

A Semantic NLP Framework for Automated Transformation of Software Requirements into Interactive Prototyping

Ali Hamzah Obaid¹ , Khansaa Azeez Obayes Al-Husseini^{2*} 

¹Al-Furat Al-Awsat Technical University College of Medical and Health Technologies, Babylon, Iraq

Email: inb.ali210@atu.edu.iq

²Al-Furat Al-Awsat Technical University Babylon Technical Institute Babylon, Iraq,

inb.khanssa@atu.edu.iq

Article's Information

Received: 24.04.2026

Accepted: 03.06.2026

Published: 7.10.2026

Keywords:

Prototype

NLP

Intelligent

Workflow

Semantic Analysis

Abstract

The natural language requirements are a common way for software development to represent requirements, but they have the disadvantage of being ambiguous and inconsistent, making requirements engineering and initial system design more challenging. This paper proposes the framework for automatic translation of textual software requirements to software structured artifacts and interactive prototypes using semantic Natural Language Processing (NLP). The proposed framework is a multi-stage NLP pipeline consisting of text normalization, tokenization, part-of-speech tagging, named entity recognition, dependency parsing and semantic relation extraction. Semantic elements extracted are represented in a structured way to capture actors, actions, entities, constraints and context information.

A novel mechanism is proposed that facilitates automatic generation of user stories, use case models, interaction flows and prototype interface components from textual requirements which is referred to as semantic mapping. To test the framework, a set of software requirements was gathered across several application areas like e-commerce, healthcare, education, smart city and Internet-of-things applications. The metrics for evaluating the performance were requirement extraction accuracy, prototype generation efficiency and modelling effort reduction.

The experimental results show that the proposed framework can successfully satisfy the requirement extraction accuracy of 93%, greatly decrease manual modelling work, and faster generate the prototype compared with the traditional requirements engineering methods. Results show that the use of semantic NLP techniques and automated prototypes can help to better interpret requirements, validate them at an early stage, and facilitate communication among stakeholders and development teams. The proposed framework offers a viable method to connect the natural language requirements with preliminary design artifacts for software.

<https://doi.org/10.46649/fjiece.v5.2.1a.7.10.2026>

*Corresponding author: inb.khanssa@atu.edu.iq

1. INTRODUCTION

Software systems are written more and more according to requirements written in natural language by the stakeholders, analysts and domain expertise [1]. Validation of system requirements, which are usually written in natural language, is a prerequisite to guarantee the validity of a system specification. Although natural language offers flexibility and accessibility to explain the functionality of the system, it also creates ambiguity, inconsistency, and variability in the way it is understood [2]. These features make it a difficult task at the initial phases of the system development to transform textual requirements into structured software models[3]. Requirement engineering is one of the corner stones of the software lifecycle because proper interpretation of the requirements of the stakeholders determines the quality of the system and the efficiency of the development process[4]. Practically, the requirement specifications are usually represented as natural language documents, which require widespread analysis by software engineers[5]. This is a manual process that is time consuming and can cause misunderstandings, missing requirements interpretation and design inconsistencies in a system[6]. The new possibilities in automated requirement analysis are recently opened by the new methods of natural language processing[7],[8]. NLP techniques can be used to derive semantic data in a text description and recognize the pertinent entities, activities, and relationships in requirement statements[9]. Meanwhile, software prototyping has emerged as a critical tool in contemporary development practice especially in agile and user-centred developmental settings[10]. Prototypes allow development teams to visualize how a system works, can test design assumptions, and can also get early feedback on the system[11]. However, in spite of these advances, most of the current methodologies view requirement interpretation and prototype development as independent processes. The NLP-based requirements analysis methods are primarily devoted to the extraction of structured information about the textual requirements, but, prototyping tools are usually based on the manually built system models[12]. The result of this separation is a gap between the requirement interpretation and early system visualization. In order to overcome this shortcoming, this paper presents a smart workflow which combines natural language processing solutions with a prototyping-based software engineering style. The suggested workflow automatically converts natural language requirement descriptions into formal development artifacts and then creates interactive system prototypes that can enable initial verification and the refinement of prototypes. This study has a number of key contributions that can be stated as follows:

- A smart NLP-based workflow architecture of automated requirement interpretation.
- A conceptual model is a description of the process of converting textual requirements into structured software artifacts.
- A prototype that would prove the practicability of the suggested approach.
- An experimental study that determines the efficiency of the workflow in terms of enhanced requirement interpretation accuracy and prototype generation efficiency.

The rest of this paper will be structured in the following way: Section 2 will review related literature on NLP-based requirement engineering and the software prototyping. Section 3 outlines the design of the proposed workflow. Section 4 presents the conceptual framework model. Section 5 presents the research design and experimental condition. The results of the experiment are discussed in Section 6 and the discussion and implications are presented in Section 7. Lastly, Section 10 gives a conclusion of the paper and gives future research directions.

