor the bot to the specific operational context—i.e., the development team using it. To enable such task heterogeneity and contextual persistence, we operationalize the GPT 3.5 model for understanding the user's intent and a specialized data store based on a vector database for long-term memory for maintaining contextual information. With these characteristics, C4SE can provide benefits to the entire software development lifecycle increasing practitioners' productivity. We presented a prototype of the tool able to perform code suggestions, code reviews, GitHub API operationalization, and unit and acceptance test case generation. A preliminary evaluation was carried out reporting encouraging results.**

***Index Terms*—Chatbot, software engineering, large language model, LangChain, vector database**

I. Introduction

Software development has become increasingly complex, requiring software teams to handle an overwhelming amount of information and tasks. For such a reason, chatbots (a.k.a. conversational agents) have become popular thanks to their ability to automate tasks and improve communication and collaboration among team members [3, 25]. Specifically, practitioners have started to use chatbots for a large plethora of tasks, e.g., supporting writing code [2, 31], information sharing [10, 17], and improving communication and collaboration in team [6]. Moreover, big techs have been proposing their own solutions for supporting developers during code writing, like Copilot,[1] for GitHub, and Tabnine.[2]

However, existing chatbots for software engineering purposes (SE chatbots) have several limitations. They cannot often fully contextualize users' prompts, resulting in limited flexibility and clarity in communication [15, 28, 31]. Moreover, they usually support a very narrow set of tasks (usually a single task such as code suggestion), limiting their effectiveness and value to software engineers. Furthermore, chatbots are usually trained with external resources (e.g., Stack Overflow and GitHub). In this way, they cannot learn from the data available within a software development project to provide personalized and context-driven support. The above limitations have been partially addressed by Large Language Models (LLMs)—i.e., complex models consisting of a neural network trained on large quantities of unlabelled text using self-supervised learning [23]. Systems using such models, like GPT-3.5,[3] can contextualize and reason over prompts and converse naturally and flexibly, eliminating the context issue described above. However, current LLMs have limitations, including challenges in controlling them to obtain the desired output, fixed and limited knowledge bases (KBs), and no memory of previous conversations.

Starting from the limitations mentioned above, this paper introduces C4SE, a chatbot powered by GPT-3.5, and designed specifically for software engineering teams. C4SE can be seen as a value-added software team member and can assist with a large plethora of tasks—e.g., code review, code suggestion, unit and acceptance test case generation, and GitHub task execution—improving team productivity, collaboration, and increasing projects' success rate. C4SE implements a novel storage technology—based on vector databases— for implementing long-term memory, allowing it to learn quickly and remember past interactions with the team.

The contributions of our paper are threefold. Firstly, we introduce C4SE, an open-source, fully extendable bot for supporting software engineers during all the phases of the software development lifecycle. Moreover, we provide a novel approach for building and designing chatbots that go beyond the limitations of traditional SE chatbots by enabling our chatbot to (1) understand and maintain the context of the conversation, (2) answer users' requests according to past interactions, and (3) propose tailored solutions on problems that go beyond his "default" knowledge. Finally, our paper provides insights into the potential of LLMs and vector databases to dynamically enrich LLM's knowledge base and ensure accurate answers to user queries and practical support to software engineers. With our study, we hope to help teams collaborate more effectively, with a consequent increase in the practitioner's quality of life.

[1]GitHub Copilot site: https://github.com/features/copilot
[2]Tabnine site: https://www.tabnine.com
[3]GPT Models site: https://platform.openai.com/docs/models

II. Related Work

The recent progress in Artificial Intelligence (AI) and Natural Language Processing (NLP) led to the widespread utilization of *chatbots* [24, 29], especially in the software engineering context [3, 21, 27]. A *chatbot* is a particular type of *bot*—an application that automates repetitive, dull, or predefined tasks [20, 25]—characterized by the fact that it communicates with users using a communication channel and natural language.

In the context of code generation, Qasse et al. [21] proposed *iContractBot*, a conversational bot, to aid developers in constructing smart contracts on blockchain platforms. A use case is presented to demonstrate how the bot can be utilized for creating models and generating the deployment artifacts for smart contracts based on a simple conversation. Moreover, GitHub and OpenAI collaborated to release *GitHub Copilot*, a so-called "AI pair programmer" able to support developers in writing source code. Utilizing the comprehensive OpenAI Codex model, which underwent training on open-source GitHub code, Copilot is competent in proposing code snippets for various programming languages starting from user comments. More related to testing activities, Okanović et al. [18] developed PerformoBot. The bot operates in the context of load testing and guides developers through the process of properly specifying the parameters of the test, other than automatically executing it.

Analyzing such tools, we identified two important limitations when analyzing the state of the art about bots for SE tasks. First, most of the available bots are designed for performing a single (or few) tasks, forcing developers to adopt many tools—often not goodly integrated through each other—during their work cycle. Second, practitioners lament a lack of context awareness of the utilized bots [15, 28, 31], which answer questions generally and not considering their working habits.

