Requirement Analysis Approach to Estimate the Possibility of Software Development Artifacts Reusing Consulting with Artificial Intelligence Technologies

Olena Chebanyuk [1,2]

[1] *Institut d'Investigació en Intel·ligència Artificial, Campus Universitat Autònoma Barcelona Carrer de Can Planas, Zona 2, 08193 Bellaterra, Barcelona, Catalonia (Spain)*

[2] *National Aviation University, 1, Liubomyra Huzara ave., 03058, Kyiv, Ukraine*

Abstract

This paper proposes a domain engineering approach to estimate the possibility of reusing software development artifacts with the consultation of artificial intelligence technologies. Estimation is done by means of comparing the semantics of software artifacts with the semantics of the requirement specification. The approach is based on the following ideas: two lists of user stories are formed where the first list contains the user stories of software artifacts, while the second list contains user stories of the future project. Then, the semantics of these two lists are compared. Software components with the same semantics from both lists of user stories are marked as possible candidates for reuse. The requirements engineer manages the artificial intelligence that performs all routine tasks for this approach. As a result of these activities, a software developer receives recommendations on which software components need to be reviewed for reuse.

The proposed approach addresses one more challenge of using artificial intelligence technologies to generate software models, represented as UML diagrams, despite the limitation that artificial intelligence tools may only generate text answers. The choice of an environment for UML diagram visualization in the case of communication with artificial technologies is grounded.

The paper also considers questions about the effectiveness of using different natural language groups, namely Germanic, Romance, and Cyrillic, to support communication with artificial intelligence technologies.

Keywords:

Requirement Analysis, AGILE, PlantUML, Artificial Intelligence, Domain Engineering, Model-Driven Engineering

1. Introduction

Artificial Intelligence (AI) technologies open up new opportunities to address current challenges in software engineering. One such challenge is the semantic analysis of software artifacts to support the software development lifecycle processes within the Agile methodology. Semantic comparison increases the accuracy of software reuse procedures.

This paper initiates a series of works that leverage the fundamentals of Model-Driven Engineering as a foundation for approaches supporting AI-powered software development lifecycle processes.

This work focuses on describing an approach for evaluating the feasibility of reusing software development artifacts using AI technologies.

According to ISO/IEC/IEEE 24765:2017 standard, a software development artifact is any artifact related to the software development process. Examples include UML diagrams, interface screenshots, *.dll files, and test cases [1].

114th International Scientific and Practical Conference from Programming UkrPROG'2024, May 14-15, 2024, Kyiv, Ukraine

* Corresponding author.

elena.chebanyuk@iiia.csic.es (O. Chebanyuk)

0000-0002-9873-6010 (O. Chebanyuk)

The same standard defines a software artifact as any type of software, such as source code, *.dll files, frameworks used in a project, or executable code [1].

Artificial intelligence technologies are envisioned to handle routine tasks of semantic analysis for various software development artifacts, with a human subsequently verifying the results. This paper considers Agile requirements analysis within the software product line approach.

A core principle of Agile is to adapt to changing customer requirements (as outlined in the Agile Manifesto). Changing requirements necessitate a sequence of actions to update the associated software development artifacts. This involves requirements verification, validation, and elicitation (review and editing of software models and requirements specifications). These activities consume a significant portion of the time spent on requirements analysis.

AI technologies can take on routine tasks, reducing the time required for artifact analysis and minimizing human error when updating data and making recommendations for reusing existing software artifacts in new projects. Semantic comparison of different user stories falls under the umbrella of semantic search. The main challenge of the semantic search procedure lies in representing various software development artifacts in a format that enables exact comparisons. Following the semantic search procedure, the requirements engineer will need to verify the results of the requirements analysis, while the developer will only need to evaluate the AI's recommendations.

2. Literature Review

Research investigating the principles of communication between users and AI technologies (e.g., chatbots) can be organized according to the steps involved in supporting the user-AI tool communication process. The schema for organizing a conversation between a user and an AI tool is represented in Figure 1.

