

DOI: <https://doi.org/10.15276/ict.03.2026.52>
UDC004.41:004.8

Detecting contradictions in software system requirements using artificial intelligence

Mariia O. Popova

ORCID: <https://orcid.org/0000-0001-6836-3690>; popova.m.o@op.edu.ua
Odesa Polytechnic National University, 1, Shevchenko Av. Odesa, 65044, Ukraine

ABSTRACT

The paper addresses the detection of semantic contradictions in textual software requirements. Such contradictions occur when separate statements prescribe incompatible rules for the same action under the same conditions. They can be difficult to identify when requirements use different wording, include exceptions, or rely on implicit business rules. The study aims to define an analysis procedure that helps a requirements analyst identify and examine potentially conflicting pairs. The procedure extracts the actor, action, object, modality, applicable conditions, and numerical constraints from each requirement and then selects pairs related to the same function. Selection also considers whether requirements belong to the same process and whether their versions are current, since similar statements may describe different system states. A large language model is used to compare their meanings, while explicit rules check numerical boundaries and negation. Particular attention is paid to cases where a permission and a prohibition prove incompatible only after their time conditions or threshold values are compared. Each proposed finding must include references to the original passages, the shared condition, and an explanation of the possible conflict. This presentation makes the analysis traceable and allows the analyst to determine whether relevant context was lost during comparison. A medical appointment cancellation example demonstrates that similar wording alone cannot establish a contradiction without considering the scope of each rule. When the scope or exceptions remain unclear, the pair should be marked for clarification rather than treated as a confirmed contradiction. The analyst remains responsible for the final decision on whether the requirements are consistent. Once a conflict has been confirmed, the analyst should record the agreed rule and identify the specification statements that require revision. For future empirical assessment, the paper proposes an expert-annotated set of requirement pairs and the measurement of precision, recall, false-positive rate, and time needed to review a flagged pair. No quantitative performance results are reported at this stage.

Keywords: requirements engineering; contradictory requirements; software systems; large language models; semantic analysis; requirements specification; logical checking; expert analysis

For citation: Popova M. O. “Detecting contradictions in software system requirements using artificial intelligence”. *Informatics. Culture. Technology*. 2026; Vol. 3 No. 2 (4): 686–696. DOI: <https://doi.org/10.15276/ict.03.2026.52>

Relevance. Requirements for a software system often originate from various stakeholders and evolve throughout the project. In extensive specifications, mutually exclusive rules may appear in different sections, making them difficult to detect during routine reading. ISO/IEC/IEEE 29148:2018 defines requirements engineering processes and the requirements for their information outputs [1]. Research into natural language specifications indicates that the automated detection of quality defects is a crucial area for supporting the work of analysts [2], [3].

The use of large language models enables the comparison of phrasings that describe the same action using different words. However, a model might misidentify conditions pertaining to different roles or timeframes as conflicts, or fail to account for an exception stated in an adjacent clause. Such limitations have been documented in experimental studies on requirements verification [4], [5]. Consequently, the results of automated analysis should be treated as a valid signal prompting further review, rather than as a definitive decision to modify the specification.

Analysis of recent research and publications. A systematic review by L. Zhao et al. summarizes research on natural language processing in requirements engineering. The authors noted significant attention paid to detecting specification defects, alongside a small proportion of studies validated in industrial settings [3]. A. E. Gärtner, T.-A. Fay, and D. Göhlich proposed a classification of contradictions and a method for identifying them through sentence formalization and sequential check questions [6]. These works establish a basis for analysis: before comparing formulations, it is necessary to determine exactly which conditions and objects they describe.

The capabilities of language models for this task were investigated by A. Fanteki et al., who compared GPT-3.5’s findings regarding inconsistencies with expert assessments across several document examples [4]. In a study by T. Mahbub et al., the GPT-4 model was applied to an

industrial specification; the accuracy of inconsistency detection was 0.43, and the authors identified difficulties in establishing links between distant parts of the document and accounting for specific context as key limitations [5]. Consequently, any conflict detected by the model requires verification against the requirements text and domain-specific rules.

A. E. Gärtner and D. Göhlich combined a language model with formal logic to analyze requirements expressed in controlled natural language; in a comparative study, their proposed ALICE system outperformed an approach based solely on a language model [7]. M. Fazelnia et al. employed natural language inference to detect conflicts between requirements, achieving higher F1 scores than ChatGPT across seven datasets, although the authors did not consider the overall quality of performance on this task to be sufficiently high [8]. A. Ferrari and P. Spoletini outlined the potential of formal methods for verifying language model outputs in requirements engineering [10]. Collectively, these results suggest the value of combining semantic comparison, explicit constraint checking, and analyst-accessible explanations. This work focuses on the shared operational contexts of requirement pairs and the traceability of inferences back to the original formulations.