2. RELATED WORK

Current studies have been paying more and more attention to the use of natural language processing methods to facilitate the processes in the context of requirement engineering. The main idea is that these methods reduce the uncertainty in the textual requirements and help the developers to convert the unstructured

requirement descriptions into the structured software artifacts[13].A number of researchers have been working on the use of NLP methods to retrieve semantic information in the requirement documents. Such methods are often based on the use of tokenization, part-of-speech tagging, named entity recognition, and dependency parsing to find system actors, actions, and domain entities in textual requirement specifications. With the extraction of these elements, NLP-based systems could be used to assist in automated classification of requirements, traceability analysis and generation of structured requirements models[14],[15].

Besides the rule-based NLP methods, modern-day literature has proposed the application of machine learning and deep learning methods in enhancing the precision of requirement extraction. These techniques are designed to determine the elements of requirements and relationships based on statistical models that are trained on annotated datasets. Although these methods have proved to give good results in terms of requirement interpretation and information extraction, their major concern is limited to the requirement analysis as opposed to the support of the later phases of the system development[16],[17].In line with these innovations, software prototyping has emerged as a common software engineering practice in contemporary software engineering techniques, especially in agile and user-centric software development processes. Prototyping enables development teams to produce initial models of system behavior, test design options, and receive feedback on the system design with stakeholders prior to implementation of the entire system. Early system design has greatly been made more efficient with the aid of rapid prototyping tools which have helped it to create interactive system models[18],[19].In spite of such developments, requirement interpretation and prototype generation have been considered as different processes in most of the existing studies. The approaches based on NLP are mainly geared towards the extraction of structured textual requirements information as opposed to prototyping tools which usually use manually developed models and specifications. As a result, it is still often necessary to use a lot of manual work to transform the natural language requirements into the prototypes of a functional system[20],[21].Newer studies seek to fill this gap by creating intermediate artifacts like user stories, use case diagrams or conceptual models based on textual requirements. Despite being an essential move in automation, several of them lack a complete workflow that consists of interpretation of requirement, model generation, and prototype construction in a single development paradigm[22],[23].Thus, the necessity of a unified solution that would integrate NLP-based requirement analysis with automated prototype generation still exists and can be used in order to facilitate early system validation and enhance the efficiency of development. The workflow suggested in the current paper is capable of overcoming this issue, as it suggests a smart processing pipeline to convert the natural language description of requirements into structured artifacts and interactive system prototypes[24],[25],[26].

2.1 Research Gap

Despite the major developments achieved in the field of NLP-based requirement analysis and software prototyping processes, there is still little integration between the fields of research. The current research literature is mostly confined to the extraction of structured information in requirement documents or to the creation of prototypes according to the manually created models.

This results in significant amounts of manual intervention in the process of converting natural language requirements into functional system prototypes. This constraint limits the development efficiency as well as the probability of misinterpretation of the requirements in the initial design phases.In order to cope with this problem, this research paper suggests a smart workflow that incorporates NLP-based requirement interpretation and auto-generated prototype generation. The proposed approach will be applied to facilitate early visualization of the system and enhance the efficiency of the process of the requirement engineering by integrating semantic analysis and the prototyping techniques.

3. PROPOSED INTELLIGENT WORKFLOW ARCHITECTURE

This section outlines a proposed intelligent workflow design that will attempt to transform natural language specifications into properly defined software objects and prototypes used for interactive systems. The requirements extraction process consists of several essential steps: text preprocessing, feature extraction and machine learning models. It uses natural language processing techniques combined with the prototyping-based cycle of system development to automate and assist in the process of interpreting requirements and initial system stability verification.

The proposed construct is organized in a sequence of multi-stage processing pipeline, under which the textual requirements descriptions are converted into structured representations over time and additional usage towards the creation of working prototypes. Such an integration allows the development teams to move to the next stage of requirement description to early system visualisation, which improves the understanding of the requirements and reduces the amount of manual modelling effort. Figure 1 shows the overall structure of the proposed smart workflow. The framework involves four major processing steps, which are requirement acquisition, NLP-based requirement analysis, semantic model creation, and prototype generation. These phases work in a linear manner at the same time allowing continuous improvement with the help of stakeholder feedback.

3.1 Requirement Acquisition

The first step of the workflow is focused on the gathering of requirement descriptions, which are written in the natural language. These requirements could be a result of various sources, such as stakeholder interviews, requirements specification documents, report of user feedback, or system design discussions. The mined textual needs are archived in a formal repository thus facilitating efficient processing at the later stages of the workflow. At this point, linguistic analysis is ready in the form of requirement statements, through the elementary operations of preprocessing of text, including text normalisation and text excision, as well as language standardisation.

3.2 REQUIREMENT ANALYSIS BASED ON NLP

The second step of the workflow involves using the natural language processing methods to interpret textual needs and determine the relevant semantic pieces of information. The stage attempts to transform unstructured textual descriptions, to structured linguistic representations which may form the backbone of system modelling.