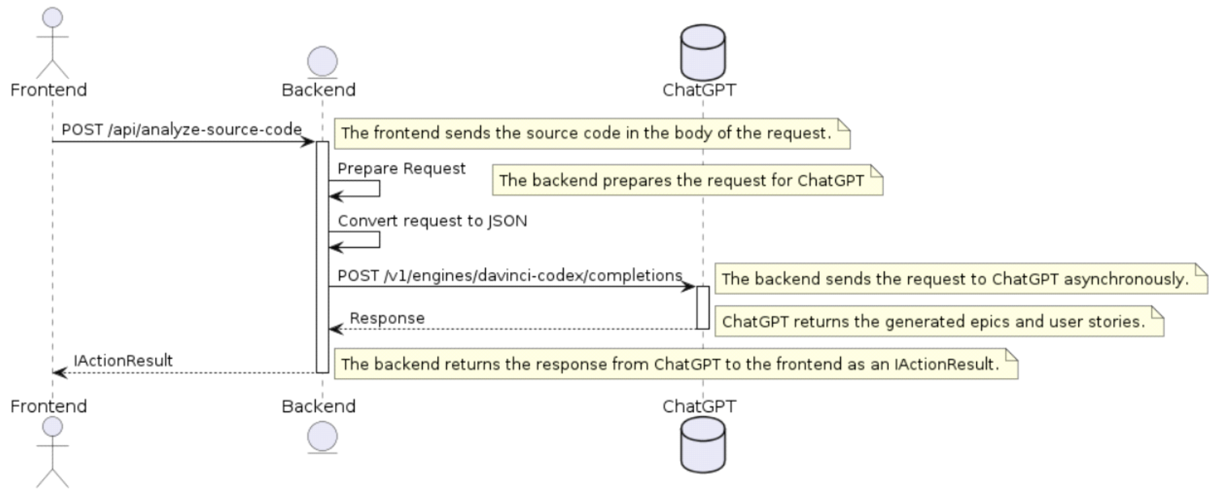

Frontend
Backend
ChatGPT
POST /api/analyze-source-code
The frontend sends the source code in the body of the request.
Prepare Request
The backend prepares the request for ChatGPT
Convert request to JSON
POST /v1/engines/davinci-codex/completions
The backend sends the request to ChatGPT asynchronously.
Response
ChatGPT returns the generated epics and user stories.
IActionResult
The backend returns the response from ChatGPT to the frontend as an IActionResult.
Frontend
Backend
ChatGPT

Figure 1: Representation of an algorithm of Organization of conversation between user and AI tool. (ChatGTP is considered).

Here are the main steps involved in organizing conversation support between a user and an AI tool:

1. Sending request to chatbot (the first four messages in Figure 1).
2. Processing a user request and extracting semantics from it.
3. Searching for an answer to the question that corresponds to the defined semantics.
4. Returning the answer to the client side and interpreting it (the last message on Figure 1).

Consider review of papers representing results about the main steps of described algorithm.

2.1. Processing User Request and Extracting Semantics of the User’s Query

Formal query creation methods for knowledge bases are based on a series of checks performed on the user's natural language query. These checks may include the presence of question words, subordinating conjunctions, semantically colored marker verbs, subject and predicate groups, etc., in the input phrase. The program uses the set of results compiled after passing these checks to automatically construct a formal query from template blocks. Named entities found in the phrase become input parameters for the queries created in this way [2].

Widely used knowledge systematization approaches utilize ontologies to organize information about the problem domain. Ontological knowledge bases are effective for systematizing large amounts of knowledge, particularly in the development of natural language dialogue systems. Today, there is widespread development of various approaches for information systematization in the medical field. Examples include the medical rehabilitation support system presented in [3] and medical diagnostic software systems [4].

In developing reference dialogue systems, neural network approaches can be combined with the usage of ontological knowledge bases, thereby increasing each other's effectiveness. So, for example, as described in [4], the ontology stores marked fragments of texts that are extracted using queries formed on the basis of user phrase data. These texts are then utilized to form a response through a large language model according to the semantic intentions defined in the user's phrase.

2.2. Searching for an Answer to the Question that Corresponds to the Defined Semantics

A knowledge base can increase the level of accuracy and precision in retrieving answers.

Consider the use of artificial intelligence techniques for requirement analysis, specifically for analyzing non-functional requirements in security system development [5]. A requirements engineer can prepare additional prompts for the AI using the analytical fundamentals of security key distribution [6], open and closed key generation for cryptographic algorithms [7], or ensuring the safe use of credit cards [8] or networks to develop software systems that meet high-level security requirements [9].

Conclusion from the review. The current level of AI development is sufficient to begin experiments on applying these technologies to software development within the Agile methodology.

3. The task, the Research Questions, and a Scientific Novelty of the Proposed Approach

Task: To propose an approach to estimate the possibility of reusing software artifacts during the requirement analysis consulting with artificial intelligence technologies

3.1. Research questions (RQs):

1. Propose a concept for semantic comparison of different types of software development artifacts, namely software artifacts and user stories.
2. Ground the selection criteria for the environment used to visualize software models (UML diagrams).
3. Provide the support for different natural language groups, including Cyrillic, Romance, and Germanic languages.
4. Enable the analysis of source code written in various object-oriented programming languages.

5. Choose effective AI tools for source code analysis and semantic comparison of user stories written in natural languages.

3.2. Scientific novelty of the proposed approach

The approach addresses the challenge of analyzing the semantics of source code written in different object-oriented programming languages. The analysis results can be represented in various natural languages depending on user preferences. To achieve this, the approach combines AI technologies with established software engineering principles.

Technologies that provide scientific novelty of the proposed approach:

- From Model-Driven Engineering fundamentals Text-to-Text and Text-to-Model Transformations are used. Leverages the concept of text-to-model transformation from software engineering to represent the software artifact semantics in a structured format.
- From AI technologies Large Language Model (LLM) Prompt Engineering is used: Utilizes AI by employing LLM prompt engineering techniques to guide the extraction of semantic attributes while considering the specific characteristics of the chosen LLM.

4. Model-Driven Engineering Foundations of the Proposed Approach

The first step to involve Model-Driven Engineering fundamentals to approaches of software development lifecycle management is to provide the analysis of modelling environments "for text to model transformation"

Aim of this analysis is to select modelling environment with the simplest representation of textual description of UML diagram. Simple representation requires minimum efforts to teach AI tools to prepare a correct and full text representation of UML diagram. Figure 2 represents a classic model to model transformation scheme [10] with propositions (blue text labels) of elements' names that participate in the proposed approach.