LLMs, such as GPT-3.5 and ChatGPT [19], have emerged as a valuable technology to perform NLP tasks and develop chatbots. Indeed, ChatGPT is a chatbot model that can quickly generate high-quality text, saving time and effort in the software development process. ChatGPT can assist in various tasks in software development, such as requirements elicitation, natural language processing, and automated code generation [1, 8]. However, ChatGPT has several limitations and drawbacks. The first limitation is the chatbot's context size, namely the total number of tokens that comprise the user's input and the generated text, which is limited to 4096 tokens. So, it cannot deal with long conversations or analyze long documents, such as use case documents or other software artifacts. In addition, the knowledge base on which it was trained is fixed and limited to the end of 2021. So, ChatGPT does not know what happened in the world after that date, and consequently, in the software engineering field. For example, asking ChatGPT to explain what LangChain is will answer as follows: *"I'm sorry, but I don't have any information on a technology or concept called "Langchain". Could you please provide more context or background information so I can better understand what you are referring to?"*. Moreover, it cannot execute scripts or external software, which limits its potential, especially regarding software engineering tasks.

As a result, there is a need to find different approaches for chatbots based on LLMs to manage long texts (for example, for Q&A), enlarge their memory to deal with extended conversations, augment their knowledge base, and execute external software and functionalities. C4SE aims to address these limitations, as described in detail in the next section.

III. The C4SE System

This work aims to present C4SE, a chatbot designed to perform a large plethora of software engineering tasks and support both engineers and managers during the software development lifecycle. The bot relies on three different technologies: LangChain,[4] GPT-3.5, and Pinecone.[5] LangChain is a software development framework designed to simplify the creation of applications using LLMs. LangChain helps create more complex workflows using agents and chaining different prompts interactively. GPT-3.5 is one of the most advanced LLMs and a powerful tool for natural language processing tasks. Therefore, it allows C4SE to provide an intuitive and fluid conversation with users. Finally, Pinecone is a fully managed vector database that stores and indexes the vector representation of NLP sentences, allowing for fast and accurate similarity search and retrieval across massive datasets of unstructured data. Combining these technologies allows for the orchestration of the different tasks of C4SE and the enhancement of the bot with a large knowledge base.

Another essential aspect to consider is that LangChain allows using different types of LLMs (i.e., AI21,[6] Aleph Alfa,[7] Cohere,[8]) and different types of vector databases (i.e., Qdrant,[9] Chroma,[10] and Redis[11]) and simply interfacing new ones. Therefore, it is possible to choose the most suitable tool for each task; for instance, it is possible to make various LLMs interact to carry out a specific task in the best way.

A. Motivating Scenarios

In this section, to better understand the potential impact and contribution of C4SE, we present two scenarios—as-is and visionary [5]. The former aims to provide the current state and identify limitations and challenges; the second present the situation with the tool and stresses the gained benefits (the two scenarios share the same characteristics for the actor).

As-is Scenario

[4]LangChain site: https://github.com/hwchase17/langchain
[5]Pincone site: https://www.pinecone.io/
[6]AI21 site: https://docs.ai21.com/
[7]Aleph Alfa site: https://docs.aleph-alpha.com/docs/introduction/luminous/
[8]Cohere site: https://cohere.com/about
[9]Qdrant site: https://qdrant.tech/
[10]Chroma site: https://github.com/chroma-core/chroma
[11]Redis site: https://redis.com/solutions/use-cases/vector-database/

Mary is a software engineer working in a software team that has to implement a software module starting from a use case. Such a software module must take a list of books (as information) from an API and translate them into a format that an already implemented web application uses. Mary has no knowledge about the API, so they ask a bot, GitHub Copilot, to help implement the module. Therefore, Copilot provides Mary with the code, and Mary implements it and makes the software module. Then, they use Probot—a bot for automatically performing some action on GitHub—to push the code to the repository. On the day above, Mary wanted to update the code and make some documentation for acceptance testing. First, Mary indicates Probot to clone the code from the repository; to do this, she needs to specify precisely the repo and code to clone. After updating it, she started from the use case to write the acceptance testing for the implemented feature. Unfortunately, there is no bot for performing this, so Mary has to do this on their hand.

The as-is scenario described a typical situation reported by many works on state of the art about chatbots adopted for software engineering. First, using different bots to perform each task leads to a **loss of productivity** due to the time spent interacting with and mastering the different tools [26, 28]. Second, linked to the previous issues, there is a **lack of context issue**; the fact that the tools do not share or maintain knowledge leads developers to repeat essential information to achieve their goals [15, 28, 31]. Moreover, the use of third-party tools—most of the time deployed on the web—could also bring **privacy issues**—at least in terms of practitioners' perceptions [4, 9, 16].

Visionary Scenario

[...] Mary has no knowledge about the API, so he asks C4SE to help implement the module. Therefore, the bot provides Mary with the code, and Mary implements it and makes the software module. Moreover, Mary asks the bot to push the developed component on the GitHub repository. The day after, Mary wanted to update the code and do acceptance testing for the developed feature. Using C4SE, he simply asks the bot to provide the code generated the day before and give it the use case asking for acceptance testing specification; C4SE, using the collected context, provides a possible test based on the high-level specification and the code.