The steps involved in the NLP analysis take a number of steps:

- tokenisation and pre-lexical treatment.
- part-of- speech tagging to recognize grammatical roles.
- named entity recognition to identify domain entities and actors.
- dependency parsing to find relationships among requirement components.
- action, object and interaction semantic extraction.

The system recognizes critical components of requirement statements through these operations, which are; actors, system functions, domain entities, and interrelationships.

The construction of the semantic model follows this step.

After the linguistic feature extraction using requirement statements, the workflow provides structured software artefacts as a representation of semantic interpretation of system requirements. These artefacts are a step in between the chasm between textual descriptions of requirement and system design models.

The artefacts that are as a result include:

- user stories performing the definition of system functionality in the eyes of the stakeholders.
- use-case models of system interaction.

- conceptual entity models of system components and their interrelationships.
- interaction-flow diagrams of system behavior.

These designed diagrams diminish ambiguity in interpretation of requirements and facilitate stakeholder-software developer communication.

3.3 Prototype Generation

The last phase of the working process converts the developed artefacts of the requirements in the form of interactive prototypes of the system. Prototyping facilitates early system visualisation to the stakeholders and also enables them to test their interpretation of requirements before the actual system implementation begins.

The process of prototype generation has a number of steps:

- production of layout elements of an interface.
- building of interaction flows based on use-case scenarios.
- system modelling
- modelling of system behaviour
- design of interactive prototype interfaces.

The prototypes created provide the stakeholders with a chance to experiment with system behaviour and provide feedback with regards to functionality and usability of the system.

3.4 Feedback And Refinement, An Iterative Approach.

One of the most significant features of the suggested workflow is the presence of an iterative feedback mechanism. The stakeholders are given the ability to engage with the developed prototypes and make their contributions towards the interpretation of requirements, system functionality and interface design. This feedback will be incorporated in the workflow so as to refine model of requirements and regenerate more improved system prototypes. Iteration process helps in ensuring that the system requirements have been correctly interpreted and matched with the expectations of the stakeholders in the initial stages of development.

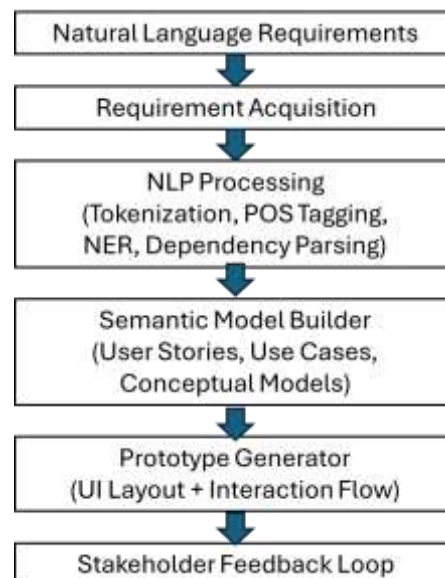


Fig.1 Proposed Intelligent NLP- Prototyping System Architecture

4. Conceptual Model of the Semantic NLP-Driven Prototyping Framework

In this section, the conceptual model of proposed semantic NLP-based framework is presented. The model represents the transformation of natural language requirements to structured software engineering artifacts

and interactive system prototypes in a systematic way. The proposed model differs from the traditional requirement engineering models in that requirement interpretation and the creation of prototypes are not separate activities, but are part of a single transformation pipeline. The conceptual model provides a strong link between requirements data, semantic interpretation mechanisms, component generation procedures, and prototyping processes. The goal of this link is to reduce interpretation ambiguity and provide an automated path from textual specifications to actionable prototyping representations.

There are four proposed layers, each connected to the other, in the conceptual model:

- Requirement Representation Layer .
- Semantic Interpretation Layer
- Semantic Mapping and Artifacts Generation Layer.
- Prototype Construction Layer.

The entire conceptual change process is presented in figure 2 according to the proposed framework.

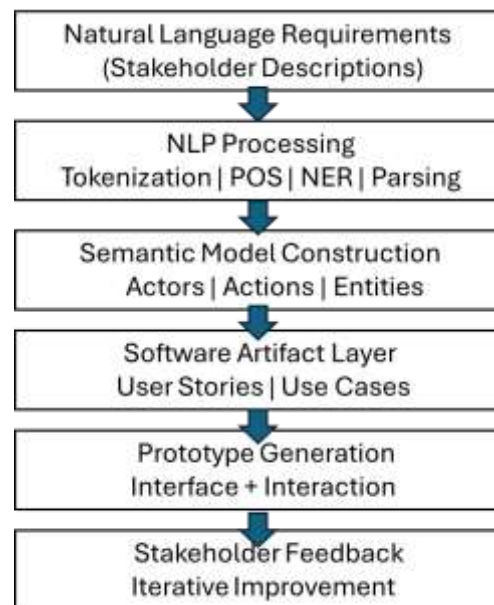


Fig.2 Layered conceptual model illustrating the transformation of natural language requirements into structured artifacts and system prototypes.

4.1 Requirement Representation Layer

The Requirement Representation Layer is the starting point of the framework. This is when software requirements are captured in natural language from requirement specification documents, stakeholder meetings, user stories, and business process descriptions.