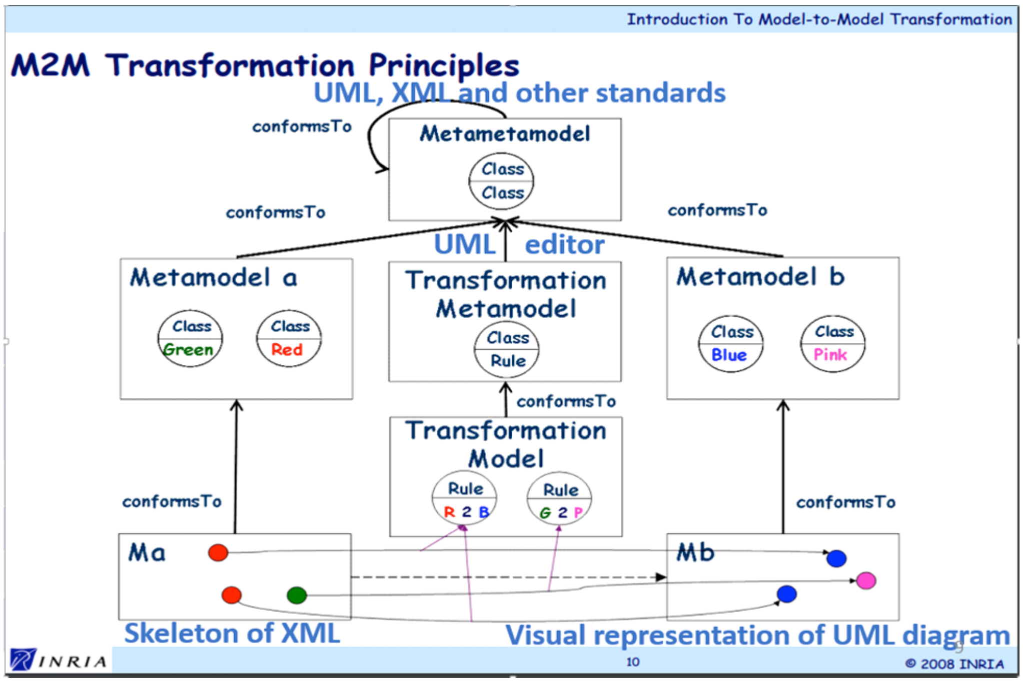


Figure 2: Classical "Model to Model transformation" scheme with description of key elements necessary for transformation. Figure is taken from [10].

Research area is concentrated on the generating "Model a" (Ma on the figure 2). "Model a" is a concrete type of software model (UML diagram) for concrete project. The description of this model must be generated by artificial intelligence tools. The research task is to prove that AI may generate correct model interpreting metamodel a correctly. “Metamodel a” in this scheme is a description of possible elements of UML diagram. For example, to describe class diagram Metamodel a support description of class diagram elements (classes and interfaces) and all possible relationships between them (inheritance, aggregation, composition, and association).

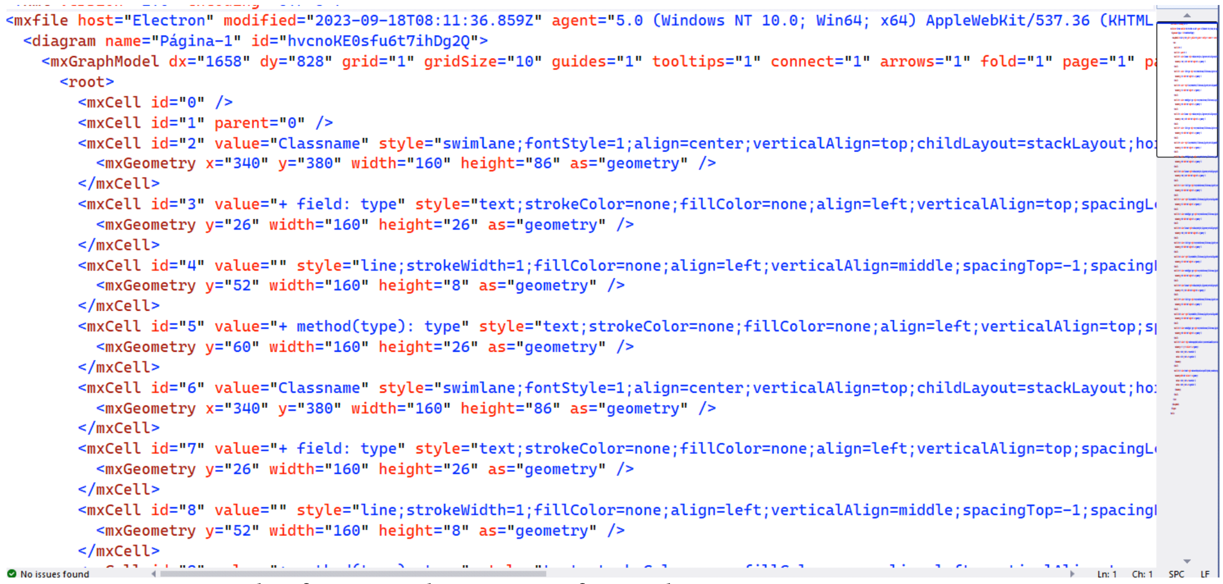
The Text-to-Model transformation (elements “Transformation Metamodel”, ”Transformation Model”, and “MetaModelb”) is done by modelling environment. The next modelling environments were considered:

- Visual studio plug-in for class diagram generating;
- DrawIO;
- Luquidchart;
- PlantUML;
- ASTAH UML.