From the visionary scenario, we can evince how the previously-mentioned limitations are partially or fully addressed. First of all, having one bot that does everything negates the need to move to different tools and waste time. Moreover, the capacity of C4SE to maintain information and the fact that all the tasks involve its use lead to the maintenance of the context and address the lack of contextual information

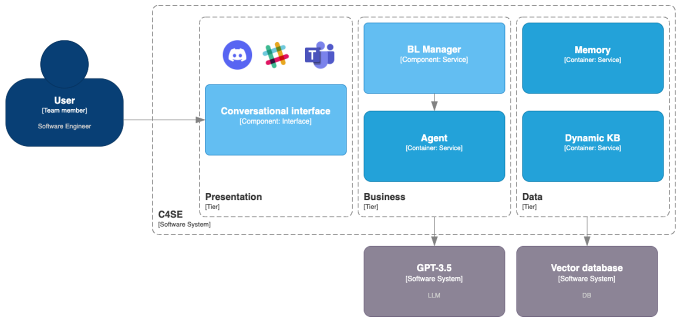


Fig. 1. The C4SE's architecture.

that characterizes the state-of-the-art solutions. Furthermore, C4SE is designed to be an open-source on-premise tool resolving the privacy issues derived from using on-cloud third parties tools. Finally, as an ulterior note, our solution partially resolves the well-known ***noise problem***—i.e., the tendency of a bot or a set of collaborating bots to be unnecessarily verbose, leading to information loss [9, 28]. Indeed, having a single bot to communicate with that is tailored to the single user and business brings improved communication through a more precise interaction.

B. Tool Architecture and Functionalities

The main goal of C4SE is to provide practitioners with a unique place to perform the majority of their tasks—e.g., code and test generation. In order to reach this, we developed the tool as a typical three-tier application characterized by various modules depicted in Figure 1.

Conversational Interface. This component is responsible for making the bots feature available to different communication platforms—e.g., Teams, Slack, and Discord.

BL Manager. Such a module is responsible for classifying user requests and executing the steps to fulfill them—implemented using agents described below. It exploits the power of GPT-3.5 to classify requests using a specialized prompt, which allows launching the correct agent based on user queries and executing external scripts and services that cannot be done with GPT-3.5 alone—e.g., GitHub API query.

Agents. Each potential request for C4SE—or set of cohesive requests—is mapped in an agent. For example, all the requests related to GitHub are responsible for a single agent (GitHub Task Agent). An agent is a software component implementing the application logic to fulfill a specific request. The package Agents contains the developed agents and offers a unified interface for querying them. Each agent is a service that uses its specific pipeline (or chain) to produce responses. A chain allows the joining of various components to form a unified application. For instance, we can create a chain that accepts input from the user, applies a prompt template to format it, and transfers the formatted response to an LLM. We can develop more complex chains by integrating numerous chains or combining chains with other components (i.e., Vector databases, LLMs, Document loaders).

Memory. This module uses a specialized vector database to memorize conversations between C4SE and users, allowing long-term memory. We can configure the Memory component to temporarily (i.e., for an hour) or permanently store the conversations. Having long-term memory, C4SE can extract the most relevant information from previous conversations to answer user queries appropriately. For example, suppose the user has previously asked C4SE to write a Java Entity class using JPA and wants the bot to explain the code. In that case, the user can ask, "Can you explain the previous code?". C4SE, using the Memory module, will retrieve from the conversation history with that specific user (the vector database index) the information relevant to the question (i.e., the previous Java class). The bot will then augment the context and, exploiting the correct agent, send the question to the LLM to provide the final answer.

Dynamic Knowledge Base. Such a module stores and indexes new data as embedding vectors into C4SE. It provides the necessary services to "upsert" (i.e., update and insert) and "retrieve" documents exploiting vector databases. Documents can be of any type, such as plain text, pdf, html, Microsoft Word, Excel, Powerpoint, XML, etc. The upsert service converts documents into embeddings and indexes them in the vector database. The retrieval service allows searching for the most relevant documents from a semantic point of view to a given search query. This component allows searching for the most relevant information in the C4SE knowledge base and ingesting this information as an augmented context into the prompt to send to GPT-3.5. For example, we can upload the LangChain documentation using the upsert service and then ask C4SE questions about it (i.e., "provide the snippet to develop a ConversationalRetrievalChain in LangChain"). C4SE will forward the questions to the correct agent (i.e., "SnippetAgent"). Using Dynamic memory, the agent will retrieve the information relevant to the questions and augment the context with it. In our example, this means injecting into the final prompt the collected information as follows:

Augmented prompt

"... Answer the QUESTION at your best.
Current conversation:{augmented_context}
QUESTION: {input}"

In the prompt above, the variable "augmented_context" represents the augmented information, namely the most relevant *k* results retrieved from the vector database. Then, using the augmented prompt, the agent will request the LLM to generate the final answer.

