Model-Driven Development Using LLMs: The Case of ChatGPT

Virginia Niculescu[a], Maria-Camelia Chisăliţă-Creţu[b],
Cristina-Claudia Osman[c] and Adrian Sterca[d]
Babeş-Bolyai University, Cluj-Napoca, Romania

Keywords: Model-Driven Development, Large Language Models, Conceptual Diagrams, Business Process Model and Notation, Entity-Relationship Diagrams, User Productivity, Business Analysts.

Abstract: The recent rise of Large Language Models (LLMs) suggests the possibility for users with different levels of expertise to generate software applications from high-level specifications such as formatted text, diagrams or natural language. This would enhance productivity and make these activities accessible to users without a technical background. Approaches such as Model-Driven Engineering (MDE) and Workflow Management Systems (WfMSs) are widely used to enhance productivity and streamline software development through automation. This study explores the feasibility of using LLMs, specifically ChatGPT, in software development, focusing on their capability to assist business analysts (BAs) in generating functional applications. The goal of this paper is threefold: (1) to assess the extent to which LLMs comprehend conceptual model diagrams, (2) to evaluate the reliability of diagram-based code generation, and (3) to determine the level of technical knowledge required for users to achieve viable solutions. Our methodology evaluates the effectiveness of using LLMs to generate functional applications starting from BPMN process diagrams and Entity-Relationship (ER) diagrams. The findings provide insights into the reliability and limitations of LLMs in diagram-based software generation, the degree of technical expertise required, and the prospects for adopting LLMs as tools for BAs.

1 INTRODUCTION

A significant factor behind the difficulty of developing complex software is the wide conceptual gap between the problem and the implementation domains of discourse. Business analysts (BAs) are able to provide a good description of the problem; business processes can be represented as diagrammatic visualizations using standardized or non-standardized languages. In order to be analyzed and managed, business processes need representations that are formal, standardized and at the same time, easy to understand.

The Business Process Model and Notation (BPMN) represents such a representation and it is a standard managed by an international organization, the Object Management Group (OMG). Nowadays, practitioners use BPMN on a daily basis to design various complex business processes.

The term Model-Driven Engineering (MDE) is typically used to describe software development approaches in which abstract models of software systems are created and systematically transformed into concrete implementations (France and Rumpe, 2007). MDE is meant to increase productivity and simplify the process of the design and implementation.

Another approach that has the same goals of increasing productivity and at the same time ensuring simple and correct development of the business applications is represented by the Workflow Systems (WfSs). A Workflow Management System (WfMS) is a system that allows defining, managing and executing processes.

Large Language Models (LLMs) are a specialized subset of GenAI, focusing on language generation, while GenAI consists of a greater variety of AI models capable of creating various forms of content. Recent popularity of the LLMs leads to the idea that in a near future they could be used for developing software applications. This for sure will increase productivity and allow non-specialized IT users to create applications from specifications. The specifications of the software applications could be given in different format: formatted text, diagrams or even natural language. The general goal of our research is to study

[a] https://orcid.org/0000-0002-9981-0139
[b] https://orcid.org/0000-0002-1414-0202
[c] https://orcid.org/0000-0002-1414-0202
[d] https://orcid.org/0000-0002-5911-0269

Niculescu, V., Chisăliţă-Creţu, M.-C., Osman, C.-C. and Sterca, A.
Model-Driven Development Using LLMs: The Case of ChatGPT.
DOI: 10.5220/0013484400003928
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 328-339
ISBN: 978-989-758-742-9; ISSN: 2184-4895

how close we are from this stage.

Business processes are defined as a set of business activities that represent the steps required to achieve a business objective. They include the flow and use of information and resources (OMG, 2013). The most common modeling languages used in process representations are Event-driven Process Chains (EPCs) (Keller et al., 1992) and Business Process Model and Notation (BPMN) (OMG, 2013) diagrams.

Targeted users of our research are BAs who have to manage artefacts as business process diagrams that consists of meaningful actions performed in specific business areas. Often, BAs have to cope with changing requirements or defining new business processes that should be accommodated into the existing software. They may be asked to provide a proof of concept that helps the decision makers or to build a quick solution that can be extended or adapted later. In this context, LLMs become a tool at hand that may be employed to obtain results faster then other software development methodologies.

We aim to investigate whether LLMs may be used as effective tools to be employed in software development by specialists, i.e., BAs, that may not have technical knowledge in particular, as developers do have.

Considering BAs as the targeted users, a promising approach of using LLMs may be considered: to start by providing a minimal set of information comprised from just the conceptual models described using diagrams. The diagrams implicitly describe the processes and the functionalities of the application. An entity-relationship model (ER model) is also given in order to assure the correct generation of all entities involved. This approach is similar to the approach used by the WfMS. Based on all these, we may assume that the code sources of the application could be generated by AI.

ChatGPT was chosen as the LLM variant in our study. ChatGPT uses a complex machine learning model built on a Generative Pre-trained Transformer (GPT) architecture. The model is trained on a massive dataset of text from books, articles, websites, publicly available sources (Bala et al., 2025).

In the specific context of the proposed endeavour, the users (BAs) provide ER diagrams and BPMN process diagrams. So, we constrained the diagram analysis to these types of diagrams.

For a structured study approach we have identified three research questions, as follows:

- **RQ1** – To what extent does ChatGPT comprehend conceptual model diagrams?
- **RQ2** – To what extent diagram-based code generation is reliable?
- **RQ3** – How much technical knowledge should the user have in order to build a reliable solution?

The next section presents related works to our approach. Section 3 describes the methodology employed throughout the research and the evaluation results. The answer for each research question are included in this section after the results analysis of the corresponding experiments. Based on the obtained results, we present an incipient analysis of the possible directions to follow in achieving success in using LLMs in software development in Section 4. The paper concludes with the final remarks.