Let's assume that the requirement repository is defined as:

$$R = \{r_1, r_2, r_3, \dots, r_n\}$$

Where:

R = Represents The Complete Requirement Set

r_i = Represents an Individual Requirement Statement.

All requirements statements contain one or more semantic elements that describe system functions, user interaction, or operational constraints. As an example: " The customer places orders via the mobile application. "

Although these statements are easily understood by humans (users, clients, etc.), they remain unsuitable for direct software modeling because they contain implicit semantic information that must first be extracted and formally formulated. Consequently, requirements statements are referred to the semantic interpretation layer for linguistic analysis and knowledge extraction.

4.1 Semantic Interpretation Layer

The Semantic Interpretation Layer converts the unstructured textual requirements into machine interpretable representations of them.

The framework uses NLP pipeline with multiple stages such as:(Text Normalization, Tokenization, Part-of-Speech Tagging, Named Entity Recognition (NER) , Dependency Parsing , Semantic Relation Extraction)

The function to represent the semantic extraction process is:

$$S = F(R)$$

Where:

S : Represents the Generated Semantic Model

F : Represents the NLP Interpretation Function

R : Represents the Requirements set,

The following function can be used to define the internal structure of a natural language processing (NLP) interpreter:

$$F(R) = E(D(N(P(T(R))))$$

Where:

R : represents the set of natural language requirements.

S : represents the structured semantic

T : *Tokenization.*,

P : *POS Tagging.*,

N : *Named Entity Recognition.*

D : *Dependency Parsing.* ,

E : Semantic Extraction

The semantic representation is defined as:

$$S = \{Actors, Action, Objects, Context, Attributes, State, Temporal Constraints, Relation\}$$

This improved framework representation allows for the accommodation of complex software requirements that include conditions, constraints, states, and contextual information.

For example, the following condition: "A registered premium customer has the right to cancel an order within 12 hours of its submission."

$$S = (Customer, Cancel, Order, Registered, Status = Submitted, 12h, Ownership)$$

The constructed semantic model forms the basis for generating automated software components.

4.3 SEMANTIC MAPPING AND ARTIFACT GENERATION LAYER

Bridging the gap between requirements interpretation and software design is one of the proposed framework's contributions to the semantic binding layer. After semantic extraction, the framework automatically transforms semantic representations into software engineering elements using a rule-based binding engine. The transformation process is displayed as follows:

$M \{(US) = \text{User Stories}, (UC) = \text{Use Cases}, (IF) = \text{Interaction Flows}, (CM) = \text{Conceptual Models}\}$

A sample table summarizing the rules of joining.

Semantic Element	Generated Artifact
Actor	Use Case Actor
Action	Use Case Function
Object	Domain Entity
Context	Operational Condition
Attribute	Interface Field
State	Workflow Status
Temporal Constraint	Validation Rule
Relation	Interaction Lin

To illustrate how the conversion process works, consider the following:

"The customer places the order via the mobile app."

The semantic extraction component generates the following:

Element: Value , Agent: Customer , Action: Place ,Object: Order ,Context: Mobile App

The framework then creates the user story:

"As a premium customer, I want to place an order via the mobile app to complete the purchase and payment process." . The corresponding use case is automatically generated as follows:

- Use case: Order Placement
- Principal actor: Customer
- Precondition: User authentication
- Postcondition: Order saved successfully

Interaction flows and conceptual entity models are subsequently derived from the generated use case structure. This semantic linking mechanism ensures traceability between textual requirements and generated design outputs, while minimizing manual modeling effort.

4.4 Prototype Construction Layer

Executable prototypes are obtained from the final layer's transformation of the software output. They can be represented as follows:

$$P = G(S)$$

where:

P = represents the generated prototype, $G(S)$ = represents the prototype generation function

We can formally describe how the prototype is generated as follows:

$$G(S) = UI(S) + UC(S) + IF(S)$$

Where:

S = semantic represntation, P = generated system prototype, UI = Interface Generator

UC = Use Case Genarator, IF = Inteaction Flow Generator

Automatically, the framework generates the following:

User interface components , Navigation structures ,Form elements ,Verification mechanisms
Interactive workflows.

The integration of these functions (semantic interpretation and prototype generation) transforms the entire framework.:

$$P = G(F(R))$$

These elements are assembled into an interactive prototype capable of simulating system functionality before implementation.

4.5 Conceptual Model Summary .

The conceptual model proposes the automatic integration of the above four phases into a single flow. The framework overcomes some of the major drawbacks of current NLP-based requirement engineering methods by going beyond an extraction of the actor-action tuple and implementing an explicit semantic mapping mechanism. The value of this model lies in its provision of a structured and repeatable framework that aims to automate the translation of user requirements (natural language) into a trial and active software version, while ensuring complete semantic consistency throughout the system lifecycle.

