eISSN 2783-6851

VILNIUS TECH
Vilnius Gediminas Technical University

NEW TRENDS in COMPUTER SCIENCES

2025
Volume 3
Issue 2
Pages 154–172
https://doi.org/10.3846/ntcs.2025.26832

FUZZY-BASED ASSESSMENT OF STAKEHOLDER FEEDBACK IN SOFTWARE REQUIREMENTS

Karolis TRINKŪNAS ✉, Jolanta MILIAUSKAITĖ

Department of Information Systems, Vilnius Gediminas Technical University, Vilnius, Lithuania

Article History:
- received 27 April 2026
- accepted 20 May 2026

Abstract. Requirements engineering is essential for aligning software systems with stakeholder needs, yet incorporating stakeholder feedback into existing software requirements remains challenging due to ambiguity, inconsistency, and limited validation mechanisms. This paper proposes an AI-based evaluation approach for assessing the quality of software requirements and the correctness of feedback-driven modifications. The method combines natural language processing, large language models, and a hierarchical fuzzy inference system to support systematic evaluation. A multi-label text classification model is trained to identify common requirement defects, including ambiguity, subjectivity, vagueness, non-verifiability, and negativity. These indicators are aggregated using a fuzzy inference hierarchy to compute requirement quality scores and evaluate the preservation and correctness of changes introduced by stakeholder feedback. The approach is implemented as a web-based software system and evaluated against multiple AI-based methods and human assessments using defined quality criteria and performance metrics. Results indicate that the proposed hybrid method provides consistent and interpretable evaluations and can support more structured assessment of stakeholder feedback implementation in requirements engineering.

Keywords: requirement quality, modification, stakeholder, feedback, FIS, LLM, AI, text classification, assessment.

✉Corresponding author. E-mail: ***karolis.trinkunas@stud.vilniustech.lt***

1. Introduction

Requirements engineering (RE) plays a critical role in software development by defining, analyzing, and validating stakeholder needs that guide system design and implementation (ISO/IEC/IEEE 29148:2018 (International Organization for Standardization, 2018). Despite its importance, RE frequently suffers from issues such as ambiguity, inconsistency, subjectivity, and incompleteness, particularly in large-scale and evolving systems ((International Organization for Standardization, 2018). These problems are well documented in both industrial and academic contexts (Papapanos & Pfeifer, 2023; Zhao et al., 2020). Norheim et al. (2024) further emphasize that such issues are exacerbated as requirements are iteratively refined over time. When stakeholder feedback is incorporated into existing requirements, the risk of introducing new defects or degrading requirement quality further increases (Femmer et al., 2014).

Recent advances in artificial intelligence (AI), especially large language models (LLMs) such as GPT-based architectures and transformer models like BERT, have demonstrated significant potential in supporting RE activities (Devlin et al., 2018). A recent systematic review further shows that generative AI is increasingly being applied across RE tasks, although research remains unevenly distributed across RE phases and industrial adoption is still limited (Cheng

et al., 2026). Prior research shows that AI techniques can assist in requirements classification, ambiguity detection, refinement of natural language (NL) specifications, and analysis of large volumes of unstructured textual input (Binder & Mezhuyev, 2024; Krishna et al., 2024). Recent deep-learning work has further shown that requirement-smell detection can be formulated as a multi-label classification task, allowing multiple defects to be detected within a single requirement statement (Alem et al., 2025). Vogelsang and Fischbach (2024) further distinguish between RE-related understanding tasks, such as classification and traceability, and generation tasks, such as requirements completion and test case generation. These approaches improve efficiency and scalability compared to traditional manual methods. Recent work on product requirements has shown that LLMs can support requirement reformulation and quality assessment against predefined quality criteria (Ellsel & Stark, 2025). However, existing approaches still primarily focus on standalone requirement or user-story quality rather than on evaluating the quality of requirement changes introduced through stakeholder feedback. Recent review evidence also indicates that AI-based requirements quality assessment still faces limitations related to standardized evaluation procedures, reproducible datasets, and integration into real requirements engineering workflows (Wolf et al., 2026).

In practice, stakeholder feedback is often informal, incomplete, or conflicting, making its integration into existing requirements difficult to assess systematically. Manual validation of feedback-driven changes is time-consuming and highly dependent on analyst expertise, while existing AI-based approaches provide limited support for evaluating whether such changes preserve requirement correctness or improve overall quality (Norheim et al., 2024; Sami et al., 2024). Although AI has been applied to detect ambiguities and inconsistencies in standalone requirements, comparatively little attention has been paid to evaluating the correctness of requirement modifications resulting from stakeholder feedback (Binder & Mezhuyev, 2024; Zhao et al., 2020).

Existing AI-based requirement quality assessment methods commonly evaluate individual requirement statements in isolation. However, stakeholder feedback implementation is a transformation-oriented task: the revised requirement must be judged in relation to both the original requirement and the requested change. A modified requirement may appear clear and well-formed while still failing to implement the requested stakeholder feedback, or it may implement the requested change while unintentionally removing important original requirement content. Therefore, evaluating stakeholder-driven requirement modifications requires criteria that go beyond standalone requirement quality assessment.

This paper addresses this gap by proposing a hybrid evaluation framework for assessing stakeholder-driven software requirement modifications. Unlike approaches that evaluate isolated requirement statements, the proposed framework evaluates the transformation from an original requirement to a modified requirement under a specific stakeholder change request. The framework assesses whether the requested change has been correctly implemented, whether unaffected original requirement content has been preserved, and whether the resulting requirement contains linguistic quality defects. To support this assessment, the framework integrates multi-label requirement defect classification, instruction-driven LLM-based semantic evaluation, and a hierarchical fuzzy inference system that aggregates defect severity and modification correctness into an interpretable Requirement Quality Index.

The remainder of this paper is structured as follows. Section 2 describes the materials and methods, including the system architecture, dataset preparation, requirement-modification examples, defect classification model, LLM-based correctness evaluation, fuzzy inference system, and evaluation procedure. Section 3 presents the experimental results. Section 4 discusses the findings, implications, limitations, and future work. Section 5 concludes the paper.