Consider the results of the analysis of the effectiveness of modeling environments for generating UML diagram text descriptions.

DrawIO's requirement is to set coordinates for UML diagram elements complicates the creation of complex formal requests for AI tools.

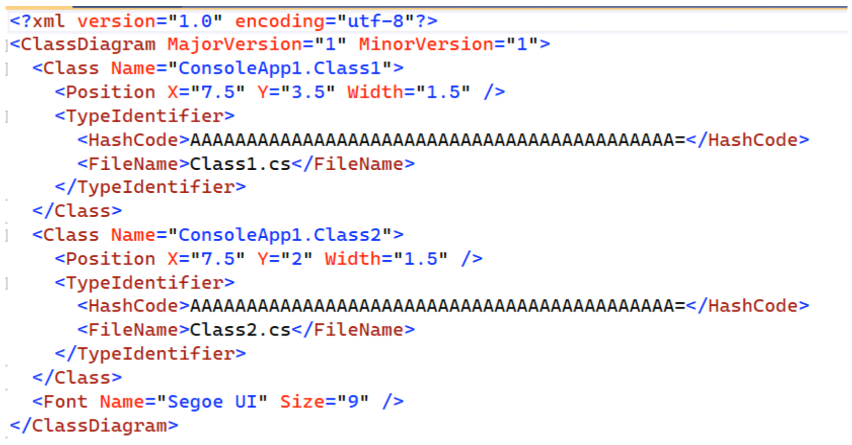
Figure 3 illustrates a portion of the description for a simple class diagram. Note the pointer on the right side of the figure.



```
<mxfile host="Electron" modified="2023-09-18T08:11:36.859Z" agent="5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
  <diagram name="Página-1" id="hvcnoKE0sfu6t7ihDg2Q">
    <mxGraphModel dx="1658" dy="828" grid="1" gridSize="10" guides="1" tooltips="1" connect="1" arrows="1" fold="1" page="1" pa
      <root>
        <mxCell id="0" />
        <mxCell id="1" parent="0" />
        <mxCell id="2" value="Classname" style="swimlane;fontStyle=1;align=center;verticalAlign=top;childLayout=stackLayout;ho
          <mxGeometry x="340" y="380" width="160" height="86" as="geometry" />
        </mxCell>
        <mxCell id="3" value="+ field: type" style="text;strokeColor=none;fillColor=none;align=left;verticalAlign=top;spacingL
          <mxGeometry y="26" width="160" height="26" as="geometry" />
        </mxCell>
        <mxCell id="4" value="" style="line;strokeWidth=1;fillColor=none;align=left;verticalAlign=middle;spacingTop=-1;spacing
          <mxGeometry y="52" width="160" height="8" as="geometry" />
        </mxCell>
        <mxCell id="5" value="+ method(type): type" style="text;strokeColor=none;fillColor=none;align=left;verticalAlign=top;s
          <mxGeometry y="60" width="160" height="26" as="geometry" />
        </mxCell>
        <mxCell id="6" value="Classname" style="swimlane;fontStyle=1;align=center;verticalAlign=top;childLayout=stackLayout;ho
          <mxGeometry x="340" y="380" width="160" height="86" as="geometry" />
        </mxCell>
        <mxCell id="7" value="+ field: type" style="text;strokeColor=none;fillColor=none;align=left;verticalAlign=top;spacingL
          <mxGeometry y="26" width="160" height="26" as="geometry" />
        </mxCell>
        <mxCell id="8" value="" style="line;strokeWidth=1;fillColor=none;align=left;verticalAlign=middle;spacingTop=-1;spacing
          <mxGeometry y="52" width="160" height="8" as="geometry" />
        </mxCell>
```

Figure 3: Example of DrawIO description of UML diagram

The next visualization environment considered is Visual Studio. The analysis reveals that the generated descriptions do not encompass all relationships between elements. Figure 4 exemplifies a description of two classes that are linked by inheritance. This description shows that inheritance relationship is missed. It is represented only in file with source code. Additionally, the complexity of the XML file structure hinders the creation of effective requests for AI technologies.



```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Class Name="ConsoleApp1.Class1">
    <Position X="7.5" Y="3.5" Width="1.5" />
    <TypeIdentifier>
      <HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
      <FileName>Class1.cs</FileName>
    </TypeIdentifier>
  </Class>
  <Class Name="ConsoleApp1.Class2">
    <Position X="7.5" Y="2" Width="1.5" />
    <TypeIdentifier>
      <HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
      <FileName>Class2.cs</FileName>
    </TypeIdentifier>
  </Class>
  <Font Name="Segoe UI" Size="9" />
</ClassDiagram>
```

Figure 4: Example of Visual Studio description of UML diagram.