5. Research Methodology

The method used in this study is Design Science Research (DSR) which is used for developing and evaluating a framework based on the semantic NLP in automated software requirement analysis and prototype generation. The research approach adopted is a framework design, prototype construction and experimental testing technique to evaluate the feasibility of converting natural language requirements to structured software artefacts and interactive prototypes.

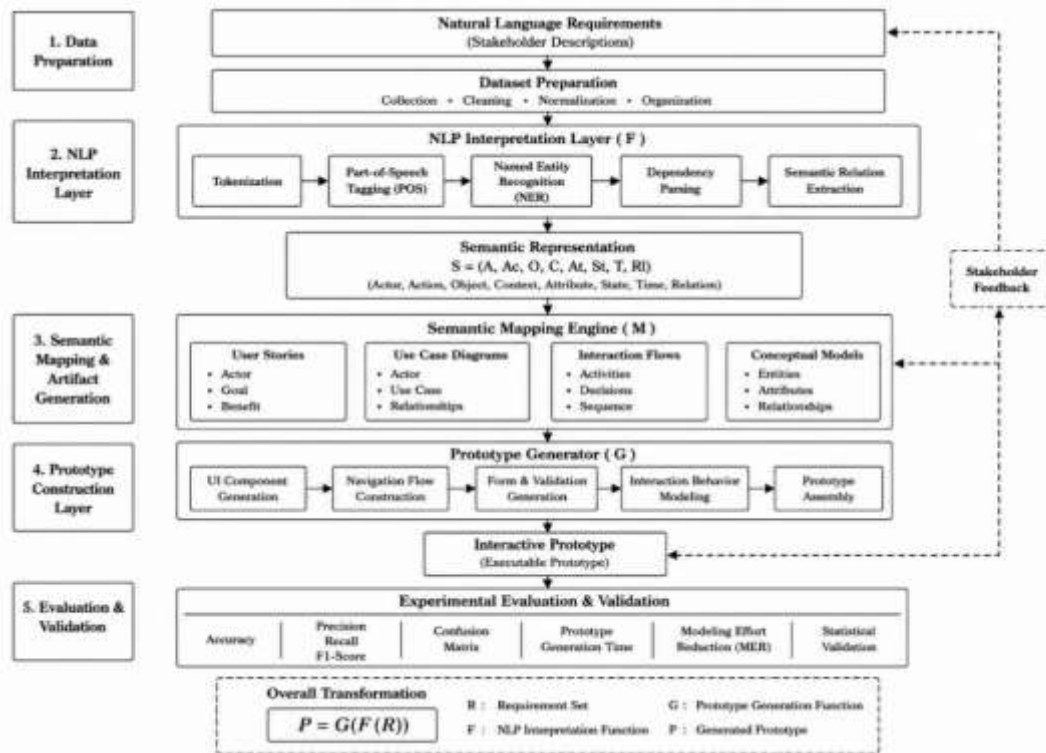


Figure 3 . Conceptual Transformation Model for Automated Requirement-to-Prototype Generation Using Semantic NLP.

5.1 Requirement Dataset Preparation

In order to gain diversity and domain independence, 120 software requirements statements were gathered from several domains of application. The data set was carefully packaged as a proof-of-concept benchmark to assess the suitability of the proposed framework to work in diverse software environments.

Table 1. Dataset Distribution

Domain	Requirements
E-Commerce	28
Healthcare Information Systems	25
Educational Systems	20
Smart City Applications	22
IoT Applications	25
Total	120

All requirement statements were preprocessed prior to processing, and consisted of the following:

- Text normalization
- Removal of duplicated statements
- Grammar correction
- Terminology standardization
- Noise reduction

The above preprocessing operations helped to achieve linguistic consistency and minimise ambiguity in the semantic extraction process

5.2 NLP – Based Requirement Interpretation.

The second stage utilizes the natural language processing methods to process the requirement statements and identifies the relevant semantics. This stage aims at converting textual descriptions that are in unstructured form into structured representations that may be used to model the system.

Here are the main steps in the NLP processing pipeline:

- Tokenization - the splitting up of requirement sentences into single tokens.
- Part -of-Speech Tagging - assignment of grammatical roles of each token.
- Named Entity Recognition- identification of system actors and domain entities.
- Dependency Parsing - syntactic relationships between tokens.
- Semantic Extraction- extracting verb, noun, and interaction of requirement statements.

With the use of these operations, the main semantic aspects of requirements are identified such as actors, system functions, entities, and mutually dependent relationships between them. The information that was extracted is then translated into a high level semantic representation which is used to produce software modeling artifacts. Technology-based knowledge acquisition entails generating structured artifacts to support decision making in social, economic, and political spheres.

The extracted information is organized using the semantic representation model presented in Section 4. The natural language processing function is defined as:

$$S = F(R)$$

Where : (R) = the set of requirements and (S) = represents the created semantic representation.