2. Materials and methods

This section describes the materials and methods used to develop and evaluate the proposed AI-supported requirement modification assessment framework. It presents the system architecture, dataset preparation process, requirement-modification examples, defect classification model, LLM-based correctness evaluation, fuzzy inference-based quality assessment, and experimental evaluation procedure.

2.1. System architecture

The proposed framework was implemented as a web-based, service-oriented evaluation pipeline that integrates machine learning, LLM-based assessment, fuzzy reasoning, and user interaction to evaluate stakeholder feedback implementation in software requirements (see Figure 1). The pipeline is methodologically consistent with recent work that frames LLM integration into requirements engineering as a structured process involving input definition, model and prompt configuration, application, and evaluation (Stein et al., 2026).

Users submit original requirements, modification requests, and revised requirements through a React-based web application, which communicates with a NestJS backend acting as an API orchestrator. The backend coordinates evaluation requests to a Python-based evaluation service, which invokes Hugging Face–hosted models, including a multi-label text classification model for detecting requirement defects and an instruction-driven large language model for assessing Preservation Correctness and Change Correctness. Defect scores and correctness scores are normalized to the range [0,1] and aggregated using a hierarchical fuzzy inference system to produce final requirement quality metrics. Evaluation results and associated quality

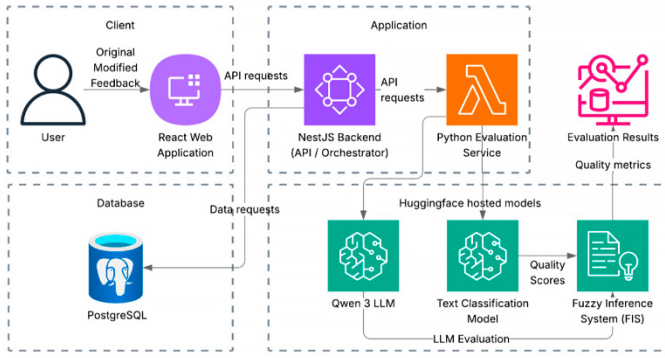

Figure 1. Proposed system architecture

scores are stored in a PostgreSQL database and returned to the user through the web interface. The proposed system architecture, especially the Hugging Face-hosted models presented in Figure 1, is described in more detail in the following sections.

2.2. Dataset preparation and label harmonization

The corpus for the multi-label defect classification model was constructed from four subsets of the Automatic Requirements Testability Analysis Dataset (ARTA), a publicly available requirements-smell dataset (Zakeri-Nasrabadi & Parsa, 2024). The selected subsets contain natural-language software requirements annotated with linguistic indicators related to requirement quality. Because the original subsets use partly different smell labels, naming conventions, and annotation schemes, the labels were manually inspected, normalized, and mapped to five canonical defect categories used in this study: Subjective, Ambiguous, Nonverifiable, Negative, and Vague. These categories are consistent with recent requirement-smell taxonomies that include subjective language, ambiguous expressions, negative statements, vague pronouns, and open-ended or non-verifiable terms (Alem et al., 2025).

The datasets used for corpus construction are summarized in Table 1. The table reports the dataset identifier, original number of requirements, number of source projects, original smell labels, and the canonical labels used in this study.

To ensure consistency across the combined corpus, original ARTA labels and representative linguistic cues were mapped to the five canonical defect categories according to their semantic meaning and the detection strategy implied by the original smell definition. Labels that did not correspond to the selected taxonomy were excluded from the final corpus. Table 2 summarizes the resulting mapping.

The resulting harmonized label dictionary associated each original label or linguistic cue with a single canonical defect category, improving consistency across the combined corpus and reducing noise from inconsistent label names, overlapping smell definitions, and

Table 1. Source datasets used for corpus construction

Dataset	Original size	No. of Projects	Original labels	Labels used in this study
DS1	985	6	10 smell labels, including subjective, ambiguous, loophole, nonverifiable, negative, vague pronoun, uncertain verb, and polysemy	Subjective, Ambiguous, Nonverifiable, Negative, Vague
DS2	1,092	8	8 smell labels: subjective, ambiguous, loopholes, open-ended, superlatives, comparatives, negative statements, vague pronouns	Subjective, Ambiguous, Negative, Vague
DS3	1,522	6	Same schema as DS2	Subjective, Ambiguous, Negative, Vague
DS4	1,153	4	8 smell labels with inconsistent naming and spelling; includes combined open-ended/non-verifiable label	Subjective, Ambiguous, Nonverifiable, Negative, Vague

Table 2. Mapping from original ARTA labels to canonical defect categories.

Canonical category	ARTA source label(s)	Mapping rule	Example indicators
Subjective	Subjective language	Dictionary / lexical-semantic	user-friendly
Ambiguous	Ambiguous adverb/ adjective	Dictionary / lexical-semantic	almost
Nonverifiable	Non-verifiable terms; Open-ended; Loopholes	Dictionary / lexical-semantic; merged from related ARTA labels	sufficient; as far as possible; as needed
Negative	Negative statement	POS / rule-based normalization	must not; not; never; without
Vague	Vague pronouns	POS / pronoun-reference based	which; it; this; that; they, when the referent is unclear

dataset-specific annotation conventions. Before model training, all requirement statements were pre-processed using a consistent protocol: conversion to plain text; removal of duplicates, metadata, identifiers, comments, and empty fields; normalization of punctuation, quotation marks, capitalization, and whitespace; preservation of negation terms, modal verbs, vague quantifiers, and subjective adjectives; and removal of entries above 32 words. Stopword removal and stemming were not applied, because these operations could remove linguistically meaningful requirement-quality cues.

Each requirement was represented using binary labels indicating the presence or absence of the five canonical defect categories. To reduce class imbalance, the requirements were grouped by canonical defect labels and split into training, validation, and test subsets using an approximate 70/15/15 ratio. Exact balance could not be achieved because of differences in the number of available examples per category, and label co-occurrence patterns were not explicitly considered because multi-label requirements represented only a small portion of the corpus.

Augmented samples used later during classifier training were generated only after the train/validation/test split and included only in the training subset. The validation and test subsets were not augmented, preventing leakage of augmented or duplicated variants into evaluation data. The test subset served as the final held out set for assessing generalization to previously unseen requirements. The final distribution of defect categories across the training, validation, and test subsets is presented in Table 3.