2 RELATED WORK

The application of LLMs in BPM semantic quality improvement is explored by (Ayad and Alsayoud, 2024). The paper investigates the extent to which GenAI technologies can aid the modeler by suggesting improvements. The study uses GPT-4o and examines its capabilities by employing various combinations of prompts, incorporating proposed textual syntax, and integrating contextual domain knowledge. The findings indicate that the knowledge generated by GPT-4o is predominantly generic, encompassing ambiguous and general concepts that extend beyond the specific domain. The use of specific proposed prompts helps to refine the generated knowledge, leading to more specific and comprehensive outcomes that align closely with the intended domain.

Kanuka et al. (Kanuka et al., 2023) explore the capabilities of LLMs, particularly OpenAI's ChatGPT, in addressing the challenges associated with software modeling, explicitly focusing on the bidirectional traceability problem between design models and code. The study aims to showcase the proficiency of ChatGPT in understanding and integrating specific requirements into design models and code. The paper investigates its potential to offer solutions to the bidirectional traceability problem through a case study. The findings indicate that ChatGPT is capable of generating design models and code from natural language requirements, thereby bridging the gap between these requirements and software modeling. ChatGPT has limitations in suggesting a specific method to resolve the problem, but exhibited the capacity to provide corrections to be consistent between design models and code.

The research conducted by (Rajbhoj et al., 2024) suggests that generative AI techniques have the potential to reduce the skill requirements necessary for software development and significantly accelerate the development process.

GitHub Co-pilot[1] is examined by Wermelinger (Wermelinger, 2023), focusing on the performance for generating code, tests, and details in offering support for students to solve computer science problems described as text. The study compares Copilot with OpenAI Davinci in terms of correctness, diversity, and guidance needed to obtain correct solutions. They reported that DaVinci demonstrated more effectiveness than Copilot regarding correctness and diversity.

Dakhel et al. (Dakhel et al., 2022) carried out an assessment of GitHub Copilot as an AI pair programmer. The researchers investigated the quality of the generated code compared to the human elaborated code considering a set of programming tasks. The results showed the Copilot is capable of providing solutions for almost all fundamental algorithmic problems, but some of them are buggy. When comparing Copilot to humans, the results indicate that the ratio of correct human solutions is greater than Copilot's correctness ratio, while the buggy solutions generated by Copilot require less effort to be repaired.

The study of Kamrul et al. (Siam et al., 2024) presents a thorough evaluation of leading programming assistants, including ChatGPT, Gemini (Bard AI), AlphaCode, and GitHub Copilot. The evaluation is based on tasks like natural language processing and code generation accuracy in different programming languages like Java, Python, and C++. The study offers a comparison of different LLMs and provides essential feedback on the rapidly changing area of AI models, emphasizing the need for ethical developmental practices to actualize AI models' full potential. Results indicate the strengths, weaknesses, and the importance of further modifications to increase the reliability and accuracy of the latest popular models.

Liukko et al. (Liukko et al., 2024) documents the development of a real life web application in the financial domain using ChatGPT. They adopt an Agile methodology in developing the software.

Dae-Kyoo Kim (Kim, 2024) compares OpenAI's ChatGPT and Google's Bard in developing a tour reservation web application. Both LLMs are used for producing various software development life cycle (SDLC) artifacts from a textual description of the application: generating the functional and non-functional requirements, domain modeling, and implementation. The author mentions that both models identified correctly many entities of the domain of the application, but also that they missed some of the entities. Still, they can not produce class diagrams or sequence diagrams for the the modeling phase. When it comes to code generation, the author mentions that ChatGPT and Bard produced errors of different kinds in their generated code, including missing import statements, missing modifiers, undefined variables, undefined data types, undefined methods, and parameter mismatches. When asked to correct these compile errors, both tools were able to correct most of the errors. The author concludes that both tools are helpful in developing an application, but they also have limitations.

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

3 METHODOLOGY AND EVALUATION

Our approach implies the use of GenAI tools in the SDLC process. Specifically, we intend to explore the usage of ChatGPT as LLM tool in the software development process. Moreover, we propose the use of BPMN models and ERDs in the development of the software product. Figure 1 represents the general approach of our study.

Our general endeavor is to evaluate the possibility to generate source code for a complex web application starting from business processes (depicted as BPMN models) and business domain concepts (depicted as ERDs). This analysis is split into three phases directed by the established research questions.
For each phase of the evaluation we set the following stages:

1. establish the evaluation criteria appropriate to the corresponding research question;
2. create the input data set for evaluation;
3. establish the prompts that allow us to evaluate the established criteria (the initial prompt may be adapted based on the responses);
4. analyse the results and extract conclusions.

Full details about prompts, data, and evaluation of studied diagrams could be accessed at the link available at the reference (Chisăliţă-Creţu et al., 2025).

RQ1: To What Extent Does ChatGPT Comprehend Conceptual Model Diagrams?

The first goal is to evaluate the ChatGPT's degree of comprehension of the used diagrams. We have evaluated two types of diagrams, namely ERDs and BPMN diagrams for processes.