The functional behavior of C4SE is illustrated by the sequence diagram in Fig. 2 that shows how the key components interact to respond to user requests. Let us examine the following example to clarify the system interactions:

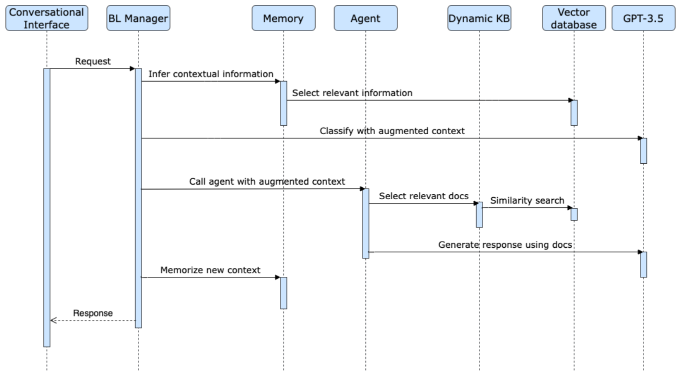


Fig. 2. C4SE functional behavior.

1) The user asks the system the following question: *"Please, provide an example of a prompt template code usage in LangChain."*
2) The Conversational Interface forwards the request to the BL Manager component, which uses the Memory component to retrieve the most relevant information based on the previous conversations.
3) The Memory component exploits the Vector DB to retrieve the most relevant information related to the user's query.
4) Suppose there are no conversations yet; the Memory component returns an empty list.
5) The BL Manager component sends the classification prompt to the LLM, asking GPT-3.5 to classify the user's question as follows:

Classification prompt

Classify the following question as Code task, GitHub task, Test case, or Q&A. The rules are the following:
- The question should be classified as a GitHub task if it means executing a task on or related to a specific repository.
- The question should be classified as a Test case if and only if it explicitly requests an acceptance test case for a provided use case.
- The question should be classified as a "Code task" if it is related to code or unit test code.
- The question should be classified as Q&A in all the other cases.
Provide only the classification.
Current conversation:{history}
Question:{input}

6) The LLM classifies the user's question as a "Code task."
7) The BL Manager component instantiates the Code task agent to process the user's request.
8) To answer the user's request, the Code task agent inquiries the Dynamic KB component for the most relevant documents.

9) The Dynamic KB retrieves the most relevant information using the Vector database as follows:

Relevant information

PromptTemplates are responsible for constructing a prompt value. They are basically objects that expose a format_prompt method for constructing a prompt. For example
from langchain.prompts import PromptTemplate
string_prompt =
PromptTemplate.from_template("tell me a joke about {subject}")

Part of the context has been omitted for simplicity.

10) The Code task agent sends a new query to the LLM, augmenting the context with the information retrieved, as follows:

New prompt with augmented context

Act as a software engineer. Check if the question relates to the last code we discussed in the conversation. Answer the question at your best.
Current conversation:{augmented_context}
QUESTION: {input}

Where the variable "augmented_context" contains the new augmented context.

11) GPT-3.5, using its "in-context learning" capabilities, provides the requested code as follows:

GPT-3.5 final response

Sure, here's an example usage of PromptTemplate in LangChain:
from langchain.prompts import PromptTemplate
# Define a prompt template
prompt_template = PromptTemplate(
input_variables=["subject"],
template="Tell me a joke about {subject}.")
# Format the prompt template with input
formatted_prompt =
prompt_template.format(subject="cats")
# Print the formatted prompt
print(formatted_prompt)

12) The BL manager component memorizes the new conversation using the Memory component. It sends the final response to the Conversational interface.

C. Novel Characteristics

C4SE presents a novel approach to address the challenges faced by software engineering practitioners in their day-to-day tasks. The bot's architecture is designed to provide a unique platform for performing various tasks while addressing the challenges existing chatbots pose.

First, our bot was designed specifically for the software engineering context and, consequentially, for software engineers and project managers. In fact, we designed a specialized prompt to classify users' requests and provide them with the correct answer or interaction. Moreover, such a focus on software engineering allows the bot to be more accurate in its interactions, thus limiting the risk of producing inaccurate responses and lowering practitioners' productivity.

Second, LLMs are stateless and limited in their memory conservation, other than presenting, in most cases, query limits (i.e., the length of a single request/response exchange) [7]. This limitation makes it challenging to maintain the conversation chain, leading to a loss of context and causing frustration in users. C4SE addresses this challenge by using a specialized data store to memorize conversations between the bot and users, thus allowing long-term memory contextual information. This mechanism enables C4SE to answer more accurately based on the conversation context by splitting the user's request into multiple tasks and combining multiple answers to overcome the LLM's query limits.

Third, software engineering constantly evolves, requiring practitioners to be up-to-date with new technologies, processes, methodologies, and other developments. C4SE is built to be always up-to-date with an infinite knowledge base that enables practitioners to acquire new skills and execute new SE tasks. The bot's dynamic knowledge base is another novel feature of C4SE. C4SE can also use multiple Knowledge Base memory modules (using several vector databases) to serve different agents. This aspect has two important implications: 1) the bot can scale by using several KB memory instances; 2) each agent can have its specialized dynamic memory.