The values in parentheses indicate the number of unique requirement statements associated with the corresponding defect category. The dataset reflects a naturally imbalanced distribution, with Ambiguous and Subjective defects occurring most frequently, while Nonverifiable and Negative instances are comparatively less common. This imbalance motivated the use of imbalance-aware splitting and targeted augmentation during classifier training.

Table 3. Defect category distribution after label harmonization.

Defect Category	Train	Validation	Test	Total label assignments
Subjective	264 (80 unique)	51 (36 unique)	53 (41 unique)	368
Ambiguous	345 (140 unique)	76 (58 unique)	82 (59 unique)	503
Nonverifiable	54 (21 unique)	17 (10 unique)	11 (7 unique)	82
Negative	97 (4 unique)	15 (4 unique)	21 (2 unique)	133
Vague	113 (18 unique)	22 (7 unique)	35 (9 unique)	170
Total label Assignments (Unique requirement statements)	873 (263 unique)	181 (115 unique)	202 (118 unique)	

2.3. Requirement-modification examples

The evaluation dataset consisted of 20 manually prepared requirement-modification cases. Each case included an original requirement and a requested stakeholder modification. The modified requirements were then produced by either human participants or AI-based approaches and evaluated using the proposed framework.

To make the experimental material more transparent, Table 4 presents five representative requirement-modification examples used in the evaluation. The selected examples cover different types of stakeholder feedback, including adding missing information, restricting requirement scope, introducing measurable constraints, improving verifiability, and extending functional behaviour.

These examples demonstrate the two-part evaluation logic used in the proposed framework. First, Change Correctness assesses whether the requested stakeholder modification is reflected in the revised requirement. Second, Preservation Correctness assesses whether the parts of the original requirement that are outside the scope of the requested modification remain semantically preserved. This distinction is necessary because a modified requirement may correctly implement the requested change while still weakening or removing important original requirement content.

Table 4. Representative requirement-modification examples

Case	Original requirement	Requested change	Example modified requirement
1	Display “Logged Out” on user exit.	Add logout timestamp.	Display “Logged Out” with the exact logout timestamp on user exit.
2	Link project proposal information with related records.	Link only approved proposals.	Link approved project proposal information with related records.
4	Perform periodic backups via live synchronization.	Run every 15 minutes and store remotely.	Perform live-synchronized backups every 15 minutes and store them on a remote server.
9	Optimize loading speed on devices.	Define load time under 2 seconds.	Load on devices within 2 seconds.
11	Process account update request, verify input, and confirm completion.	Add rollback if update fails.	Process the account update request, verify input, roll back failed updates, and confirm successful completion.

2.4. Defect classification and change correctness evaluation

The proposed framework uses two AI-supported evaluation components. The first component detects requirement quality defects through multi-label text classification. The second component evaluates the correctness of stakeholder-driven requirement modifications using an instruction-driven LLM evaluator.

The multi-label classification model was trained to identify five requirement defect categories: Subjective, Ambiguous, Nonverifiable, Negative, and Vague. Each requirement statement was represented using binary labels indicating the presence or absence of these defect categories. The final selected configuration used the sentence-transformers/all-MiniLM-L6-v2 encoder, one training epoch, 100 SetFit iterations, a batch size of 32, and a learning rate of 2×10^{-5}. The classifier head was implemented as a weighted binary-relevance logistic regression layer, where one logistic regression classifier was trained independently for each defect category. This design is suitable for multi-label requirement-smell detection because a single requirement statement may contain more than one smell category (Alem et al., 2025). Scikit-learn's balanced class weighting was used, and rows without any positive smell label were down-weighted using a sample weight of 0.5.

Final label assignment was performed using label-specific thresholds tuned on the validation subset by maximizing per-label F1 over thresholds from 0.05 to 0.95. Hint-aware augmentation was applied only to the training subset after the train/validation/test split. The validation and test subsets were not augmented and were used only for model selection and final evaluation. This design prevented augmented variants of the same requirement from leaking into evaluation data and producing overly optimistic performance estimates. The selected classifier training configuration is summarized in Table 5.

Table 5. Classifier training configuration

Parameter	Value
Model architecture	SetFit with SentenceTransformer encoder and weighted binary-relevance logistic regression head
Base encoder	sentence-transformers/all-MiniLM-L6-v2
Target labels	Subjective, Ambiguous, Nonverifiable, Negative, Vague
Dataset split	Training / validation / test = 70/15/15
Random seed	4
Training epochs	1
SetFit iterations	100
Batch size	32
Learning rate	(2 \times 10^{-5})
Class weighting	Balanced class weights
Zero-label row weight	0.5
Augmentation scope	Training subset only
Augmentation type	Counterfactual removal and hint-only variants
Threshold tuning	Label-specific thresholds selected on the validation subset
Threshold search range	0.05–0.95 in steps of 0.05
Model selection metric	Macro-F1 on the validation subset

To improve sensitivity to linguistic signals, the training data was enriched with linguistic hints consisting of words or expressions directly associated with specific defect categories. In addition, a curated dictionary of known requirement smells was integrated as an external knowledge source (Zakeri-Nasrabadi & Parsa, 2024). Hint-aware data augmentation was applied by generating counterfactual samples, in which quality-indicative terms were removed, and hint-only samples, which emphasized category-relevant cues. This allowed the model to better learn contextual associations related to requirement quality defects.

Requirement modifications were evaluated using the Qwen/Qwen3-4B-Instruct-2507 model under a structured instruction-engineering configuration. This use of an LLM as a quality-oriented evaluator is aligned with recent work showing that LLMs can support the assessment of textual requirements against predefined quality criteria (Ellsel & Stark, 2025). The model was not treated as an autonomous correctness authority, but as a constrained comparative evaluator operating under explicitly defined scoring rules. It received three inputs: the original requirement, the requested modification, and the modified requirement. It then produced two normalized scores: Preservation Correctness and Change Correctness, each defined on the interval [0,1].