Evaluation Criteria
The first goal in our study is to evaluate the ChatGPT's degree of comprehension of the various types

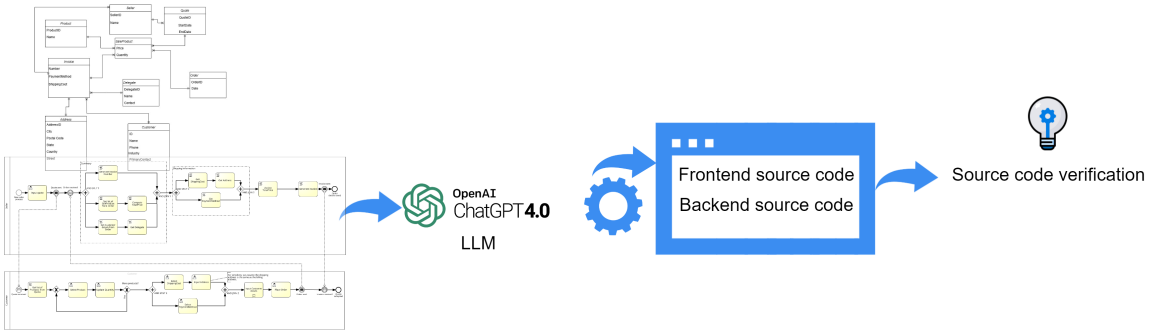

Figure 1: General approach.

of diagrams provided as input. The response generated is assessed based on the following criteria:

- ***C01:*** ChatGPT is able *to correctly identify* the type of the received diagram;
- ***C02:*** the extent to which ChatGPT is able *to correctly describe* the symbols and relationships included in the diagrams;
- ***C03:*** the extent to which ChatGPT is able *to correctly transform* the diagram into a persistent format through conversion to another artifact, e.g., database schema, XML format;
- ***C04:*** ChatGPT is able *to improve* the existing diagrams;
- ***C05:*** the extent to which ChatGPT is able *to generate* a diagram based on a persitent format (database schema, XML format);

Criteria ***C01*** and ***C04*** are evaluated with *yes* or *no*, while criteria ***C02***, ***C03***, and ***C05*** use the scale from 1 (low accuracy) to 5 (high accuracy). Criteria ***C03*** and ***C05*** are complementary.

ER Diagrams

Data

Five samples of ER diagrams were provided as input for ChatGPT to analyze. Two samples of ERDs do not include attributes. From the remaining ones, one ERD highlights primary keys (PKs) and foreign keys (FKs) using graphical symbols, one ERD uses the text acronyms PK and FK, while the last ERD does not include distinct marks for attributes or keys. Because it is more commonly used, we opted for Crow's Foot notation instead of Chen notation.

Prompts

The prompts given to ChatGPT are listed below and the evaluation of ***C01*** and ***C02*** use the same prompt.

P1.ERD: "Explain the uploaded diagram."

P2.ERD: "Provide the database schema for the uploaded ERD."

P3.ERD: "Improve the database schema with new entities, attributes, and relationships between entities."

P4.ERD: "Recreate the ERD for the improved database schema. Please use the same type of diagram as the uploaded one, including the crow's foot notation."

Results Analysis

For ERDs, the responses include details about the type of the diagram, entities, attributes, and relationships. Additional data produced refers to generated database schema. A third type of data generated includes the improvements for the database schema, in terms of new entities, attributes, and relationships. At the end, ChatGPT generates a diagram that should represent the improved database schema.

The prompt **P1.ERD** requested to explain the uploaded diagrams. The responses offered by ChatGPT indicate that it manages to identify correctly the type of the diagram. Therefore, criterion ***C01*** is evaluated with *yes* for all five recorded experiments, explicitly saying that an ERD was uploaded. Other not recorded experiments indicate that ChatGPT answered that the processed ERD is a *concept diagram* and the user needed to ask ChatGPT to clarify what type of concept diagram actually is it. For this case, the responses indicate that ChatGPT successfully identifies the diagram type only if additional prompts that refine the response are present.

P1.ERD prompt responses allowed the evaluation of the ***C02*** criterion, too. ChatGPT manages to correctly describe the ERDs, identifying and detailing entities, attributes, and relationships. The ***C02*** criterion is evaluated with *5* for all recorded experiments. For the cases where the ERDs do not specify attributes, ChatGPT offered details about the entities and relationships only, but offered information about the cardinality, usage, and purpose of the diagram.

The prompt **P2.ERD** requested *to generate the database schema for the uploaded ERD*. Almost all generated database schema using SQL statements meet partially the initial ERDs. For instance, ERD1 and ERD2 do not specify attributes. Still, ChatGPT

adds attributes to the existing entities and other new entities, considered bridge tables to manage ***m:n*** relationships. ***CO3*** criterion in these cases is evaluated to *4* because of the improvements suggested for the missing attributes. ERD3 includes attributes, emphasizes crow's foot notation, and primary and foreign keys. Still, ChatGPT adds two new relationships between the existing entities. For the ERD3 sample and this specific result, the ***CO3*** criterion is evaluated to *3*, as ChatGPT does not follow the details mentioned in the diagram. ERD4 presents graphical symbols for primary keys, foreign keys, and the crow's foot notation. During the experiment for ERD4, ChatGPT successfully manages to generate the database schema that entirely mirrors the provided ERD and the ***CO3*** criterion is evaluated to *5*. All database schema provided are normalized to 3NF.

The prompt **P3.ERD** requested *to improve the database schema with new entities, attributes, and relationships*. For all experiments, the data generated indicate improvements and the ***C04*** criterion was evaluated to *yes*. For some cases, the initial entities were replaced with new entities and relationships that ChatGPT has evaluated as offering a better perspective for the application domain through flexibility and adaptability.

The prompt **P4.ERD** requested *to recreate the ERD for the improved database schema*. The prompt explicitly requested to use the crow's foot notation, but the generated diagrams do not follow the standard notation for ERD. The ***C05*** criterion was evaluated to *1* for all run experiments. ChatGPT did not suggest using particular tools that could successfully generate the ERD based on the provided database schema in any recorded experiments.

Table 1 summarizes the results of the recorded experiments on ERDs together with the evaluation for the five used criteria.

Table 1: Experimental results for ERDs.

ERD	***C01***	***C02***	***C03***	***C04***	***C05***
ERD1, no attr.	yes	5	4	yes	1
ERD2, no attr.	yes	5	4	yes	1
ERD3, with attr.	yes	5	3	yes	1
ERD4, with attr	yes	5	5	yes	1
ERD5, with attr.	yes	5	3	yes	1

BPMN Diagrams

Data

We have used five BPMN diagrams [2].