Previous versions of Visual Studio utilized supplementary files, such as those with the .layout extension, to store information about element placement, color, and other visualization attributes.

A similar analysis was conducted for Gleek and PlantUML. Ultimately, PlantUML was selected due to its simpler, more accurate text notation for representing UML diagrams. This environment facilitates the formalization of instructions for AI tools in natural language across various types of natural languages [11].

5. Description of Technologies Stacks Allowing to Integrate ChatGTP to Software System for Requirement Analysis

To begin, an API key from OpenAI must be obtained through your account on the OpenAI website (in the API keys section). Then, a new project should be created using a code editor or an Integrated Development Environment (IDE) [12].

For ASP.NET Core applications, the OpenAI API client package can be utilized to streamline communication with the ChatGPT APIs (use the command Install-Package OpenAI from NuGet packages [13]). This package offers a collection of classes and methods that simplify interaction with the APIs.

The OpenAIClient class can be wrapped in a service class that implements an interface. It needs to be configured with your API key, and methods to transmit text prompts to the ChatGPT API and receive human-like responses. The Completions method of the OpenAIClient class can be used to send a completion request to the ChatGPT API. This method takes a CompletionRequest object as a parameter, which should contain the text prompt to be sent to the API. The CompletionRequest object can be customized with various parameters such as the maximum length of the generated text, the number of responses to generate, and the value to control the randomness of the generated text. The Completions method returns a CompletionResponse object, which contains the generated text as well as other metadata such as the usage statistics and the model used for generation. The generated text can be extracted from the CompletionResponse object and returned to the user as a natural language response [14,15].

This service class can be injected into the controller using dependency injection. The controller should expose endpoints that take user queries or context as input and return natural language responses generated by the ChatGPT API.

For Node.js applications, an HTTP client library like Axios can be used to communicate with the ChatGPT APIs. The API key should be incorporated in the request headers, and the request body should hold the text prompt to be sent to the API. After receiving the API response, it can be parsed and processed to extract the generated text. The parsed response can then be returned to the user as a natural language response [13].

Then the endpoints need to be tested to confirm that they are functioning correctly. This can be done using a tool such as Postman. User queries or context can be passed to the API, and natural language responses can be returned for testing purposes [14].

Finally, the application can be deployed to a web server or a cloud platform such as Azure or AWS. When deploying the application, it is important to ensure that the API key is securely stored and not exposed in the application code or configuration files. The API key should be securely stored in a configuration file or an environment variable and not hardcoded in the application code. When deploying the application, it is important to ensure that the API key is not exposed in the deployment configuration or the application logs. The application can be further enhanced by implementing features such as user authentication, input validation, and error handling [15].

6. Description of the Proposed Approach

The proposed approach modifies the classical domain engineering methodology by incorporating the following steps:

6.1. Domain Analysis Stage

During domain analysis, the semantic content of source code is extracted and represented as a set of epics with corresponding user stories. AI techniques are employed to derive these user stories from the codebase. To facilitate reuse, these extracted user stories are compared semantically with those generated during requirement analysis. Subsequently, an AI tool produces behavioral UML diagrams in PlantUML format. Traceability links are established between the user stories and these UML diagrams.

This stage leverages Model-Driven Engineering (MDE) principles, specifically Text-to-Text transformations (source code to user stories) and Text-to-Model transformations (PlantUML descriptions of UML diagrams).

6.2. Application Engineering Stage

In the application engineering phase, user stories are generated from requirement specifications or product vision documents using AI tools. A semantic comparison is then conducted between the project user stories and those derived from the source code. The resulting analysis identifies potential reusable software components, along with their associated UML diagrams, which are presented to developers as candidates for reuse.

Figure 5 presents a UML sequence diagram illustrating the proposed approach. The "Transformation Fundamentals" actor highlights the MDD aspects utilized at each step of the process.