The scientific contribution of this work lies in the description of a procedure wherein a potential contradiction is treated as a verifiable pair of requirements with a clearly defined shared context. Unlike the ALICE system, which relies on controlled natural language and automated inference [7], the proposed procedure addresses freely formulated textual requirements and entails a final decision by the analyst. In a single verification cycle, it combines the determination of the scope and validity of the document versions, a three-state outcome ("contradiction," "consistent requirements," "insufficient context"), and a card containing precise specification excerpts, results of semantic matching and explicit rule-based checks, and the rationale for the decision. Individual components are known from previous studies [5], [7], [8]; the distinction lies in their integrated application and the traceability of the conclusion. The effectiveness of this combination still requires experimental validation.

Research objective. To establish a reproducible procedure for detecting contradictions between textual requirements for software systems, in which the results of semantic analysis can be verified against the original formulations and context.

Main body of the study. Developing a procedure for detecting contradictions requires defining the level at which specification consistency is verified. In this study, the unit of analysis is a pair of textual requirements, accompanied by document excerpts that explain the terms used and the scope of the relevant rules. The procedure focuses primarily on identifying pairwise contradictions: for any two active requirements, it determines whether their provisions can be satisfied simultaneously under shared conditions. This approach allows for the comparison of specific provisions while maintaining their link to the original source. The result of checking a single pair does not characterize the consistency of the entire specification, as other provisions may contain additional conditions or establish exceptions. Therefore, for each potential contradiction, it is necessary to reconstruct the scenario in which both requirements apply and determine the specific system actions they entail. The chosen level of analysis influences the selection of pairs, the content of the explanation, and the subsequent analyst review of the conclusion.

Textual requirements for a software system are formulated at various project stages and often originate from multiple stakeholders. Some provisions describe user actions, while others define access restrictions, operation timeframes, or exceptions to general rules. As the specification is refined, initial wordings may persist alongside newer versions, and interrelated conditions may end up in different sections of the document. Under these circumstances, examining a single sentence without considering its source and scope can lead to a misleading impression regarding the consistency of the requirements. Therefore, the subject of analysis is not only the linguistic similarity of the provisions but also the possibility of their simultaneous implementation in a specific situation.

For practical analysis, it is necessary to distinguish between differences in wording and differences in the actual provisions. Two provisions might use different terms for the same operation while establishing compatible rules. Conversely, nearly identical sentences may

sometimes specify incompatible timeframes or quantitative limits. The regulatory context also plays a role: the user role, the object's state, the type of service, the document version, and the existence of any specific exceptions. If such information is missing or inconsistent, a definitive conclusion regarding a conflict requires further justification. Therefore, it is advisable to compare related requirements while documenting the specific text segments upon which each determination is based.

The proposed approach treats automated analysis as a means of pre-selecting and explaining potentially conflicting pairs. A language model helps align the content of provisions even when identical actions are described using different wording, while explicit rules allow for the separate verification of numerical limits, temporal relationships, and negations. The results of these checks may diverge—for instance, due to context omitted from the source text or ambiguous interpretations of a condition. In such cases, both results should be retained alongside references to the original requirements, with the final decision left to an analyst. This workflow ensures that the conclusion remains reviewable following specification changes and necessitates a clear definition of exactly which pair of provisions is deemed contradictory. A contradiction is defined as a pair of requirements that cannot be simultaneously satisfied under the same conditions regarding the same role, action, and object. Differences in wording or the presence of an exception do not, in themselves, constitute proof of a contradiction. This distinction aligns with approaches to classifying contradictions and combining semantic analysis with logical verification [6], [7].

To establish a contradiction, one must examine not only the opposing nature of the provisions but also the overlap of the conditions under which they apply. A permission granted to a patient and a prohibition imposed on an administrator may be entirely consistent, even though both requirements concern the same operation. Similarly, differing restrictions for initial and follow-up appointments do not create a conflict if their scopes of application do not coincide. If the role, service type, or rule validity period is unspecified, concluding that there is an incompatibility would be premature; the missing conditions must first be clarified.

In the first step, each requirement is assigned an identifier, and the actor, action, object, modality (“may,” “must,” “prohibited”), condition, and numerical constraint are extracted. The requirement text and the reference to its source are preserved unchanged. In the second step, pairs relating to the same function or business rule are selected. This reduces the volume of comparisons and allows the necessary context—rather than the entire specification—to be passed to the language model. Research on natural language inference also supports the validity of comparing the content of requirement pairs [8].