- BPMN1 – collaboration diagram with different gateways types and annotations (asynchronous communication),
- BPMN2 – collaboration diagram with exclusive gateways (synchronous communication),
- BPMN3 – process diagram with inclusive, exclusive and parallel gateways,
- BPMN4 – collaboration diagram with event-based gateways and different intermediate events,
- BPMN5 – process diagram with exclusive and event-based gateways.

Prompts

P1.BPMN: "Explain the uploaded diagram."

P2.BPMN: "Generate the XML-file (*.bpmn) corresponding to the uploaded diagram."

P3.BPMN: "Improve the uploaded diagram by including new elements (tasks, events, gateways, data objects, swimlanes etc. - or any other BPMN concept)."

P4.BPMN: "Recreate the BPMN model for the improved BPMN model. Use BPMN specification: https://www.omg.org/spec/BPMN."

Results Analysis

Our evaluation criteria are consistent with those previously employed for the assessment of ERDs, while incorporating necessary adaptations to ensure their applicability within the BPMN context. The first criterion ***C01*** evaluates the ability of ChatGPT to identify the diagram type (process diagram modeled using BPMN). The next criterion ***C02*** refers to the ability of ChatGPT to correctly describe the flow of the BPMN diagrams (namely, to identify the BPMN symbols and the corresponding relationships between them). The prompt used for the assessment of the first two criteria is **P1.BPMN**. The third criterion ***C03***, evaluates the ability of ChatGPT to extract from the diagrammatic visualization of a BPMN model, the corresponding XML representation (*.bpmn file). The prompt used for the assessment of this criterion is **P2.BPMN**. Criterion ***C04*** evaluates the capacity to bring improvements for the uploaded BPMN diagram. The improvements may include the incorporation of new BPMN symbols (such as tasks, gateways, data objects, etc.). The corresponding prompt of the third criterion is **P3.BPMN**. Through the fourth prompt **P4.BPMN** we assess the ability of ChatGPT to produce a BPMN model that reflects the previously proposed improvements.

The analysis of BPMN diagrams demonstrates ChatGPT's accurate analysis of BPMN process models, aligning with the diagrams' symbol usage, successfully fulfilling criteria ***C01*** and ***C02***. Criteria

[2]https://github.com/camunda/bpmn-for-research/tree/master/BPMN\%20for\%20Research/English

Table 2: Experimental results for BPMN diagrams

BPMN	*C01*	*C02*	*C03*	*C04*	*C05*
BPMN1	yes	5	2	yes	1
BPMN2	yes	5	2	yes	1
BPMN3	yes	5	2	yes	1
BPMN4	yes	5	2	yes	1
BPMN5	yes	5	2	yes	1

C03 and ***C05*** are partially met as although ChatGPT proposes new concepts to be added on the diagram (usually it proposes the incorporation of tasks, gateways, data objects, intermediate events, swimlanes, sub-processes, text annotations, message flows, etc.), the XML file that is generated does not comply with BPMN specification (OMG, 2013). The response for **P3.BPMN** is usually the XML format of the improved model with additional narrative explanations related to the proposed updates. The *.bpmn file cannot be read by BPMN editors like SAP Signavio or bpmn.io, Bizagi being one of the BPMN modeler tools able to partially interpret the XML files (but the visualization provided overlaps the identified BPMN elements). By providing additional details in the prompt (for example: "The position of the symbols is missing from the XML file and the elements are overlapped, can you please update the XML file?" or "The same position of the symbols is provided for the elements from the XML file and the elements are overlapped, can you please update the XML file?"), the XML file is improved, it still fails to accurately represent a machine-readable BPMN model. Consequently, a well-defined sequence of prompts is required for ChatGPT to effectively fulfill the initial task (therefore, ChatGPT users should enhance their prompt engineering skills).

Nevertheless, criterion ***C05*** is minimally fulfilled. The provided visualization deviates from the BPMN standard. In some situations it reveals the Python source code used in order to provide the graphical visualization (see **P4.1** for BPMN5, reachable at the link from the reference (Chisăliță-Creţu et al., 2025)). Still, it cannot provide standardized diagrams, neither if the XML structure (for example the XML corresponding file of a BPMN diagram) is provided.

Table 2 summarizes the results of the recorded experiments on BPMN models, together with the evaluation for the five used criteria.

Answer to RQ1:
From all these experiments we may conclude that ChatGPT is able to understand the ERDs, the entities and their relations being extracted from the images. The database schema is not accurately generated as some relations are not defined at all, or they are not appropriately defined. Also, it is possible that new tables, attributes, and relationships to be added.
Requested improvements show that ChatGPT is able to add new entities, attributes, and relationships based on the deduced data domain represented by the diagram. Reversely, ChatGPT is not able to generate ERDs following domain-specific modeling notation as dedicated standalone tools are able to.
The BPMN model analysis reflects a good interpretation of the diagrams, but shows that a well-defined sequence of prompts is required for ChatGPT to effectively fulfill the transformation into the XML persistent format.
ChatGPT is not able to produce standardized diagrams, neither if the entire XML structure of the BPMN diagram is provided.
We may conclude that the interpretation and transformation work executed by ChatGPT for the analysed diagram types is very useful since the number of inaccuracies is low in percentage, and with additional further verification and prompts the correct solution could be obtained.

RQ2: To What Extent Diagram Based Code Generation Is Reliable?

The second goal of our study is to assess the extent to which the generated code by ChatGPT is able to fit the real needs of the BAs from various perspectives.

To assess this capability, we conduct a case study on developing an application using some of the most widely adopted technologies in software development, including React for the frontend, Spring for the backend, and MySQL for the database.