Preservation Correctness measures whether the modified requirement preserves the valid content and intent of the original requirement outside the scope of the requested modification. Change Correctness measures whether the requested stakeholder change is correctly implemented. The evaluation template defines anchor ranges for both criteria, where 1.0 indicates full preservation or full application, 0.5 indicates partial preservation or partial application, and 0.0 indicates that the content was lost, contradicted, or not implemented. If the requested change is not implemented, the Change Correctness score is constrained toward zero. If unrelated parts of the original requirement are altered, removed, or semantically degraded outside the modification scope, the Preservation Correctness score is proportionally reduced.

The evaluation focuses on semantic satisfaction of the requested change rather than simple keyword overlap. A fixed numerical JSON output format is enforced to ensure consistent machine-readable scoring. The LLM-generated Preservation Correctness and Change Correctness scores are then used as framework inputs and integrated into the hierarchical fuzzy inference system, where they contribute to the final requirement quality assessment.

2.5. Fuzzy inference-based quality assessment

Requirement quality is inherently multi-dimensional and involves linguistic uncertainty as well as partial correctness. Fuzzy inference systems are suitable for this type of assessment because they can represent uncertain, subjective, and potentially conflicting software-quality attributes more flexibly than crisp scoring methods (Barzegar et al., 2025). To integrate heterogeneous evaluation signals, such as requirement defects and the correctness of stakeholder-driven changes, a Mamdani-type fuzzy inference system (FIS) was adopted as the final decision layer of the proposed framework. Figure 2 presents the principal structure of the FIS.

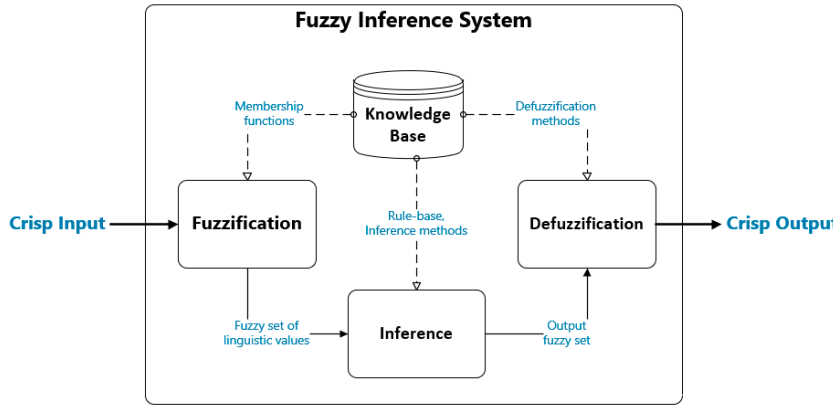

Figure 2. FIS principal schema

A Mamdani FIS was selected due to its transparency and linguistic expressiveness, which are important in requirements engineering contexts where evaluation results must be interpretable by human reviewers (Alonso & Magdalena, 2011; Dutu et al., 2017). As shown in Figure 2, the FIS consists of four main components: fuzzification, inference, defuzzification, and a knowledge base.

During fuzzification, crisp input values are converted into fuzzy sets. For example, the triangular membership function is defined in Eq. (1):

$$\mu_A\left(x;a_1,a_2,a_3\right)=\begin{cases}0, & if\ x\le a\\ \dfrac{x-a_1}{a_2-a_1}, & if\ a_1<x\le a_2\\ \dfrac{a_3-x}{a_3-a_2}, & if\ a_2<x<a_3\\ 0, & if\ x\ge a_3\end{cases}, \tag{1}$$

where a_1, a_2, and a_3 are parameters of the triangular membership function, $\mu_A(x)$ is the membership grade of x in A.

The inference stage performs fuzzy reasoning by applying fuzzy IF–THEN rules. A general fuzzy rule is expressed in Eq. (2):

$$\text{"If-Then" Rule: If } b_1 \text{ is } B_1^i \text{ AND } b_2 \text{ is } B_2^i \text{ AND} \ldots \text{AND } b_n \text{ is } B_n^i, \text{ THEN } c \text{ is } B^i, \tag{2}$$

where $b\in\mathbb{R}^n$ are input variables of fuzzy sets B_n^i.

The membership of the aggregated output across all rules is calculated using min-max inference, as shown in Eq. (3):

$$\mu_{B'}\left(y\right)=\max_{i=1,\ldots,n_R}\left\{\sup_{x\in X}\left(\mu_{A'}\left(x\right)\cdot\prod_{k=1}^{r}\mu_{A_{i,k}}\left(x_k\right)\cdot\mu_{B_i}\left(y\right)\right)\right\}, \tag{3}$$

where $\mu_{B'}\left(y\right)$ is the membership of the aggregated output fuzzy set B' at output value y, $\mu_{A'}\left(x\right)$ is the membership degree of the observed input fuzzy set A', n_R is the number of

fuzzy rules, and $\mu_{B_i}(y)$ is the membership degree of the consequent fuzzy set of rule *i*. The product operator models conjunction within rule antecedents, while the maximum operator aggregates contributions across rules.

During defuzzification, fuzzy output values are converted into crisp values. The knowledge base contains the fuzzy IF-THEN rule base and the database of membership functions used by the rules (Bardossy & Duckstein, 2022).

The final evaluation layer is implemented as a hierarchical Mamdani-type fuzzy inference system composed of three interconnected subsystems: **DefectSeverity**, **Correctness**, and **RequirementQuality**. Hierarchical fuzzy models have been used in software quality evaluation to decompose overall quality into lower-level characteristics and aggregate them into interpretable final quality scores (Barzegar et al., 2025). Following this logic, the proposed framework applies a hierarchical fuzzy structure to requirement modification assessment. These mappings are defined in Eqs. (4)–(6):

$$DS = F_{DS}(S, A, NV, N, V); \tag{4}$$

$$Corr = F_{Corr}(PC, CC); \tag{5}$$

$$RQI = F_{RQI}(DS, Corr), \tag{6}$$

where F_{DS}, F_{Corr}, and F_{RQI} are the mappings implemented by the DefectSeverity, Correctness, and RequirementQuality subsystems, respectively. The DefectSeverity score *DS* is calculated from Subjectivity (*S*), Ambiguity (*A*), Non-verifiability (*NV*), Negativity (*N*), and Vagueness (*V*). The Correctness score *Corr* is calculated from Preservation Correctness (*PC*)and Change Correctness (*CC*). The final Requirement Quality Index (*RQI*) is produced by combining *DS* and *Corr*. All inputs and outputs are normalized to the interval [0,1].