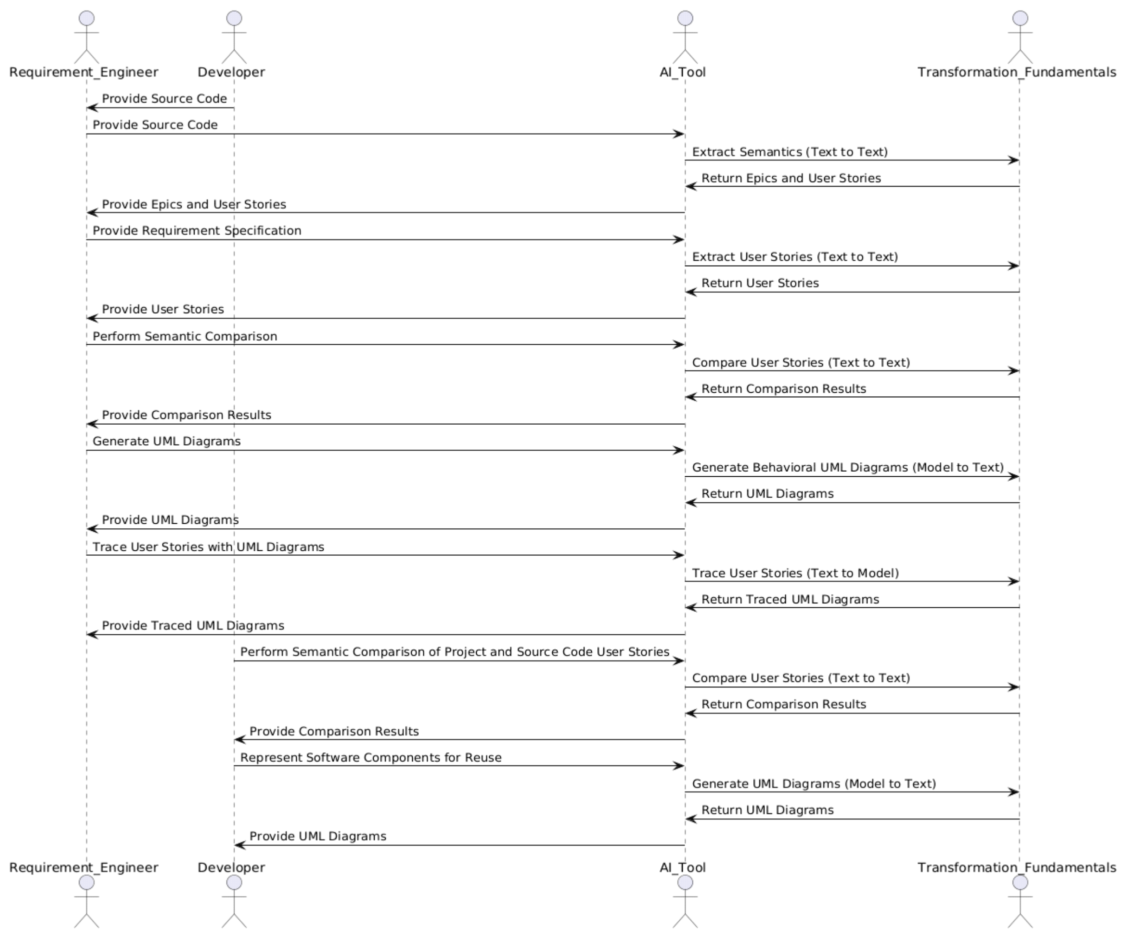

Requirement_Engineer
Developer
AI_Tool
Transformation_Fundamentals
Provide Source Code
Provide Source Code
Extract Semantics (Text to Text)
Return Epics and User Stories
Provide Epics and User Stories
Provide Requirement Specification
Extract User Stories (Text to Text)
Return User Stories
Provide User Stories
Perform Semantic Comparison
Compare User Stories (Text to Text)
Return Comparison Results
Provide Comparison Results
Generate UML Diagrams
Generate Behavioral UML Diagrams (Model to Text)
Return UML Diagrams
Provide UML Diagrams
Trace User Stories with UML Diagrams
Trace User Stories (Text to Model)
Return Traced UML Diagrams
Provide Traced UML Diagrams
Perform Semantic Comparison of Project and Source Code User Stories
Compare User Stories (Text to Text)
Return Comparison Results
Provide Comparison Results
Represent Software Components for Reuse
Generate UML Diagrams (Model to Text)
Return UML Diagrams
Provide UML Diagrams

Figure 5: UML sequence diagram, illustrating the proposed approach.

7. Experiment

The experiment aim is to demonstrate the feasibility of addressing research questions using simple source code modules. ChatGPT Copilot is selected as AI technologies for this purpose. To investigate language influences, Bulgarian (Cyrillic), Catalan (Romance), and English (Germanic) are chosen as representative languages.

7.1. Domain Analysis Phase

The experiment employs two Python scripts performing speech-to-text conversion as the source code. Table 1 outlines the prompts used for domain analysis in the three selected languages.

Table 1
Domain analysis prompts

English	Catalan	Bulgarian
Hello, can you provide a brief description of what this code does? Please generate a list of corresponding user stories.	Hola, pots donar una breu descripció del que fa aquest codi? Si us plau, genera una llista d'històries d'usuari corresponents.	Здравейте, можете ли да дадете кратко описание на този код? Изградете историите на потребителя.

Write a PlantUML description of a sequence diagram explaining what actions can be performed by this module. Write a PlantUML description of a component diagram as well. Thank you!!!!!	Construeix una descripció del diagrama de seqüència en PlantUML de quines accions es poden realitzar amb aquests mòduls. I construeix una descripció del diagrama de components en PlantUML de quines accions es poden realitzar amb aquests mòduls. Moltes gràcies!!!	Изградете описание на диаграмата на последователността в PlantUML за действията, които могат да се извършат с тези модули. Изградете описание на диаграмата на компонентите в PlantUML за действията, които могат да се извършат с тези модули. Много Ви благодаря!!!

Then, the Python source code for the class is given:

```
class Microphone:
    def __init__(self, file_name, voice_threshold=15, stop_seconds=2):
        # ... code here ... [16]
```

Result User stories and PlantUML diagrams have been generated. Notably, the generated UML diagrams are identical regardless of the prompt language.
Furthermore, the number and content of user stories remain consistent across different prompts. An example of a generated user story is provided below.