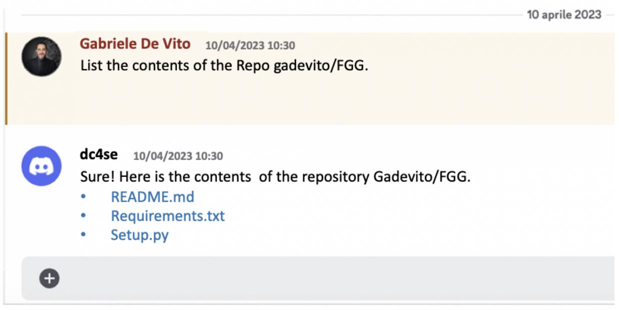
Finally, the bot's architecture is distributable and can exploit microservices and containerized computing technologies. Concretely, each agent—responsible for handling a single task and the corresponding interaction with users—can be implemented as microservices exploiting its own resources.

D. Use Case Example

At the current state, we have tested and validated five features of C4SE, i.e., code suggestion, code review, GitHub task execution, unit testing generation, and acceptance testing generation. It is important to note that, seeing the knowledge base of the bot, the potential tasks that it can perform are much more than these. This section presents a few examples of C4SE in action.

The conversation between the user and C4SE starts with the user asking C4SE to show the list of the contents of a GitHub repository, as shown in Figure 3. The chain of reasoning C4SE performs on the request and the contextual information allows the tool to identify what the user wants accurately.

In the second request—Figure 4—the user asks for a code snippet. In this case, C4SE executes the Coding agent because the request is related to coding. Moreover, as shown in Figure 5, the users request the test code to test the component.

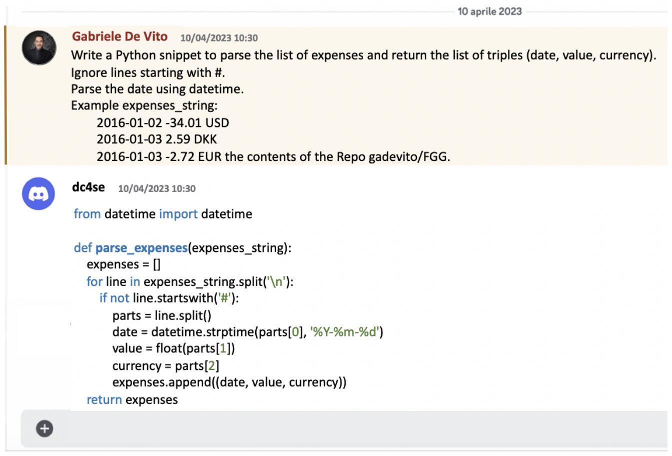


10 aprile 2023

Gabriele De Vito 10/04/2023 10:30
List the contents of the Repo gadevito/FGG.

dc4se 10/04/2023 10:30
Sure! Here is the contents of the repository Gadevito/FGG.
- README.md
- Requirements.txt
- Setup.py

Fig. 3. C4SE GitHub task example.



10 aprile 2023

Gabriele De Vito 10/04/2023 10:30
Write a Python snippet to parse the list of expenses and return the list of triples (date, value, currency).
Ignore lines starting with #.
Parse the date using datetime.
Example expenses_string:
2016-01-02 -34.01 USD
2016-01-03 2.59 DKK
2016-01-03 -2.72 EUR the contents of the Repo gadevito/FGG.

dc4se 10/04/2023 10:30

```
from datetime import datetime

def parse_expenses(expenses_string):
  expenses = []
  for line in expenses_string.split('\n'):
    if not line.startswith('#'):
      parts = line.split()
      date = datetime.strptime(parts[0], '%Y-%m-%d')
      value = float(parts[1])
      currency = parts[2]
      expenses.append((date, value, currency))
  return expenses
```

Fig. 4. C4SE Code Snippet example.

Then, the user asks C4SE to provide the contents of a specific file in the same GitHub repository of the first example—Figure 6—without reporting further details. As we can notice, C4SE has the memory of the previous conversation and can execute the task correctly.

In the last example—Figure 7—the user's request is related to a use case (CPM:US1.001), which was previously uploaded in the C4SE knowledge base DB. In this case, the user asks to produce the acceptance test. C4SE infers contextual information from its KB. It ingests the information in the context of the Test Cases agent that can produce the requested test case.