During structuring, it is important to maintain the link between the concise record and the full text of the requirement. Otherwise, an exception contained in a subsequent sentence or a note regarding the rule's effective date might be lost. For each provision, it is advisable to record the document version, the section, and related term definitions. If the same role is referred to as “patient” in one part of the specification and “user” in another, they can be matched using the project glossary, but the decision regarding role equivalence must remain verifiable. Preliminary selection generates a list of candidates; it should not independently assign a “contradiction” status to a pair.

For a selected pair, the model must return one of three conclusions: “contradiction”, “consistent requirements”, or “insufficient context”. Along with the conclusion, it specifies the requirement identifiers, the exact text excerpts, the shared condition, and the reason for the discrepancy. Numerical limits, negations, and temporal relations are additionally verified using explicit rules. If the source text does not substantiate the model's explanation, the pair is referred for manual review without automatic correction. The application of logical rules in conjunction with language models is also described in [7].

The procedure is defined as model-agnostic: no specific large language model or version is fixed, as the experimental implementation has not yet been conducted. For each selected pair, the prompt includes the identifier, full verbatim text, source, revision, and effective date of each requirement, where known. Additional context is limited to term definitions and business rules that

directly define the scope of these provisions; missing information is explicitly noted. To reproduce future evaluations, the full prompt text, the name and version of the model used, and the generation parameters must be preserved.

The prompt template for the language model is as follows: "Requirement R1: ID={id1}; source={source1}; revision={rev1}; validity={valid1}; text={text1}. Requirement R2: ID={id2}; source={source2}; revision={rev2}; validity={valid2}; text={text2}. Context: term definitions={definitions}; business rules={rules}. Use only the provided data. Determine whether both requirements can apply simultaneously to the same actor, action, and object. Identify the shared situation and compare modalities, temporal and numerical limits, and exceptions. If information is insufficient, do not assume missing conditions. Return only a structured response in the specified format."

The expected response follows the JSON format: {"id_R1": "...", "id_R2": "...", "status": "...", "shared_condition": "...", "evidence_R1": "...", "evidence_R2": "...", "reason": "...", "missing_context": ["..."], "clarification_question": "...". The fields id_R1 and id_R2 reflect the identifiers of the submitted requirements; the status field takes only the values "contradiction", "consistent requirements", or "insufficient context". The fields evidence_R1 and evidence_R2 contain exact excerpts from the respective requirements, while missing_context lists the information needed to reach a conclusion. If the provided information is insufficient to establish an overlap in the requirements' scopes, the model returns the status "insufficient context" and formulates a specific clarification question in the clarification_question field. If the provided information indicates that the scopes do not overlap, the result is "consistent requirements", and the clarification_question field remains empty. Numerical limits, negations, and temporal relationships are additionally verified using explicit rules. A response lacking confirmation in the original text is not automatically accepted; it is stored in the verification card alongside the rule results and the analyst's decision.

The prompt to the model should be limited to the text of the two requirements, definitions of relevant terms, and the business rules governing their scope. The response should be required to consistently identify the shared scenario, highlight incompatible provisions, and specify any information missing for a definitive conclusion. A separate check should be performed to ensure the model correctly interprets terms such as "before", "after", and "no later than", as well as the inclusion of the boundary time point. It is advisable to document the analysis results in a verification record containing requirement identifiers, their versions, the model's explanation, the logical check outcome, and the analyst's decision. This enables a re-evaluation of the case following any specification changes. The sequence for identifying and verifying contradictions is illustrated in Fig. 1.

For instance, in a doctor appointment booking service, requirement R1 ("An authorized patient may cancel an appointment 12 hours prior to the visit") is incompatible with requirement R2 ("An authorized patient is prohibited from cancelling an appointment less than 24 hours prior to the visit") if both apply to the same type of appointment. This conclusion relies on a specific shared scenario: cancellation 12 hours before the visit. If R2 applies only to a specific service, the conflict is resolved once the scope is clarified. Consequently, the analyst verifies the validity of the requirements, any exceptions, and the agreed-upon business rule before documenting the decision.

To validate the procedure, a corpus of requirement pairs must be compiled with expert annotations—categorized as "contradictory", "consistent", or "insufficient data". This corpus should include explicit prohibitions, discrepancies in numerical limits, requirements involving exceptions, and semantically similar yet compatible statements. Comparing results against expert annotations allows for the calculation of conflict detection precision and recall, as well as the separate identification of false positives and the time required to review a single candidate pair. Until such an evaluation is conducted, claims regarding the procedure's effectiveness remain a research hypothesis. The need for rigorous empirical evaluation of requirements analysis tools is highlighted in previous studies [9], while the integration of language models with formal methods is discussed in [10].