As a developer, I want to initialize the microphone module so that I can capture voice commands.	Com a desenvolupador, vull inicialitzar el mòdul del micròfon per capturar comandes de veu.	Като разработчик за софтуер аз искам да инициализирам модул за микрофон с цел да се запишат гласовите команди.

7.2. Application Engineering Phase

The product vision document from a customer is obtained (see figure 6).

Figure 6: The project vision document, obtained from a customer

Explanation of product vision document: Please write me "Speech to Text" module on python

Table 2
Prompts of requirement engineer

English	Catalan	Bulgarian
Hello! I need to write a 'speech to text' application in Python. Please, generate me user stories and epics for this	Hola! Necessito escriure una aplicació de 'conversa a text' en Python. Genera històries d'usuari i èpiques per a l'aplicació	Здравей, искам да напиша приложението което конвертира «глас в текст» на Python. Моля, генерирай ми истории за потребтеля и епици за

application	‘conversa a text’, si us plau.	това приложение

Result user stories are generated. Their content do not depend upon prompt language. The number and sense of user stories for all prompts are the same.
Example of one user story is provided below

As a user, I want to be able to convert my spoken words into text so that I can save time on typing.	Com a usuari, vull poder convertir les meves paraules parlades en text per estalviar temps en escriure.	Като потребител искам да мога да преобразувам изговорените си думи в текст, за да спестя време за писане.

7.3. Semantic comparison of requirement specification and source code

Table 3
Semantic comparison prompts

English	Catalan	Bulgarian
Hello i have two arrays of user stories.	Hola, tinc dues llistes d’històries d’usuari.	Здравейте, имам два набора от потребителски истории.
Please, compare them semantically and give me an advice which user stories from the second list may be used to create software for the first list	Si us plau, compara-les semànticament i prepara una llista d’històries d’usuari de la segona llista que signifiquin el mateix que les històries d’usuari de la primera llista.	Моля, сравнете ги семантично и ми дайте съвет кои потребителски истории от втория списък могат да бъдат използвани за създаване на софтуер за първия списък.
array 1 (future project) As a user, I want to be able to convert my spoken words …….. … my speech, so that may be used later.	llista 1 (futur projecte) Com a usuari, vull poder convertir les meves paraules parlades ……… … el meu discurs perquè més tard.	Набор 1 (бъдещ проект): Като потребител искам да мога да конвертирам моите произнесени думи...... от речта ми, така че да може да се използва по-късно.
array 2 (user stories of existing components) As a developer, I want …… I want to hear playback of audio responses.	llista 2 (històries d’usuari de components existents) Com a desenvolupador, vull …… vull escoltar la reproducció de respostes d’àudio.	Набор 2 (съществуващи компоненти): Като разработчик искам...... искам да чуя възпроизвеждане на аудио отговори.
Thank you!	Moltes gracias!!!!	Благодаря!

The user stories that have the same semantics are defined. For this project the first user story corresponds to the list of other stories that describe realization of voice commands of software components

As a user, I want to be able to convert my spoken words into text so that I can save time on typing.	Com a usuari, vull poder convertir les meves paraules parlades en text per estalviar temps en escriure.	Като потребител, аз искам да превърна казаните думи в текста.
As a developer, I want to	Com a desenvolupador, vull	Като разработчик искам да инициализирам модула за

initialize the microphone module so that I can capture voice commands.	inicialitzar el mòdul del micròfon per poder capturar comandes de veu.	микрофон, за да мога да улавям гласови команди.
As a system user, I want to open an audio stream to receive voice input.	Com a usuari del sistema, vull obrir un flux d'àudio per rebre entrada de veu.	Като системен потребител искам да отворя аудио поток, за да получавам гласов вход.
As a voice command system user, I want to read audio data from the microphone module.	Com a usuari del sistema de comandes de veu, vull llegir dades d'àudio del mòdul del micròfon.	Като потребител на система за гласови команди искам да чета аудио данни от модула за микрофон.
As a user interacting with the voice command system, I want to start and stop recording voice commands.	Com a usuari que interactua amb el sistema de comandes de veu, vull començar i aturar la gravació de comandes de veu.	Като потребител, който взаимодейства със системата за гласови команди, искам да започвам и спирам записването на гласови команди.
As a system administrator, I want to save recorded audio files for analysis or archival purposes.	Com a administrador del sistema, vull guardar els fitxers d'àudio gravats per a l'anàlisi o fins arxivístics.	Като системен администратор искам да запазвам записаните аудио файлове за анализ или архивиране.

8. Conclusion

The paper proposes the approach to estimate the feasibility of reusing software development artifacts during requirement analysis. This approach is grounded in the semantic analysis of software modules and a subsequent comparison between the semantics of these modules and user requirements. User stories are employed as semantic attributes for both software development artifacts and requirement analysis artifacts.

Artificial intelligence plays a crucial role in automating routine tasks such as recognizing software component semantics and comparing user stories derived from both source code and requirement analysis.

The proposed approach offers several advantages through the utilization of artificial intelligence:

- It enables the search for software components written in different programming languages.
- It eliminates limitations imposed by the natural language used by developers and requirement engineers.
- It facilitates efficient processing of extensive requirement specifications.
- It reduces human involvement in discovering the semantics of software development artifacts.
- It increases the likelihood of reducing development time and costs.