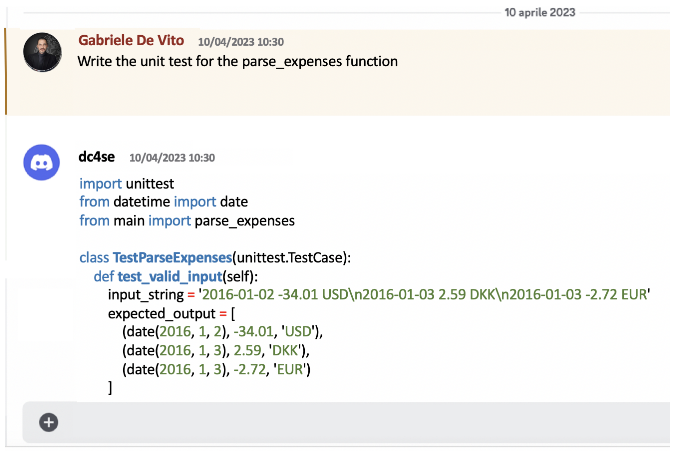
IV. C4SE Evaluation

This section presents the preliminary evaluation of the tool and associated lessons learned.

A. Evaluation Methodology

As a preliminary evaluation of C4SE, we decided to assess its usability. The ISO 9241-11 standard [12] defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". Effectiveness,

[12]ISO 9241-11 site: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en



10 aprile 2023

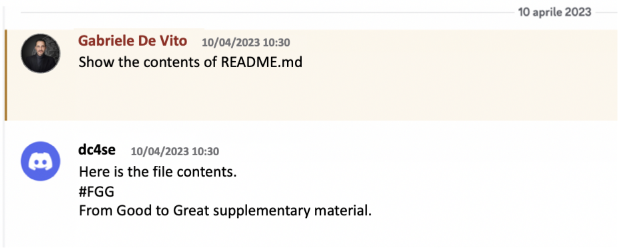
Gabriele De Vito 10/04/2023 10:30
Write the unit test for the parse_expenses function

dc4se 10/04/2023 10:30

```
import unittest
from datetime import date
from main import parse_expenses

class TestParseExpenses(unittest.TestCase):
  def test_valid_input(self):
    input_string = '2016-01-02 -34.01 USD\n2016-01-03 2.59 DKK\n2016-01-03 -2.72 EUR'
    expected_output = [
      (date(2016, 1, 2), -34.01, 'USD'),
      (date(2016, 1, 3), 2.59, 'DKK'),
      (date(2016, 1, 3), -2.72, 'EUR')
    ]
```

Fig. 5. C4SE Unit Test example.



10 aprile 2023

Gabriele De Vito 10/04/2023 10:30
Show the contents of README.md

dc4se 10/04/2023 10:30
Here is the file contents.
#FGG
From Good to Great supplementary material.

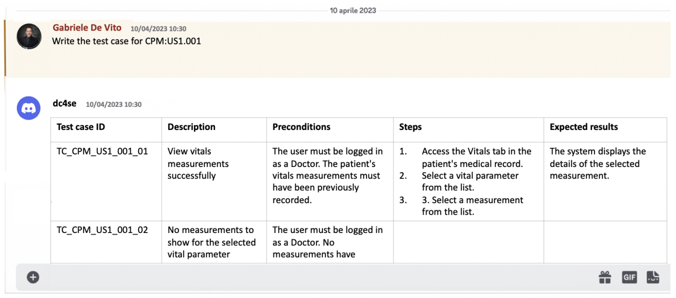
Fig. 6. C4SE GitHub task with ingested context.

efficiency, and satisfaction are considered important aspects in chatbot evaluation [11, 12, 22]. These aspects are so defined:

- *Effectiveness*, i.e., the accuracy and completeness with which users achieve a desired goal;
- *Efficiency*, i.e., the trade-off between the resource used—expressed in terms of completion time, mental, and communication effort—and accuracy and completeness with which the users achieve their goals;
- *Satisfaction*, i.e., the user's satisfaction when using a tool in a specific context.

In order to evaluate the bot in terms of these three factors, we performed a two-step study that has been demonstrated to be helpful in evaluating chatbot usability [22]. First, we recruited 5 participants and asked them to complete a series of tasks using our bot in an experiment in vitro; then, we interviewed them to collect first impressions and feedback. For conducting the experiment, we followed the well-known guidelines provided by Wohlin et al. [30], while for the interviews, we followed the guidelines of Hove and Anda [13] and Kitchenham and Pfleeger [14].

We asked each participant to execute four tasks in line with the current version of C4SE. Specifically, we asked them to use the bot to (1) get a use case from a GitHub repository; (2) generate the Java or Python code to implement the selected use case; (3) generate an acceptance test case for the use case; and (4) generate the unit test for one of the classes obtained during the second task. At the end of



10 aprile 2023

Gabriele De Vito 10/04/2023 10:30
Write the test case for CPM:US1.001

dc4se 10/04/2023 10:30

Test case ID	Description	Preconditions	Steps	Expected results
TC_CPM_US1_001_01	View vitals measurements successfully	The user must be logged in as a Doctor. The patient's vitals measurements must have been previously recorded.	1. Access the Vitals tab in the patient's medical record. 2. Select a vital parameter from the list. 3. 3. Select a measurement from the list.	The system displays the details of the selected measurement.
TC_CPM_US1_001_02	No measurements to show for the selected vital parameter	The user must be logged in as a Doctor. No measurements have		

Fig. 7. C4SE Test Case example.

each step, participants analyzed the answers provided by the bot and, eventually, asked for explanations or more details. After this, each participant was interviewed by the first two authors of the paper about C4SE effectiveness, efficiency, and satisfaction. For each factor, participants expressed their level of agreement with a series of statements (Likert Scale from *Strongly Disagree (1)* to *Strongly Agree (5)*. For example, as for effectiveness, we asked for accuracy and completeness.