The considered application is a sale web application that involves two types of actors: seller and customer. The seller prepares a quote and sends it to the customer. Using the quote, the customer selects the desired products. The customer then provides additional information, such as the shipping cost, the address, and the payment method. After entering customer details, described in the data model as Name, Primary Contact, Phone and Industry, the customer creates and submits the order to the seller. Upon receiving the order, the seller generates an invoice number. Using the order details, the total price is calculated, and the delegate is included in the invoice. The shipping and the payment methods are also incorporated based on the order information. The seller adds the shipping address details, updates the total price, generates the invoice and sends it to the customer.

Evaluation Criteria

- ***C06:*** the extent to which ChatGPT is able to generate code consistent with the diagrams provided by the user;
- ***C07:*** the extent to which ChatGPT is able to generate an entire application;
- ***C08:*** the extent to which ChatGPT is able to improve the generated code as a consequence of using additional prompts (clarification);
- ***C09:*** the extent to which ChatGPT is able to integrate new requirements in the previously generated code;
- ***C10:*** the extent to which ChatGPT is able to guide the user regarding the platform prerequisites for successful deploy of the generated application.

Data

For this application we consider that the user (BA) will provide only two diagrams. From the business point of view they include enough information about the entities and the processes into which they are involved:

- an ER diagram that describes the main concepts together with their attributes and relations between them (Fig. 2);
- a BMPN diagram that describes the main functionalities of the application (Fig. 3).

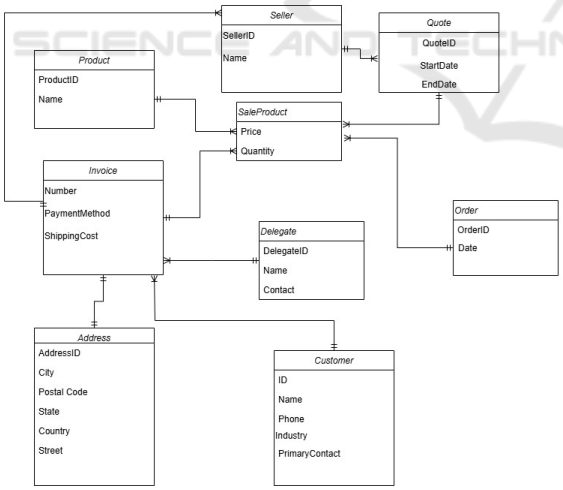

Figure 2: ER Diagram for the Sale web application.

Prompts and Results Analysis

For this goal of evaluating the code generation, we started from an initial prompt and following prompts were given depending on the analysis of the responses.

All-in-One Approach

Since we intend to find out if the application that respects the requirements implicitly given in the diagrams, we tried with an initial prompt that asked ChatGTP upfront to generate all the code files for the web application, both the frontend part and the backend.

Initial Prompt:
"Develop a web application using the generated database and the BPMN process from the uploaded picture. The web application should use a MySQL database and should be made from a React frontend and a Java backend. Please pack everything together as a downloadable zip archive."

Response:
"Creating a web application with the described architecture involves the following steps:
1. Database Setup: Set up the MySQL database with the previously generated schema and data. 2. Backend Development: Build a Java Spring Boot backend that connects to the database and exposes RESTful APIs for the BPMN processes.
3. Frontend Development: Create a React frontend that interacts with the backend to follow the BPMN workflow.
4. Packaging: Organize the application as an IntelliJ IDEA project and package it as a downloadable zip archive.
I will prepare the required files and structure the project. This process may take some time. The web application project has been successfully created and packaged as a zip archive. You can download it using the link below:
Download ProductManagementWebApp.zip "

The provided zip archive contains just the main SpringBoot class of the application for the backend and a simple Javascript skeleton file for the frontend. It can be noticed that the response includes short description about the structure of the application, but it suggests that the entire application was built, which it not correct.

During the next attempt, we gave ChatGPT the same prompt but without the requirement of packaging all code in a zip archive. This time it generated more relevant code, different classes for the Java backend and .jsx files for the React frontend application. Although the code was good and more complete, it still had many inconsistencies between different code files (e.g. objects that call methods which does not exist). We had to spend a lot of time to manually correct the code so that it would compile and finally it would also run correctly.

Iterative Approach

Analysing the results of the first two attempts lead to the conclusion that the strategy should be changed and for the third attempt we adopted the following iterative-based development strategy: we would ask

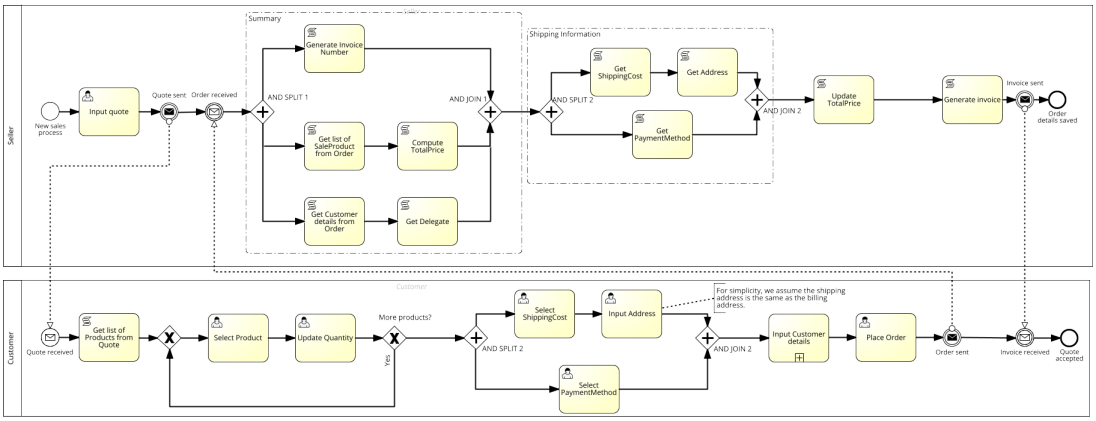

Figure 3: BPMN diagram of the main processes of the Sale application.