9. Acknowledgements

This paper is performed as a part of a research project "Ingeniería de dominio para los desarrollos de inteligencia artificial" (Domain engineering for artificial intelligence development) de Instituto de Investigación en Inteligencia Artificial (IIIA, Catalonia, Spain), Consejo Superior de Investigaciones Científicas (CSIC, Spain).

I would like to express my sincere gratitude to Professor Carles Sierra, Research Professor and Director of the Artificial Intelligence Research Institute (IIIA) at CSIC, for his exceptional leadership and unwavering support throughout this project, guidance in both technical and organizational aspects, as well as his encouragement in my Catalan language studies.

Additionally, I would like to extend my heartfelt thanks to Joan Jené, Head of the Technology Transfer & Development Unit (UDT) at the IIIA, for generously providing the source code modules essential for experiments [16], as well as his encouragement in my Catalan language studies.

References

[1] “ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary” https://www.iso.org/standard/71952.html

[2] A., Linvin, O., Palagin, V., Kaverinsky, K., Malakhov. “Ontology-driven development of dialogue systems.” South African Computer Journal, vol. 35, no. 1, 2023, pp. 37–62. https://doi.org/10.18489/sacj.v35i1.1233.

[3] Palagin, O., Kaverinsky, V., Petrenko, M., Malakhov, K. “Digital Health Systems: Ontology-Based Universal Dialog Service for Hybrid E-Rehabilitation Activities Support.” Proceedings of the 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Dortmund, Germany, vol. 1, 2023, pp. 84–89. https://doi.org/10.1109/IDAACS58523.2023.10348639.

[4] Boiarskyi, O., Popereshnyak, S. “Automated System and Domain-Specific Language for Medical Data Collection and Processing.” In: Babichev, S., Lytvynenko, V. (eds) Lecture Notes in Computational Intelligence and Decision Making. ISDMCI 2021. Lecture Notes on Data Engineering and Communications Technologies, vol. 77, Springer, Cham, 2022, pp. 25. https://doi.org/10.1007/978-3-030-82014-5_25.

[5] Palagin, O., Kaverinskiy, V., Petrenko, M. G., Malakhov, K. "Fundamentals of the Integrated Use of Neural Network and Ontolinguistic Paradigms: A Comprehensive Approach." Cybernetics and Systems Analysis, vol. 60, no. 1, 2024, pp. 111–123. https://doi.org/10.1007/s10559-024-00652-z.

[6] Masol, V., Popereshnyak, S. "Joint Distribution of Some Statistics of Random Bit Sequences." Cybernetics and Systems Analysis, vol. 57, no. 1, 2021, pp. 139–145. https://doi.org/10.1007/s10559-021-00337-x.

[7] Tynymbayev, S., Gnatyuk, S., Ibraimov, M., Namazbayev, T., Mukasheva, A. "Cybersecurity Providing in Information and Telecommunication Systems." CEUR Workshop Proceedings, vol. 3654, Kyiv, Ukraine, 2024, pp. 513–519.

[8] Popereshnyak, S. "Technique of the Testing of Pseudorandom Sequences." International Journal of Computing, vol. 19, no. 3, 2020, pp. 387–398. https://doi.org/10.47839/ijc.19.3.1888.

[9] Baisholan, N., Turdalyuly, M., Gnatyuk, S., Baisholanova, K., Kubayev, K. "Implementation of Machine Learning Techniques to Detect Fraudulent Credit Card Transactions on a Designed Dataset." Journal of Theoretical and Applied Information Technology, vol. 101, no. 13, 2023.

[10] Figure model to model transformation is taken from https://wiki.eclipse.org/images/9/90/OMCW_chapter10_Modelplex-WP6 Training_IntroductionToM2M.pdf

[11] “PlantUML Language Reference Guide (1.2020.22)” https://pdf.plantuml.net/1.2020.22/PlantUML_Language_Reference_Guide_en.pdf

[12] Ranasinghe, N. "Effortlessly Integrate OpenAI ChatGPT APIs in .NET Core 7 WebAPI with Ease." Medium, 2023. URL: https://medium.com/@nirajranasinghe/effortlessly-integrate-openai-chatgpt-apis-in-net-core-7-web-api-with-ease-7658ab26afc3.

[13] Friedner, J. "Make an API Request to Chat GPT-4 with Next.js Using JavaScript." Medium, 2023. URL: https://medium.com/@JohanFriedner/make-an-api-request-to-chat-gpt-4-with-next-js-using-javascript-c238b47bd88a.

[14] Tripathy, J. "ChatGPT Integration in ASP.NET Core Using OpenAI." Jayant Tripathy, 2023. URL: https://jayanttripathy.com/chatgpt-integration-in-asp-net-core-using-openai/.

[15] "ChatGPT Completions in ASP.NET Core Web API." C# Corner, 2023. URL: https://www.c-sharpcorner.com/article/chatgpt-completions-in-asp-net-core-web-api/.
[16] Jené, J “Source code, taken for experiment” https://drive.google.com/file/d/1WxoRGndDxMEMMgdK3tScqA7CcrQVhG7o/view?usp=drive_link