All subsystems use centroid defuzzification with min–max inference. Membership functions are defined using Z-shaped, S-shaped, triangular, and trapezoidal functions. These functions were selected to provide smooth transitions between adjacent linguistic levels while preserving interpretable boundary regions for clearly low and clearly high values.

The DefectSeverity subsystem aggregates five linguistic defect indicators - Subjective, Ambiguous, Nonverifiable, Negative, and Vague - into a single severity score. Treating these indicators as separate inputs is appropriate because requirement-smell categories may differ in perceived severity, frequency, and practical impact (Gentili & Falessi, 2025). Each input variable is modelled using three membership functions: Low, Average, and High. The output DefectSeverity variable also consists of three linguistic levels. This subsystem implements 12 fuzzy rules, with some rules weighted to emphasize stronger defect combinations and to distinguish isolated defects from compound requirement defects. This weighting reflects the assumption that different smell types and smell combinations may contribute unequally to perceived requirement quality. The membership functions used in the DefectSeverity subsystem are presented in Table 6.

Table 6. Membership functions for the DefectSeverity subsystem

ID	Parameter	Crisp	Linguistic Term	Membership Function	Universe of Discourse
A	Ambiguity	0.0 0.5 1.0	Low Average High	zmf [0.2 0.5] trimf [0.35 0.5 0.65] smf [0.55 0.75]	[0 1]
S	Subjectivity	0.0 0.5 1.0	Low Average High	zmf [0.25 0.55] trimf [0.4 0.55 0.7] smf [0.65 0.85]	[0 1]
N	Non-verifiability	0.0 0.5 1.0	Low Average High	zmf [0.2 0.5] trimf [0.35 0.5 0.65] smf [0.55 0.75]	[0 1]
V	Vagueness	0.0 0.5 1.0	Low Average High	zmf [0.2 0.5] trimf [0.35 0.5 0.65] smf [0.55 0.75]	[0 1]
Neg	Negativity	0.0 0.5 1.0	Low Average High	zmf [0.25 0.55] trimf [0.4 0.55 0.7] smf [0.6 0.8]	[0 1]
DS	Defect Severity (output)	0.0 0.5 1.0	Low Average High	trapmf [0 0 0.2 0.4] trimf [0.35 0.5 0.65] trapmf [0.6 0.8 1 1]	[0 1]

The Correctness subsystem integrates Preservation Correctness and Change Correctness into a unified Correctness score. Both inputs are represented using the linguistic levels Low, Average, and High. The output Correctness variable also uses three linguistic levels. This subsystem employs 7 weighted fuzzy rules, prioritizing high correctness when both preservation and change implementation are strong, and penalizing cases where either dimension is low. The membership functions used in the Correctness subsystem are shown in Table 7.

The RequirementQuality subsystem combines DefectSeverity and Correctness into the final RequirementQuality score. The two input variables use three linguistic levels: Low, Average, and High. The final output uses five linguistic levels: Very Low, Low, Average, High, and Very High. This subsystem implements 9 fuzzy rules that map combinations of defect severity and correctness into interpretable quality categories. The membership functions for the RequirementQuality subsystem are presented in Table 8.

Table 7. Membership functions for the Correctness subsystem

ID	Parameter	Crisp	Linguistic Term	Membership Function	Universe of Discourse
PC	Preservation Correctness	0.0 0.5 1.0	Low Average High	zmf [0.25 0.55] trimf [0.4 0.55 0.7] smf [0.65 0.85]	[0 1]
CC	Change Correctness	0.0 0.5 1.0	Low Average High	zmf [0.25 0.55] trimf [0.4 0.55 0.7] smf [0.65 0.85]	[0 1]
C	Correctness (output)	0.0 0.5 1.0	Low Average High	trapmf [0 0 0.2 0.4] trimf [0.35 0.5 0.65] trapmf [0.6 0.8 1 1]	[0 1]

Table 8. Membership functions for the Requirement Quality subsystem

ID	Parameter	Crisp	Linguistic Term	Membership Function	Universe of Discourse
DS	Defect Severity	0.0 0.5 1.0	Low Average High	trapmf [0 0 0.2 0.4] trimf [0.35 0.5 0.65] trapmf [0.6 0.8 1 1]	[0 1]
C	Correctness	0.0 0.5 1.0	Low Average High	trapmf [0 0 0.2 0.4] trimf [0.35 0.5 0.65] trapmf [0.6 0.8 1 1]	[0 1]
RQ	Requirement Quality (final output)	0.0 0.25 0.5 0.75 1.0	Very Low Low Average High Very High	trapmf [0 0 0.1 0.25] trimf [0.15 0.3 0.45] trimf [0.35 0.5 0.65] trimf [0.55 0.7 0.85] trapmf [0.75 0.9 1 1]	[0 1]

In total, the hierarchical FIS comprises 28 fuzzy rules across the three subsystems. This hierarchical design enables modular reasoning while avoiding the rule-combinatorial explosion that would arise in a flat seven-input fuzzy inference system. A flat system with three linguistic terms for each of seven inputs would require the number of rules shown in Eq. (7):

$$N_{flat} = T^n = 3^7 = 2187\,, \tag{7}$$

whereas the proposed hierarchical system requires only the number of rules shown in Eq. (8):

$$N_{hier} = R_{DS} + R_{Corr} + R_{RQI} = 12 + 7 + 9 = 28, \tag{8}$$

rules, where T is the number of linguistic terms per input variable, n is the number of input variables, and R_{DS}, R_{Corr}, and R_{RQI} denote the number of rules in the DefectSeverity, Correctness, and RequirementQuality subsystems.

The hierarchical structure separates linguistic defect impact from modification correctness, enhancing interpretability and traceability of intermediate reasoning stages. Both numerical outputs and linguistic memberships are preserved, allowing reviewers to inspect subsystem-level contributions to the final requirement quality score.