ChatGPT to generate small chunks of code and an experienced software developer (i.e. one of the authors of the paper) would review the code and correct it (and sometimes even gave ChatGPT back the corrected code as input) so that errors don't propagate in the development process. In the remaining lines of this subsection we document the interaction with ChatGPT in this third attempt methodology of code development assisted by ChatGPT.

For this approach we start with a prompt that uploaded the BPMN process diagram and asked ChatGPT:

"I want to implement the above BPMN process in a web application with React frontend and Java Springboot backend. First, please analyze the BPMN diagram and tell me what do you understand from it."

After we verified that the provided interpretation of the diagram was correct, we gave it the ERD image with the database concepts and asked:

"Please generate SQL scripts for creating a Mysql database for the above BPMN process. The conceptual diagram of the database is depicted in the attached image."

The generated SQL scripts were mainly correct but they also had problems. Three out of nine relations in the diagram were not recognized by ChatGPT and some attributes wrongly identified; on the plus side, ChatGPT correctly identifies most of the foreign keys.

We then corrected the SQL scripts and instructed ChatGPT to use the corrected ones in the rest of the dialogue. Following, we asked ChatGPT to generate sample data for the database which we used in order to populate the database - they required minor corrections.

Next, we started to implement the backend API service. The prompt used was this:

"Generate Java code for the backend REST API implementation: generate JPA entity classes, service classes, repository classes, API Endpoints for the above database tables."

ChatGPT generated classes for the entities, service, JPA repository, controller, but it generated only two for each category. It also generated the JPA Hibernate configuration. These classes contained correct code.

It was necessary to explicitly ask ChatGPT to generate all the entities. As a result these were generated but there were some inconsistencies. For instance, some of them were placed on a different package then the previous generated entities (i.e. `model.*` instead of `entity.*`), so we had to manually verify and correct them.

During the following iterations, we explicitly asked ChatGPT to generate:

- the rest of the JPA repository interfaces,
- the rest of the service classes and
- the rest of the REST controllers.

It was necessary to review each generated class files, some of them requiring minor corrections related to identifier inconsistencies across different files.

Many of the entity class files required additional annotations like `@Column(name="paymentmethod")` for a `private String paymentMethod` member so that the Hibernate SQL querries would work with the database structure previously generated by ChatGPT (otherwise, the attribute name would be converted by default to the column `payment_method` which did not existed in the database). We corrected all these manually.

The controllers were correct, except for the fact they did not have the CORS (Cross-Origin Resource Sharing) security setting setup (in order to be accesible from the React server.

In addition, some of the required service or repository classes were not generated at all, so we had to remind ChatGPT to generate them like in the following prompt:

"You forgot to generate the class com.example.productsale.service.ProductService."

As a result the missing classes have been generated.

We then asked ChatGPT to generate the Gradle build file and OpenAPI documentation for the generated REST API.

Finally, for the backend side, we asked ChatGPT:

"Please generate the backend code that implements the BPMN process depicted in the diagram that I uploaded in the beginning of this chat."

In the first response, ChatGPT generated just skeleton files for Camunda without being specific to the BPMN process given by us, but after we insisted, it generated the required backend files for the BPMN process.

The service class had many inconsistencies (i.e. calls of method that did not exist, inexistent attributes), so eventually we prompted ChatGPT:

"Please look again at the updated database structure (that I have given you previously as SQL statements). The implementation of OrderProcessService class you have provided me is incorrect, it does not use the database structure I mentioned. Can you generate again the OrderProcessService class?"

At this point ChatGPT provided two code implementations of the same class and asked us to choose one. After we chosen one, there were still some inconsistencies that we corrected manually.

We were then ready to move to the generation of the React frontend app. We used the prompt:

"Can you generate the REACT frontend for the previously generated backend service ?"

There were still some inconsistencies in the generated frontend code that we had to manually correct.

The deployment phase was entirely our attribution. Everything had to be placed together configured and set. The positive aspect is the fact that after we deployed the frontend and backend applications, the web application worked correctly and implemented the functionality of the BPMN process given in the diagram.

Summarizing, ChatGPT generated approximately 1000 lines of code for the REACT frontend and 1300 lines of code for the Java SpringBoot backend.

ChatGPT service was of real help and it managed to generate a web application from a BPMN process diagram and a conceptual database diagram. But the application would not have worked correctly without an experienced software developer reviewing and correcting the generated code (criterion ***C06***). We have found that ChatGPT does not generate a complete application all at once and the best methodology in our experience was an iterative one with code inspection and review after each round (criterion ***C07***). Related to criterion ***C08***, we evaluate that ChatGPT is very efficient in correcting itself following additional clarifications from the user, although sometimes it offers two possible code solutions and asks the user to choose one. The least efficiency of ChatGPT was related to packaging the application and deploying it - the problem was more proeminent in the backend application, and integrating different code files previously generated (criteria ***C09*** and ***C10***).

Answer to RQ2: *We may conclude that ChatGPT is not able to directly create a software application in one single step: an iterative approach is necessary. During each iteration the tasks should be clear, explicitly given and focused on a single responsibility. Each iteration needs verification and corrections, but with additional clarifications from the user ChatGPT could efficiently correct itself. The most important aspect is the fact that the final goal – building a reliable software application – is reached. The effort needed for this software construction is considerably reduced and, in addition, the knowledge support offered by ChatGPT is important since at each step it produce also explanation together with the generated code.*

RQ3: How Much Knowledge Should the User Have in Order to Reach a Solution?

The third goal of our research refers to the human user employing ChatGPT in his work. Therefore, we examine the amount of technical knowledge needed to use ChatGPT to successfully obtain the desired solution. The inquiry addresses the several aspects, as follows:

Evaluation Criteria

- ***C11:*** The level of technical knowledge required for a user to be able to give the effective prompts and to understand the generated responses.
- ***C12:*** The level of technical knowledge required for the user to be able to aggregate all the received source code files, to configure the platform, and to deploy it to produce the fully working application.