5.3 Artifact and Prototype Generation

After semantic interpretation, the framework is based on a semantic mapping engine that converts extracted knowledge to software engineering-related artifacts in the form of a structured software engineering knowledge graph and a prototype executable representation. The semantic linking mechanism is based on the transformation model described in Section 4 and maintains the traceability of the requirements data and the artifacts that have been created. The resulting software products are: User Stories Use Case Models Conceptual Entity Models Interaction Flow Diagrams These artifacts are mediators between the requirements written in text and the design elements of the software. The prototype generation module then automatically converts the artifacts it produces into interactive system prototypes. The prototype construction process has the following steps: Interface Component Generation Navigation Flow Construction Form and Validation Generation Interaction Behavior Modeling Prototype Assembly These prototypes provide an opportunity to get feedback early on from stakeholders, and to refine them during the product development process before software is implemented. The formal representation of the prototype generation process is:

$$P = G(S)$$

(P) = indicates the prototype generated. The (G)= stands for prototype generation function. The NLP interpretation layer generates a semantic representation (S).

The integrated transformation process enhances and improves the bridging of the gap between natural language specifications and executable prototype representations in a single development process.

5.4 Implementation Environment

At this stage, the initial system is implemented using a software environment to ensure the reproducibility of the results. The table 2 summarizes the implementation techniques used in this framework.

Table 2 summarizes the implementation techniques

Component	Technology
Programming Language	Python 3.11
NLP Engine	spaCy
POS Tagging	spaCy POS
Named Entity Recognition	spaCy Parser
Semantic Mapping Engine	Custom Rule-Based Module
Prototype Generator	HTML/CSSTemplates
Data Storage	SQLite

The selected execution environment allows automated requirements interpretation, semantic mapping, artifact creation, and prototyping within a unified processing path.

5.5 Experimentally Design and Evaluation Metrics

The proposed framework was empirically evaluated against two main approaches: manual requirements modeling and natural language processing-assisted modeling without prototyping. Both approaches were evaluated using the same requirements dataset to ensure consistency and fairness of the comparison. The evaluation focused on measuring the quality of semantic interpretation and the efficiency of practical development using metrics such as requirements extraction accuracy, precision, recall, F1 metric, prototyping time, and finally, minimizing modeling effort (MER). Requirements extraction performance was also evaluated using standard information retrieval metrics, according to the precision equation:

$$Accuracy = \frac{TP + TN}{(TP + TN + FP + FN)}$$

$$Precision = TP / (TP + FP)$$

$$Recall = TP / (TP + FN)$$

$$F1 = (2 \times Precision \times Recall) / (Precision + Recall)$$

where:

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

The reduction in modeling effort (MER), used to evaluate development efficiency, was calculated in the following equation.

$$MER = ((T_{manual} - T_{proposed}) / T_{manual}) \times 100$$

where:

- $(T_{\{manual\}})$ denotes the effort required using traditional modeling techniques.
- $(T_{\{proposed\}})$ denotes the effort required using the proposed framework.

This metric measures the practical reduction in manual modeling effort achieved through automation.

6. Experimental Results and Evaluation

This section will present the experimental evaluation of the proposed semantic NLP-based framework. The goal of the evaluation is to verify the effectiveness of the framework for extracting semantic information from the natural language requirements and creating software artifacts and interactive prototypes. The proposed approach was tested with the dataset described in Section 5 and contrasted to the traditional requirement modeling approaches.

6.1 Requirement Extraction Performance

The performance for the semantic extraction module was assessed using the commonly used information retrieval metrics: Accuracy, Precision, Recall and F1-Score. These are metrics for evaluating the framework's accuracy in recognizing entities, attributes, actions, actors, constraints, and semantic relations from natural language requirements.

Table 3 Requirement Extraction Performance

Metric	Value
Accuracy	93.0%
Precision	92.4%
Recall	91.6%
F1-Score	92.0%

The results show that the proposed scheme has an excellent semantic extraction performance on the measured data set. The balanced Precision and Recall values suggest that the framework works effectively in detecting relevant requirement elements, while also reducing the number of false detections.

To evaluate the classification performance in more detail a confusion matrix was built up with the expert validated annotations as a reference data set.

Table 4 Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	110	10
Actual Negative	5	95

The results of the confusion matrix validate that most of the semantic elements were correctly recognised by the framework. The proposed NLP pipeline showed that the extraction accuracy was 93% with a relatively small number of misclassifications, proving the effectiveness of the NLP pipeline.

6.2 Prototype Generation Efficiency

The main goal of the proposed framework is to lower the work and time needed to convert text-based specifications to a software design artifact and functional prototype. To assess efficiency, the proposed framework was compared with the requirement modeling approaches, such as manual requirement modeling and NLP-based requirement modeling approaches..

Table 5. Prototype Generation Time Comparison

Approach	Average Time (Minutes)
Manual Modeling	84
NLP-Assisted Modeling	58
Proposed Framework	42

Table 5 shows the results obtained, indicating that the proposed framework significantly reduced the time required to create prototypes compared to the two main approaches. We note that automating semantic extraction, element creation, and interactive interface construction significantly contributed to accelerating the development process while maintaining the consistency of generated elements with requirements.