2.6. Experimental setup and evaluation metrics

The proposed framework was evaluated using requirement modifications produced by both human participants and AI-based approaches. All modified requirements were assessed using the same evaluation procedure to ensure comparability across methods. Using a standardized evaluation procedure is important when integrating LLMs into requirements engineering, because model outputs need to be assessed through defined metrics and expert-oriented review criteria (Stein et al., 2026).

The evaluation combined qualitative expert judgment with quantitative defect-resolution metrics. This follows prior work showing that requirement quality evaluation can benefit from both expert assessment and measurable defect detection. For example, Parrales-Bravo et al. (2024) evaluated ambiguity-detection training using expert-oriented learning outcomes, while

Zhao et al. (2020) used precision, recall, and F1-score to evaluate NLP-based detection of ambiguity and incompleteness.

In this study, requirement modifications were evaluated using two complementary strategies. First, an error-based assessment quantified incorrectly identified defects and correctly modified requirements against a reference set of defects established before the evaluation. Second, the resulting requirements were reviewed by a domain specialist who provided a qualitative clarity rating. Therefore, the evaluation should be understood as assessing the consistency and plausibility of requirement revisions relative to expert consensus, rather than establishing absolute ground-truth correctness.

The evaluation was performed using the proposed hierarchical fuzzy inference system, which produces a Requirement Quality Index. In the reported metrics, Change Quality denotes the final quality of the modified requirement after combining the linguistic defect assessment with the correctness of the implemented stakeholder change. It differs from Change Correctness, which only measures whether the requested modification was applied, and from Preservation Correctness, which measures whether unaffected original requirement content was preserved. The index was used to compare requirement modifications across two main conditions: requirements modified manually by human editors and requirements modified by large language models using different prompting strategies.

Each approach was evaluated according to three criteria:

1. Requirement quality before and after modification;
2. Correctness of feedback application;
3. Overall quality of the feedback integration process.

To ensure comparability, all LLMs received identical prompts when revising requirements. Human participants received a structured document containing the original requirement and the requested modification and were asked to produce the revised version.

The evaluation dataset consisted of 20 manually prepared requirement-modification cases selected to vary in difficulty and length. The cases included 10 easier and 10 more difficult modification tasks, with requirement statements ranging from 16 to 32 words. Each case was completed by 10 human participants, producing 200 human-generated modified requirements in total. These outputs were then evaluated together with the AI-generated modifications using the same assessment procedure. The evaluated approaches included requirements modified by human participants, requirements modified by humans with a software testing background, requirements modified by a PhD-level participant, and requirements modified by LLMs using zero-shot, few-shot, and prompt-engineered strategies.

Requirement quality was evaluated along five defect dimensions: Subjectivity, Ambiguity, Non-verifiability, Negativity, and Vagueness. These dimensions were aggregated into qualitative levels of Low, Average, and High. Preservation Correctness and Change Correctness were evaluated on the same scale and combined through the hierarchical fuzzy inference system to derive the final quality score.

All numerical values were normalized to the interval [0,1]. The output of the top-level fuzzy subsystem was converted into a 10-point score for each requirement by linear scaling, shown in Eq. (9):

$$Score_i = 10 * RQI_i, \tag{9}$$

where RQ_i is the Requirement Quality output for the i-th requirement. The total score for each evaluated approach was calculated as the sum of scores across all 20 requirements, shown in Eq. (10):

$$TotalScore = \sum_{i=1}^{20} Score_i \ , \ TotalScore_{\max} = 200 \, . \tag{10}$$

Thus, each requirement could receive a maximum score of 10 points, and each approach could receive a maximum total score of 200 points. For human participants, total scores were first calculated across the 20 cases for each participant and then averaged across participants; for AI-based approaches, total scores were calculated across the same 20 cases for each evaluated prompting strategy. The resulting scores were used to compare human and AI-based requirement modification approaches in the Results section.

3. Results

This section presents the experimental results of the proposed framework. First, the performance of the multi-label defect classification model is reported. Second, the evaluation results for human- and AI-generated requirement modifications are presented.

3.1. Multi-label classifier performance

The multi-label classifier was evaluated using standard multi-label classification metrics, supplemented by smell/no-smell F1 for binary defect detection. The results for the validation subset and final held-out test subset are summarized in Table 9.

Table 9. Multi-label classification performance

Metric	Validation	Test
Micro-F1	0.6965	0.6577
Macro-F1	0.7065	0.6276
Hamming Accuracy	0.9135	0.9004
Subset Accuracy	0.6939	0.6286
Smell/no-smell F1	0.8436	0.8392

Performance decreased moderately from validation to test, indicating some generalization loss. The larger drop in Macro-F1 than Micro-F1 suggests less stable performance across defect categories, especially for less frequent labels. High Hamming Accuracy shows that most label decisions were correct, while lower Subset Accuracy indicates that exact multi-label prediction remained more difficult. The high smell/no-smell F1 score suggests effective detection of requirements containing at least one defect. Overall, the classifier showed acceptable internal generalization within the five-category taxonomy, while retaining label-specific limitations. These results demonstrate the performance of the selected SetFit-based configuration

under the prepared dataset and evaluation protocol, but they should not be interpreted as evidence that this classifier is superior to alternative neural, transformer-based, or zero-shot LLM classification approaches.

3.2. Requirement modification evaluation results

The second evaluation assessed requirement modifications produced by human participants and AI-based approaches. The results were calculated using the proposed hierarchical fuzzy inference system and aggregated across the evaluated modifications. Table 10 reports the main defect-related metrics, Preservation Correctness, Change Correctness, Overall Correctness, Change Quality, and Total Score. As defined in the evaluation setup, Change Quality combines linguistic requirement quality with modification correctness, while Overall Correctness combines Preservation Correctness and Change Correctness.

Table 10. Aggregated metric performane across evaluated requirement modifications

Metric Category	Average		Difference
	Human	AI	
Subjectivity	18.64	18.41	−0.23
Ambiguity	14.10	13.40	−0.70
Non-verifiability	19.05	19.17	+0.12
Negativity	19.44	19.57	+0.13
Vagueness	17.95	17.37	−0.58
Requirement Quality	12.25	12.50	+0.25
Preservation Correctness	17.86	19.32	+1.46
Change Correctness	19.12	19.96	+0.84
Overall Correctness	18.05	19.56	+1.51
Change Quality	13.18	13.00	−0.18
Total Score	169.63	172.25	+2.62

The results show that AI-based approaches achieved slightly higher average total scores than human participants under the applied evaluation framework. AI approaches also achieved higher scores for Preservation Correctness, Change Correctness, and Overall Correctness. The largest difference appeared in Overall Correctness, where AI-based approaches scored 19.56 compared with 18.05 for human participants.