For this research question we have used the same data, prompts and responses used for the research question **RQ2**.

Results Analysis

The first plain observation is that the user should understand the diagrams that are included into the first prompt. This is an obvious observation since we assume that the targeted users are BAs for which these kinds of diagrams represent the common knowledge.

As it may be seen from the first initial prompt of **RQ2**, information about the technologies are required. This entails knowledge regarding these technologies at least at the level of knowing about their purpose and their context of usage. This means that the BA should be familiar to the corresponding terminology.

From the first response we can notice that the information about the structure of the application and its main components are given. Again, fundamentals regarding these should be known in order to be able to understand the ChatGPT's response. This leads to the conclusion that for the criterion ***C11*** the required level of technical knowledge includes fundamentals related to design and technologies used in the business process management and software application development.

Since Chat GPT was not able to provide the entire code from the beginning, the user should be able to iteratively ask explicitly for different components to be implemented and integrated. During this phase some information about other technologies that ChatGPT chooses to use are given. These should be understood by the user as well.

On the other hand, if some mistake are given by the user, ChatGPT could emphasizes it and suggest correction. For example, in the experiment we have used an entity with the name 'Order' and ChatGPT emphasized the fact that it is a SQL keyword and should be changed. This could give valuable help to the user.

Through iterative requests all the source code was obtained, but it was the user responsibility to manually extract the code and introduce it into a local project. The generated code contains errors, some inaccuracies and inconsistencies. To correct all these, advanced knowledge in software development was necessary. In addition, the solution may involve particular project configuration and settings.

Providing the solution, ChatGPT chose to use diverse set of frameworks and libraries (e.g. Flask, Spring, Axios or Fetch API). All these additional tools have to be installed, and the necessary configuration to be set.

Finally, ChatGPT was asked to give all the necessary information needed to reach a deployable application. Even if the response includes several details about the necessary steps and guidelines, they could not be successfully executed without appropriate knowledge and experience.

So, we may conclude that for the criterion ***C12*** we need an expert user with advanced technical knowledge.

Answer to RQ3: *The experiments emphasize that the user should know at least the fundamentals knowledge such that to be able to give the correct prompts and to understand the received responses. For achieving a complete functional application the user should be an experimented advanced developer. At the current stage, ChatGPT could represent more an assistant tool in software development rather than an instrument able to create concrete, functional software application for the BAs. This imposes high level of user knowledge in order to allow obtaining functional applications.*

Threats to Validity

For the previous analysis we have used ChatGPT-4o free version. Additionally, paid plans may provide better results. ChatGPT-o1 model is trained with a large-scale reinforcement learning algorithm that provides responses using Chain-of-Thought (CoT). Therefore, GPT-o1 is reported to have longer response time than GPT-4o (mini) that may affect the UX. ChatGPT-o3 mini is fast for advanced reasoning. Currently, free plan users may send 50 requests every three hours, which are reduced when pictures and diagrams should be analysed. Paid plans users do not have any issues of unavailability, even during peak hours. This implies that there is the possibility that we didn't obtain the best possible responses. During the experiments, we did not encounter disruptions with ChatGPT's functionality, but rather limitations on the number of requests allowed.

Not customizing ChatGPT may result in increased number of prompts that should be provided in order to obtain the desired results.

The computational effort that is needed to analyze and generate diagrams resulted in constraints when finishing the evaluation in single working session. Multiple ChatGPT sessions were required to generate the dataset for **RQ1**.

In addition, we noticed that there is quite a high level of non-determinism in obtaining the responses. For the same user or not, giving to ChatGPT the same prompt, different responses are generated. This means that the quality of the responses is not always the same.

4 DIRECTIONS OF LLMs INTEGRATION IN SDLC

Based on the previous analysis we may derive also some directions related to the integration of LLMs into the software development life cycle.

Software Development Life Cycle
SDLC methodologies provide a systematic management framework based on stages with specific deliverables of the software development process. Some of the common SDLC stages are: Plan, Design, Implement, Test, Deploy and Maintain. Following a SDLC methodology assures improvement of the following aspects: estimation, planning, and scheduling; risk management and cost estimation; software delivery and customer satisfaction; and visibility of the development process for all involved stakeholders.

SDLC has shifted from Waterfall models (Royce, 1970), Iterative models, V-Models (Hill, 1996), Spiral models (Boehm, 1988) or Model Driven Architecture (MDA) (OMG, 2001) to agile methodologies (Extreme Programming (XP) (Beck, 1999), Scrum (Schwaber and Beedle, 2001) or Kanban (Anderson, 2012)).

Among these we may identify two of them that may benefit the most of allowing a LLMs to be used as an actor involved in the development process.

The Iterative SDLC model presented in Fig 4, stands out as a flexible and efficient methodology that promotes continuous improvement and adaptability. The key principles of this are: Incremental Progress, Flexibility and Adaptability, Continuous Evaluation, and Risk Management.

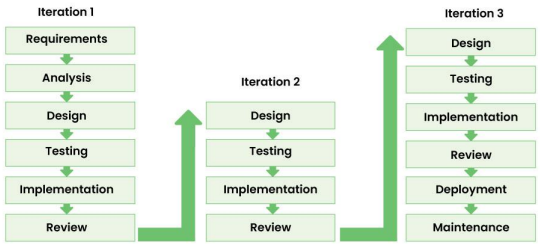

Figure 4: Iterative Model.
(source:https://www.geeksforgeeks.org/sdlc-models-types-phases-use)

Agile SDLC approach is described in Fig 5. The key principles of this model are: Iterative and Incremental Development, Customer Collaboration, Adaptability to Change, and Cross-Functional Teams.

Both of them are based on iterations in the development process. This is important since a LLM actor managed well with small and well defined tasks.