Table 6.Modelling Effort Comparison

Approach	Average Effort (Hours)
Manual Modeling	10.4
Proposed Framework	5.3

$$MER = \frac{10.4 - 5.3}{10.4} \times 100MER = 49.04\%$$

The results obtained from the equation show that the proposed framework reduced the human manual modeling effort by approximately 49%. This percentage demonstrates the benefits of integrating semantic NLP analysis with the creation of automated prototypes.

6.3 Comparative and Statistical Analysis

The competitiveness of the proposed framework is evaluated in comparison with the results obtained from representative studies in natural language processing-based requirements engineering.

Table 7 Comparison with Existing Approaches

Study	Accuracy	Automated Prototype Generation
Zhao et al.	84%	No
Javed and Lin	88%	No
Zaki-Ismail et al.	89%	Partial
Proposed Framework	93%	Yes

Through comparison, the proposed framework demonstrates the highest accuracy in semantic extraction while simultaneously improving the capabilities of automatic prototype generation. This contrasts with current approaches, which primarily focus on requirements interpretation.

The current proposal supports the complete transformation from textual requirements into interpretable and implementable prototypes.

To confirm whether the observed improvements are statistically significant, (H_0) a null hypothesis test was conducted, assuming no significant differences between the proposed framework and existing approaches. An alternative hypothesis was also tested, (H_1) assuming significant improvements.

Table 8 statistical Significance Analysis

Metric	p-value
Accuracy Improvement	0.014
Prototype Generation Time	0.009
Modeling Effort Reduction	0.011

All the p-values are below 0.05 and therefore the null hypothesis is rejected (table 8). This means that there are statistically significant improvements as a consequence of the proposed framework.

6.4 Threats to Validity

There are several limitations to the interpretation of the results of this study. First, only 120 requirements statements from a wide range of application areas were taken from the evaluation set. This type of data is adequate to demonstrate the viability of this concept, but more extensive industrial data could yield more insights regarding its scalability and generalizability. Second, the empirical evaluation used descriptions/explanations from experts, so there might be minor differences in the estimated interpretations. Third, the requirements evaluated were mostly academic and publicly available requirements, not from large-scale industrial projects. Finally, even if statistical significance tests have been carried out, larger samples and more realistic industrial conditions in future studies will yield more definite empirical evidence and provide support for the robustness and usefulness of the suggested approach.

Even with these shortcomings, the results show that the proposed framework of semantic NLP is capable of alleviating the semantic gap between textual requirements and generation of interactive prototypes. This has significantly helped enhance the semantic extraction performance, cut down the software development effort and speed up the speed of software design activities in the initial stage.

7 Conclusion and Future Work

This paper provided a semantic approach using NLP to automatically convert natural language software requirements to structured software artifacts and to create interactive system prototypes. The proposed

framework combines understanding of requirements, semantic modeling, generation of artifacts and development of prototypes in one development process. This framework introduces an enriched semantic representation and a semantic mapping mechanism to provide a direct mapping path from the textual requirements to executable prototype representation. The experimental evaluation results showed that the proposed scheme can extract the basic information about the product with an accuracy of 93.0%, and the values of Precision, Recall, and F1-score are greater than 91%. The results also indicated that prototype generation time is significantly reduced and approximately 49% of the manual modeling effort is saved as compared to the traditional requirement engineering practices. These enhancements were statistically validated and thus proved to be effective in the proposed approach. While these are encouraging findings, there are some shortcomings. Although these results are encouraging and positive, there are some shortcomings. The assessment presented in this paper was conducted on a proof-of-concept dataset comprising 120 statements of requirements gathered from various applied fields. Although there is a wide variety of this existing dataset, there is a need for more industrial datasets to better understand the scalability, robustness and adaptability of the framework to various domains. For this reason, future research is planned to focus on the combination of transformer-based language models and Large Language Models (LLMs) for semantic interpretation and contextual understanding. Moreover, future work will involve scaling up the system to implement automated architectural model generation, enhance the UI synthesis mechanisms and perform extensive industrial validation to guarantee the proposed framework's potential for integration into real software engineering settings.