In experimenting with tools usability, accurately reproducing the tool operation context comes to be mandatory to collect reliable data [22]. For such reason, we selected 5 participants with experience in developing and evaluating the usability of software development tools; we selected them from our network through a convenience sampling strategy. Moreover, we provided them with a series of dummy information and scenarios to make them enter an imaginary development context to perform the required tasks.

B. Analysis of the Results

Regarding the results from the experiment, we collected both numerical values expressing the opinions of participants about the tool and opinions expressed in natural language. We recruited five graduate students (master's degree) who attended and achieved a Human-Computer Interaction course during their degree and have experience in using support tools for SE tasks—e.g., ChatGPT and Copilot.

Concerning the perceived *effectiveness* of the tool, participants expressed positive opinions. Indeed, related to the accuracy—i.e., the tool's capacity to correctly understand the user's request—all the participants evaluated the tool with a value of three or superior (one expressed 3 and the others 4). The only concern reported for this is the perceived need for documentation associated with each task; on this, one participant said that *"for being more precise and fast in the request, I would have appreciated having documentation telling me which information was important to provide the tool for doing the task."*. Moreover, regarding the completeness of the provided answers—i.e., the tool's capacity to provide all the relevant information for satisfying the user's needs—all the participants expressed positive opinions (one expressed 3 and the others 5).

Regarding the perceived *efficiency* of the tool, the collected opinions depicted a positive picture. Participants reported that it is easy when questioned about the tool's ease and rapidity of use. Still, in its current version, it needs some support to be adopted (all participants said values ranged from 3 to 4). We obtained similar results regarding how stressful it was to interact with the tool. More for this than in the previous evaluation, participants asked for ulterior documentation in order to correctly understand which pieces of information are mandatory for obtaining precise responses.

Concerning the perceived *satisfaction* derived from using the tool, participants were generally satisfied (two reported 4, two reported 3, and one reported 2). Specifically, they said that despite the undoubted usefulness and rapidity derived from its use, the tool—as well as other similar instruments—deprives them of some processes that, when satisfied, turn out to be rewarding. For example, one participant said, *"I enjoy writing code and solving a problem, even if trivial, on my own. It gives me great satisfaction. Using the tool makes it faster but takes away a little of that pleasure. Undoubtedly, getting advice rather than completely automating the process is better."*.

To summarize, participants in the experiment expressed positive opinions about the tool and its use to accomplish some tasks related to software engineering. These results encourage further tool development and studies.

C. Lessons Learned and Limitations

After developing and evaluating C4SE, we have gained valuable insights and identified a few limitations. We learned two essential lessons regarding accuracy and Human-Computer Interaction (HCI). Regarding the first, the quality of C4SE's responses depends on the quality and relevance of the data used to enhance the bot's knowledge base. Concerning the second, designing practical HCI is crucial, especially for software engineering chatbots. Since software engineers are often pressed for time, their interaction with C4SE must be efficient and straightforward.

On the other hand, despite the potential benefits of C4SE, there are some limitations. Firstly, C4SE is limited to a predefined set of tasks. Therefore, it is critical to expand its capabilities to enable it to perform a broader range of software engineering tasks. Secondly, C4SE is currently only available in English but should support other languages, broadening its user base. Thirdly, while C4SE is designed to be scalable, further work is required to ensure it can handle large user requests and messages without experiencing performance degradation. Thus, further research should focus on these aspects to improve the effectiveness and efficiency of C4SE.

Finally, the chatbot was evaluated in an experiment in vitro with few participants and a limited number of software engineering tasks. Therefore, the results may not be generalizable. Future studies could include a more diverse sample of participants and a higher number of tasks related to different domains in real-world scenarios.

V. Conclusion

In this paper, we introduced C4SE, a chatbot designed specifically for software engineering tasks aimed at addressing the limitations of existing chatbots and providing a unique platform for software engineering practitioners to perform a large plethora of tasks. C4SE exploits LLMs and vector databases, presenting a novel approach to address daily challenges for software engineering teams and potentially increasing their productivity and overall success in software engineering projects.

The evaluation of C4SE showed positive effectiveness, efficiency, and satisfaction results. Participants expressed positive opinions about the tool's capacity to understand the user's request accurately and provide all the relevant information for satisfying the user's needs. Moreover, the tool was perceived as easy to use and fast. Nonetheless, some participants asked for further documentation to understand which information is mandatory for obtaining precise responses.