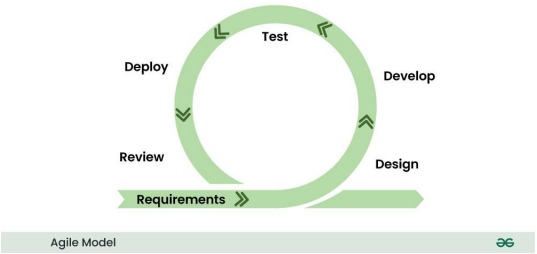

Figure 5: Agile Model.
(source:https://www.geeksforgeeks.org/sdlc-models-types-phases-use)

Comparison with Model-Driven Development
Model-driven development (MDD) approach is meant to increase productivity by using standardized models, simplifying the process of design via models of recurring design patterns in the application domain. It promotes communication between working individuals and teams by using standard terminology and recognized best practices.

At their foundation, LLMs are also working by identifying and generating models. So, they could provide information organized and recognized models as well. Diagrams or programming patterns are examples of such models from the domain of programming.

The conceptual diagrams are quite well handled by LLMs (e.g. ChatGPT). Even if the results are not perfect they provide interpretation and transformation at a high level of accuracy and could become a very useful actor in MDD approach.

Comparison with Workflow Systems Approach
Workflow systems allow secure and productive software development that starts from process diagrams, too. They provide a very high level of abstraction interaction with the user and provide automatic development of the software based on specifications provided through the workflows (processes). Obtaining the final application doesn't imply expert level of technical knowledge – it is automatically done based on a pre-existant generic implementation which is represented by the workflow engine.

They efficiently cover exactly the part that it is difficult to obtain from the interaction with a LLM actor. Still, in this case the obtained application is not an independent one and could be executed only inside the chosen workflow management system.

5 CONCLUSIONS

We conducted a research that aimed to evaluate the extent to which LLMs (e.g. ChatGPT) could be used

for the development of an application prototype starting from conceptual diagrams as ER and BPMN process diagram. This would be very useful especially from a business analyst point of view that usually starts by defining these kind of diagrams.

The research questions structure the research on three directions: diagram interpretation and management, source code generation, and user necessary knowledge. The results of the human user interaction with ChatGPT were documented and several criteria are formulated for each research question. Various result types obtained were evaluated, e.g., explanation, transformation, improvements, code, offered support.

From the conducted experiments, we may conclude that, at this phase, ChatGPT can be used much more as an assistant tool in developing software application than as a reliable developer. Since it provides very good results for small and very well specified tasks, it may be included as an assistant actor in an iterative or agile software development approaches.

As further work we propose to repeat the experiments using other LLMs, as Gemini for example, or using ChatGPT-Plus. Another investigation direction would be to modify the initial problem such that to contain not only diagrams but also descriptive functional requirements.

REFERENCES

Anderson, D. J. (2012). *Lessons in agile management: On the road to Kanban*. Blue Hole Press.

Ayad, S. and Alsayoud, F. (2024). *Exploring ChatGPT Prompt Engineering for Business Process Models Semantic Quality Improvement*, pages 412–422.

Bala, S., Sahling, K., Haase, J., and Mendling, J. (2025). Chatgpt for tailoring software documentation for managers and developers. In *International Conference on Agile Software Development*, pages 103–109. Springer.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32(10):70–77.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5):61–72.

Chisăliţă-Creţu, M. C., Osman, C., Sterca, A., and Niculescu, V. (2025). ChatGPT response collection used for evaluation. https://figshare.com/s/3bdcd6a7686a0ce20610.

Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M., and Ming Jiang, Z. (2022). *GitHub Copilot AI pair programmer: Asset or Liability?*

France, R. and Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE '07)*, pages 37–54.

Hill, D. R. (1996). *Object-Oriented Analysis and Simulation*. Addison-Wesley Longman Publishing Co., Inc., USA.

Kanuka, H., Koreki, G., Soga, R., and Nishikawa, K. (2023). Exploring the ChatGPT approach for bidirectional traceability problem between design models and code.

Keller, G., Scheer, A.-W., and Nüttgens, M. (1992). *Semantische Prozeßmodellierung auf der Grundlage" Ereignisgesteuerter Prozeßketten (EPK)"*. Inst. für Wirtschaftsinformatik.

Kim, D.-K. (2024). *Comparing Proficiency of ChatGPT and Bard in Software Development*, pages 25–51. Springer Nature Switzerland, Cham.

Liukko, V., Knappe, A., Anttila, T., Hakala, J., Ketola, J., Lahtinen, D., Poranen, T., Ritala, T.-M., Setälä, M., Hämäläinen, H., and Abrahamsson, P. (2024). *ChatGPT as a Full-Stack Web Developer*, pages 197–215. Springer Nature.

OMG (2001). Model driven architecture (MDA). https://www.omg.org/cgi-bin/doc?ormsc/01-07-01.pdf.

OMG (2013). Business Process Model and Notation (BPMN) Specification, Version 2.0.2. https://www.omg.org/spec/BPMN/2.0.2/.

Rajbhoj, A., Somase, A., Kulkarni, P., and Kulkarni, V. (2024). Accelerating software development using generative ai: Chatgpt case study. In *Proceedings of the 17th Innovations in Software Engineering Conference*, pages 1–11.

Royce, W. (1970). Managing the development of large systems: Concepts and techniques. In *9th International Conference on Software Engineering. ACM*, pages 328–38.

Schwaber, K. and Beedle, M. (2001). *Agile software development with Scrum*. Prentice Hall PTR.

Siam, M. K., Gu, H., and Cheng, J. (2024). *Programming with AI: Evaluating ChatGPT, Gemini, AlphaCode, and GitHub Copilot for Programmers*.

Wermelinger, M. (2023). Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, page 172–178, New York, NY, USA. Association for Computing Machinery.