For linguistic defect dimensions, the differences were smaller. Human participants scored slightly higher on Subjectivity, Ambiguity, and Vagueness, while AI-based approaches scored slightly higher on Non-verifiability and Negativity. Requirement Quality was also slightly higher for AI-based approaches, with an average score of 12.50 compared with 12.25 for human participants.

Overall, the results indicate that AI-based approaches achieved slightly higher aggregate scores under the instruction-centric evaluation criteria, especially for preserving original requirement content and applying requested modifications. Because the evaluation was based on 20 distinct requirement-modification scenarios, although producing 200 human-generated

modified requirements, these findings should still be interpreted within the scope of the applied evaluation framework rather than as a general conclusion about human and AI performance in requirements engineering. Interpretation of these results is provided in the Discussion section.

4. Discussion

The results suggest that AI-based approaches achieved slightly higher total scores than human participants under the proposed evaluation framework. This difference was especially visible in Preservation Correctness, Change Correctness, and Overall Correctness. Although the evaluation included multiple human responses per case, the diversity of underlying modification scenarios was limited; therefore, the findings should be interpreted cautiously. The results indicate that AI-based methods may be effective when the requested modification is clearly defined and the task requires direct application of explicit instructions, but broader validation with more diverse requirement types and stakeholder-change scenarios is needed.

AI-generated modifications tended to preserve the original requirement more consistently than human-written modifications. This can be explained by the conservative editing behaviour of LLM-based approaches, which usually modify only the parts of the requirement directly affected by the instruction. As a result, AI-based approaches achieved higher Preservation Correctness scores, because they introduced fewer unrelated changes outside the requested modification scope.

Human participants showed greater variability in the evaluation results. This does not necessarily indicate weaker performance. In many cases, human editors may reinterpret, restructure, or improve a requirement to make it clearer or more aligned with domain expectations. However, the proposed evaluation framework prioritizes instruction-centered preservation. Therefore, broader restructuring may reduce the Preservation Correctness score even when the modified requirement is useful from a wider requirement engineering perspective.

The results also show that AI-based approaches achieved near-ceiling Change Correctness scores. This indicates that LLMs are well suited for applying explicit stakeholder modification instructions when the requested change is specific and unambiguous. However, this strength depends on the clarity of the input instruction. If stakeholder feedback is vague, incomplete, or conflicting, human interpretation remains necessary before AI-based modification can be applied reliably.

These findings suggest a practical division of work in stakeholder feedback implementation. Human expertise is most valuable during interpretive stages, where stakeholder intent must be clarified, conflicts must be resolved, and trade-offs must be negotiated. AI-based methods are most useful after these decisions are fixed, because they can consistently apply clearly defined changes while preserving the original requirement structure.

The proposed framework also provides interpretability benefits. This is consistent with fuzzy inference-based software quality assessment, where expert knowledge can be encoded through transparent membership functions and rule bases before being converted into numerical quality scores (Barzegar et al., 2025). In the present study, this principle is applied specifically to the assessment of stakeholder-driven requirement modifications. By separating

Defect Severity from Correctness, the framework can distinguish between a linguistically weak requirement and an incorrectly modified requirement. This distinction is useful because a requirement may be clear but fail to implement stakeholder feedback, or it may correctly implement feedback while still containing ambiguity, vagueness, or non-verifiable wording.

Several limitations should be considered. First, the evaluation dataset consisted of 20 manually prepared requirement-modification cases, which limits the generalizability of the results. Second, the framework evaluates consistency and plausibility relative to expert-defined criteria rather than establishing absolute ground-truth correctness. This limitation is consistent with broader challenges in AI-based requirements quality assessment, where evaluation procedures and benchmark datasets remain difficult to standardize across studies (Wolf et al., 2026). Third, the classifier was not compared against alternative neural, transformer-based, or LLM-based baselines. This limitation reflects a broader challenge in GenAI-supported requirements engineering, where fragmented benchmarking practices, limited shared datasets, and inconsistent evaluation procedures restrict comparability across studies (Cheng et al., 2026). Therefore, the reported classifier results demonstrate the internal performance of the selected SetFit-based approach but do not prove its superiority over other model architectures.

Another limitation concerns the dataset used for defect classification. The source datasets differ in annotation granularity, labelling objectives, and original defect taxonomies. Although label harmonization reduces inconsistency, some semantic loss is unavoidable when heterogeneous labels are mapped to a smaller set of canonical categories. Therefore, the classification results should be interpreted within the five-category defect taxonomy used in this study.

Future work should extend the evaluation to larger and more diverse requirement-modification datasets. Additional studies should compare the proposed classifier with alternative machine learning and LLM-based baselines, including fine-tuned transformer models and zero-shot LLM classifiers. Future versions of the framework should also include confidence and uncertainty estimates, external validation by multiple domain experts, and semi-automatic or data-driven refinement of fuzzy rules and membership functions. Further evaluation should examine the robustness of the framework under vague, incomplete, and conflicting stakeholder feedback. The framework could also be extended to structured requirement artifacts and integrated into practical requirement management workflows.

5. Conclusions

This paper proposed an AI-supported framework for evaluating stakeholder feedback implementation in software requirements. The framework focuses on requirement modification evaluation rather than full requirement generation. It combines multi-label requirement defect classification, instruction-driven LLM evaluation, and a hierarchical Mamdani-type fuzzy inference system to assess both requirement quality and modification correctness.

The proposed approach separates linguistic defect severity from modification correctness. This makes it possible to distinguish requirements that are linguistically weak from requirements that incorrectly implement stakeholder feedback. The framework therefore supports more transparent and traceable assessment of modified requirements.