Despite the potential benefits of C4SE, there are limitations to be addressed in future work. C4SE is currently limited to a predefined set of tasks and is only available in English. It needs expansion to support other languages and handle diverse tasks related to different domains for better generalizability. Furthermore, further research is required to evaluate C4SE with larger sample sizes and a broader range of software engineering practitioners.

REFERENCES

[1] A. A., W. M., L. P., F. M., A. M. S., and M. T., "Towards human-bot collaborative software architecting with chatgpt," *arXiv preprint arXiv:2302.14600.*, 2023.

[2] V. Alizadeh, M. A. Ouali, M. Kessentini, and M. Chater, "Refbot: intelligent software refactoring bot," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 823–834.

[3] N. C. Bradley, T. Fritz, and R. Holmes, "Context-aware conversational developer assistants," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 993–1003.

[4] C. Brown and C. Parnin, "Sorry to bother you: Designing bots for effective recommendations," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, May 2019, pp. 54–58.

[5] B. Bruegge and A. A. Dutoit, *Object-oriented software engineering; conquering complex and changing systems*. Prentice Hall PTR, 1999.

[6] J. Cerezo, J. Kubelka, R. Robbes, and A. Bergel, "Building an expert recommender chatbot," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, May 2019, pp. 59–63.

[7] L. D., R. A. S., and et al, "Large language models with controllable working memory," *arXiv preprint arXiv:2211.05110.*, 2022.

[8] S. D., B. M., H. C., and P. J., "An analysis of the automatic bug fixing performance of chatgpt," *arXiv preprint arXiv:2301.08653.*, 2023.

[9] L. Erlenhov, F. G. d. O. Neto, and P. Leitner, "An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 445–455. [Online]. Available: https://doi.org/10.1145/3368089.3409680

[10] K. Fukui, T. Miyazaki, and M. Ohira, "Suggesting questions that match each user's expertise in community question and answering services," in *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, July 2019, pp. 501–506.

[11] A. Genov, "Iterative usability testing as continuous feedback: A control systems perspective," *Journal of Usability Studies*, vol. 1, no. 1, pp. 18–27, 2005.

[12] K. Hornbæk, "Current practice in measuring usability: Challenges to usability studies and research," *International journal of human-computer studies*, vol. 64, no. 2, pp. 79–102, 2006.

[13] S. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *11th IEEE International Software Metrics Symposium (METRICS'05)*, 2005, pp. 10 pp.–23.

[14] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 63–92.

[15] G. Melo, E. Law, P. Alencar, and D. Cowan, "Understanding user understanding: What do developers expect from a cognitive assistant?" in *2020 IEEE International Conference on Big Data (Big Data)*, Dec 2020, pp. 3165–3172.

[16] M. Monperrus, "Explainable software bot contributions: Case study of automated bug fixes," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, May 2019, pp. 12–15.

[17] Z. Ni, B. Shen, Y. Chen, Z. Meng, and J. Cao, "Crowdevbot: A task-oriented conversational bot for software crowdsourcing platform (s)." in *SEKE*, 2019, pp. 410–522.

[18] D. Okanović, S. Beck, L. Merz, C. Zorn, L. Merino, A. van Hoorn, and F. Beck, "Can a chatbot support software engineers with load testing? approach and experiences," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 120–129. [Online]. Available: https://doi.org/10.1145/3358960.3375792

[19] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, K. S. Sandhini Agarwal, A. Ray, and et al, "Training language models to follow instructions with human feedback," *arXiv*, 2022.

[20] S. Pérez-Soler, E. Guerra, and J. de Lara, "Flexible modelling using conversational agents," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2019, pp. 478–482.

[21] I. Qasse, S. Mishra, and M. Hamdaqa, "icontractbot: a chatbot for smart contracts' specification and code generation," in *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 2021, pp. 35–38.

[22] R. Ren, J. W. Castro, S. T. Acuña, and J. de Lara, "Evaluation techniques for chatbot usability: A systematic mapping study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 11n12, pp. 1673–1702, 2019.

[23] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," in *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, 2022, pp. 27–43.

[24] S. Srivastava and T. Prabhakar, "A reference architecture for applications with conversational components," in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2019, pp. 1–5.

[25] M.-A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, 2016, pp. 928–931.

[26] ——, "Disrupting developer productivity one bot at a time," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 928–931. [Online]. Available: https://doi.org/10.1145/2950290.2983989

[27] M. Wessel, I. Steinmacher, I. Wiese, and M. A. Gerosa, "Should i stale or should i close? an analysis of a bot that closes abandoned issues and pull requests," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 2019, pp. 38–42.

[28] M. Wessel, I. Wiese, I. Steinmacher, and M. A. Gerosa, "Don't disturb me: Challenges of interacting with software bots on open source software projects," 2021.

[29] ——, "Don't disturb me: Challenges of interacting with software bots on open source software projects," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW2, pp. 1–21, 2021.

[30] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[31] M. Wyrich., R. Hebig., S. Wagner., and R. Scandariato., "Perception and acceptance of an autonomous refactoring bot," in *Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, INSTICC. SciTePress, 2020, pp. 303–310.