REFERENCES

- [1] Diamantopoulos, T., Roth, M., Symeonidis, A., & Klein, E. (2017). Software requirements as an application domain for natural language processing. *Language Resources and Evaluation*, 51(2), 495-524.
- [2] Shah, U. S., & Jinwala, D. C. (2015). Resolving ambiguities in natural language software requirements: a comprehensive survey. *ACM SIGSOFT Software Engineering Notes*, 40(5), 1-7.
- [3] Letsholo, K. J. (2024). Early-Stage Requirements Transformation Approaches: A Systematic Review. *arXiv preprint arXiv:2408.05221*.
- [4] Gobov, D., & Zuieva, O. (2025). Software quality attributes in requirements engineering. *Int. J. Inf. Technol. Comput. Sci.*, 17(4), 38-48.
- [5] Izhar, R., Cosh, K., Ramingwong, L., Ramingwong, S., & Bhatti, S. N. (2025). An Efficient Methodology for the Categorization of Software Requirements Using Natural Language Processing and Similarity Analysis. *IEEE Access*.
- [6] Johnson, K., Morais, C., & Patelli, E. (2025). Enhancing procedure quality: Advanced language tools for identifying ambiguity and high-potential violation triggers. *Reliability Engineering & System Safety*, 264, 111308.
- [7] Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., & Karlsson, J. (2002). A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1), 20-33.
- [8] Al-Husseini, K. A., & Obaid, A. H. (2019). Analysis and risk management in software development using the logical-algebraic model. In *CEUR Workshop Proceedings* (Vol. 2475, pp. 241-248).
- [9] Borjigin, S. G., Austin, M. A., & Zontek-Carney, E. J. (2022). Semiautomated development of textual requirements: Combined NLP and multidomain semantic modeling approach. *Journal of Management in Engineering*, 38(5), 04022053.
- [10] Obayes, K. A., & Hamzah, A. (2022). Using of prototyping in develop an employee information management. *Measurement: Sensors*, 24, 100557.
- [11] Camburn, B., Viswanathan, V., Linsey, J., Anderson, D., Jensen, D., Crawford, R., ... & Wood, K. (2017). Design prototyping methods: state of the art in strategies, techniques, and guidelines. *Design Science*, 3, e13.
- [12] Zhang, H., Chen, P., Xie, X., Lin, C., Liu, L., Li, Z., ... & Sun, L. (2024, October). ProtoDreamer: a mixed-prototype tool combining physical model and generative AI to support conceptual design. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (pp. 1-18).
- [13] Jia, H., Morris, R., Ye, H., Sarro, F. & Mehtaev, S. (2025). *Automated Repair of Ambiguous Natural Language Requirements*. arXiv preprint. <https://doi.org/10.48550/arXiv.2505.07270>.
- [14] Necula, S., Dumitriu, F. & Greavu-Şerban, V. (2024). A Systematic Literature Review on Using Natural Language Processing in Software Requirements Engineering. *Electronics* 13(11). <https://doi.org/10.3390/electronics13112055>
- [15] Javed, M. & Lin, Y. (2020). *iMER: Iterative Process of Entity Relationship and Business Process Models Extraction from the Requirements*. arXiv preprint arXiv:2008.02502. <https://doi.org/10.48550/arXiv.2008.02502>

- [16] Zaki-Ismail, A., Osama, M., Abdelrazek, M., Grundy, J. & Ibrahim, A. (2020). *RCM: Requirement Capturing Model for Automated Requirements Formalisation*. arXiv preprint arXiv:2009.14683. <https://doi.org/10.48550/arXiv.2009.14683>
- [17] Woensel, W. V. & Motie, S. (2024). *NLP4PBM: A Systematic Review on Process Extraction using Natural Language Processing with Rule-based, Machine and Deep Learning Methods*. Enterprise Information Systems 18(11). <https://doi.org/10.1080/17517575.2024.2417404>
- [18] Fischbach, J., Hauptmann, B., Konwitschny, L., Spies, D. & Vogelsang, A. (2020). *Towards Causality Extraction from Requirements*. arXiv preprint arXiv:2006.15871. <https://doi.org/10.48550/arXiv.2006.15871>
- [19] Abad, Z. S., Gervasi, V., Zowghi, D. & Barker, K. (2018). *ELICA: An Automated Tool for Dynamic Extraction of Requirements Relevant Information*. arXiv preprint arXiv:1808.05857. <https://doi.org/10.48550/arXiv.1808.05857>
- [20] Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E. & Batista-Navarro, R. T. (2020). *Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study*. Software Systems. <https://doi.org/10.1007/s10270-020-00756-0>
- [21] (2023). *An empirically based model of software prototyping: a mapping study and a multi-case study*. Empirical Software Engineering. <https://doi.org/10.1007/s10664-023-10331-w>
- [22] (2025). *Digital variant design V-model for rapid product development*. The International Journal of Advanced Manufacturing Technology 139. <https://doi.org/10.1007/s00170-025-15982-1>
- [23] Yang, Y., Li, X., Liu, Z., Ke, W., Zu, Q. & Chen, X. (2018). *Automated Prototype Generation from Formal Requirements Model*. arXiv preprint arXiv:1808.10657. <https://doi.org/10.48550/arXiv.1808.10657>
- [24] Fávero, E. M. & Casanova, D. (2021). *BERT_SE: A Pre-trained Language Representation Model for Software Engineering*. arXiv preprint arXiv:2112.00699. <https://doi.org/10.48550/arXiv.2112.00699>
- [25] (2025). *Ontology-based NLP tool for tracing software requirements and conceptual models: an empirical study*. Requirements Engineering 30(2), pp. 123-145. <https://doi.org/10.1007/s00766-025-00447-4>
- [26] Al-Husseini, K. A. O., & Obaid, A. H. (2020). Interaction between project tasks and risk management tasks in software development. *Periodicals of Engineering and Natural Sciences (PEN)*, 8(4), 2300-2308.