The evaluation results indicate that AI-based approaches achieved slightly higher correctness and total quality scores than human participants under the applied instruction-centered evaluation criteria. These findings suggest that AI-based methods can support clearly defined requirement modifications, while human involvement remains important for interpreting stakeholder intent, resolving ambiguity, and making broader judgment-based improvements.

Overall, the proposed framework provides a lightweight and interpretable method for assessing stakeholder-driven requirement modifications. It can support requirements analysts by identifying linguistic defects, checking whether requested changes were correctly applied, and aggregating the results into an explainable quality score.

Author contributions

KT was responsible for manuscript preparation, data collection and preparation, design and development of proposed approach. JM was responsible for work supervision and review.

Disclosure statement

The authors state that there were no competing financial, professional or personal interests from other parties.

References

Alem, A. L., Gebretsadik, K. K., Mengistie, S. A., & Admas, M. F. (2025). Multi-label software requirement smells classification using deep learning. *Scientific Reports*, *15*, Article 5761. https://doi.org/10.1038/s41598-025-86673-w

Alonso, J. M., & Magdalena, L. (2011). Special issue on interpretable fuzzy systems. *Information Sciences*, *181*(20), 4331–4339. https://doi.org/10.1016/j.ins.2011.07.001

Bardossy, A., & Duckstein, L. (2022). *Fuzzy rule-based modeling with applications to geophysical, biological and engineering systems*. Taylor and Francis. https://doi.org/10.1201/9780138755133

Barzegar, A., Barzegar, Y., Verde, L., Bellini, F., Pisani, P., & Marrone, S. (2025). Measuring software product quality based on fuzzy inference system techniques in ISO standard. *Procedia Computer Science*, *270*, 6045–6054. https://doi.org/10.1016/j.procs.2025.10.074

Binder, M., & Mezhuyev, V. (2024). A framework for creating an IoT system specification with ChatGPT. *Internet of Things*, *27*, Article 101218. https://doi.org/10.1016/j.iot.2024.101218

Cheng, H., Husen, J. H., Lu, Y., Racharak, T., Yoshioka, N., Ubayashi, N., & Washizaki, H. (2026). Generative AI for requirements engineering: A systematic literature review. *Software – Practice and Experience*, *56*(2), 141–170. https://doi.org/10.1002/spe.70029

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*: *Human Language Technologies* (vol. 1, pp. 4171–4186). The Association for Computational Linguistics.

Dutu, L. C., Mauris, G., & Bolon, P. (2017). A fast and accurate rule-base generation method for mamdani fuzzy systems. *IEEE Transactions on Fuzzy Systems*, *26*(2), 715–733. https://doi.org/10.1109/TFUZZ.2017.2688349

Ellsel, C., & Stark, R. (2025). Advancing requirements engineering with large language models. *Procedia CIRP*, *136*, 701–706. https://doi.org/10.1016/j.procir.2025.08.120

Femmer, H., Mendez Fernandez, D., Juergens, E., Klose, M., Zimmer, I., & Zimmer, J. (2014). Rapid requirements checks with requirements smells: two case studies. In *Proceedings of the 1st International*

Workshop on Rapid Continuous Software Engineering. Association for Computing Machinery. https://doi.org/10.1145/2593812.2593817

Gentili, E., & Falessi, D. (2025). Practitioners' perceptions on requirements smells. *Information and Software Technology*, *187*, Article 107823. https://doi.org/10.1016/j.infsof.2025.107823

International Organization for Standardization. (2018). *Systems and software engineering – Life cycle processes – Requirements engineering* (standard No. ISO/IEC/IEEE 29148:2018). https://www.iso.org/standard/72089.html#lifecycle

Krishna, M., Gaur, B., Verma, A., & Jalote, P. (2024). Using LLMs in software requirements specifications: An empirical evaluation. In *Proceedings of the IEEE International Conference on Requirements Engineering* (pp. 475–483). IEEE. https://doi.org/10.1109/RE59067.2024.00056

Norheim, J. J., Rebentisch, E., Xiao, D., Draeger, L., Kerbrat, A., & De Weck, O. L. (2024). Challenges in applying large language models to requirements engineering tasks. *Design Science*, *10*, Article e16. https://doi.org/10.1017/dsj.2024.8

Papapanos, K., & Pfeifer, J. (2023). *A literature review on the impact of artificial intelligence in requirements elicitation and analysis* (MSc Thesis). Stockholm University. https://urn.kb.se/resolve?urn=urn:nbn:se:su:diva-219598

Parrales-Bravo, F., Gómez-Rodríguez, V., Chiquito-Vera, L., Rendón-Quijije, I., Caicedo-Quiroz, R., Tolozano-Benites, E., Vasquez-Cevallos, L., & Cevallos-Torres, L. (2024). DEAR: Detecting ambiguous requirements as a way to develop skills in requirement specifications. *Electronics*, *13*(15), Article 3079. https://doi.org/10.3390/electronics13153079

Sami, M. A., Rasheed, Z., Waseem, M., Zhang, Z., Herda, T., & Abrahamsson, P. (2024). *Prioritizing software requirements using large language models*. arXiv. https://doi.org/10.48550/arXiv.2405.01564

Stein, A., Mirzai, A., Axmann, J., & Vietor, T. (2026). Integrating the capabilities offered by large language models into the requirements engineering process. *Digital Engineering*, *10*, Article 100098. https://doi.org/10.1016/j.dte.2026.100098

Vogelsang, A., & Fischbach, J. (2024). *Using large language models for natural language processing tasks in requirements engineering*: *A Systematic Guideline*. arXiv. https://doi.org/10.48550/arXiv.2402.13823

Wolf, E., Trendowicz, A., & Siebert, J. (2026). Quality assessment of software requirements using artificial intelligence methods: A systematic literature review. *Information and Software Technology*, *191*, Article 107979. https://doi.org/10.1016/j.infsof.2025.107979

Zakeri-Nasrabadi, M., & Parsa, S. (2024). Natural language requirements testability measurement based on requirement smells. *Neural Computing and Applications*, *36*(21), 13051–13085. https://doi.org/10.1007/s00521-024-09730-x

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., & Batista-Navarro, R. T. (2020). *Natural Language Processing* (*NLP*) *for requirements engineering*: *A systematic mapping study*. arXiv. https://doi.org/10.48550/arXiv.2004.01099