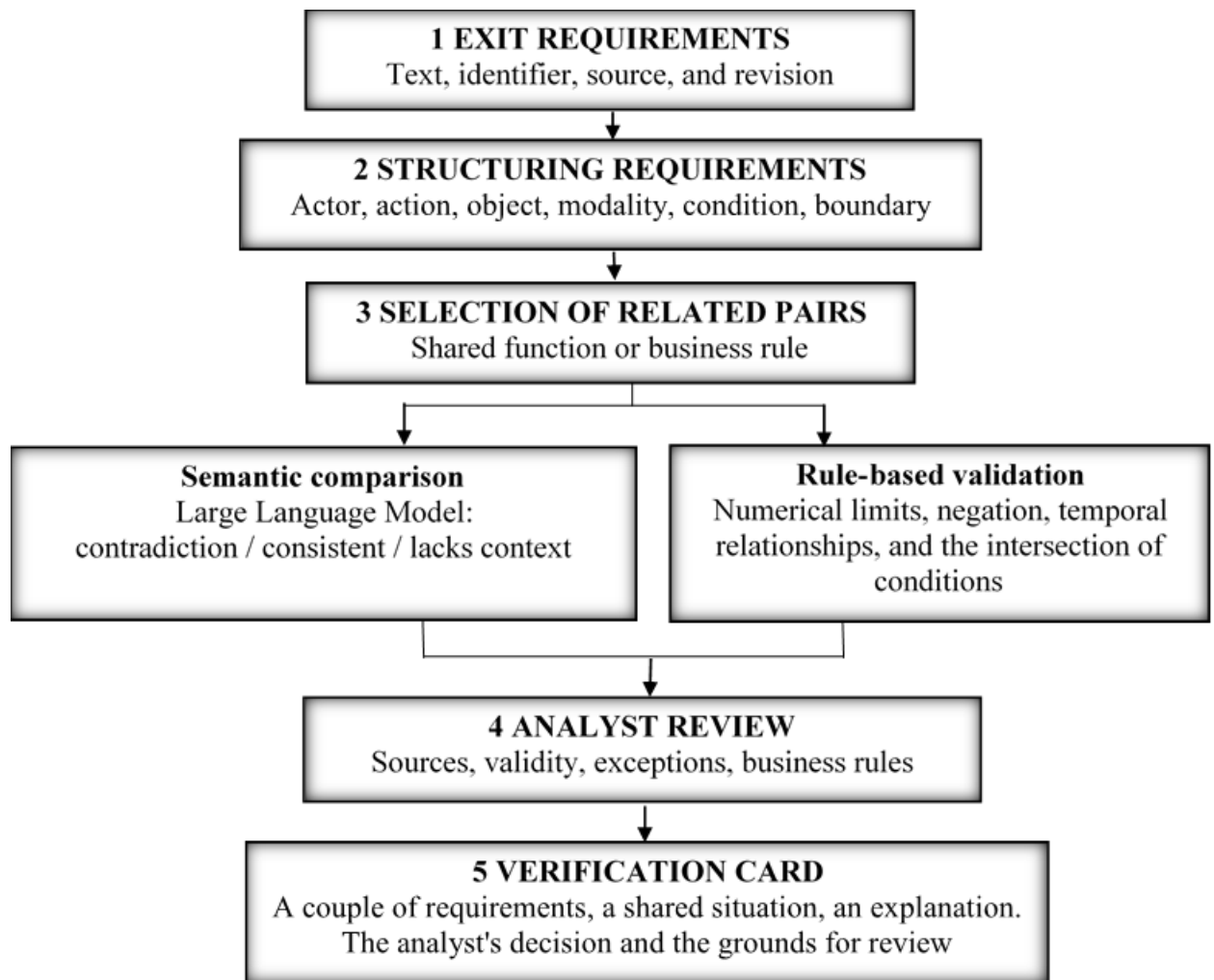


Fig. 1. Procedure for detecting and verifying contradictions in software system requirements

Source: compiled by the author

To create the test set, each pair is independently evaluated by at least two requirements analysis specialists; Discrepancies between their conclusions are resolved after reviewing the relevant business rules. It is important to separately evaluate explicit objections, time constraints, exceptions, and instances of insufficient context, as an aggregate metric might mask errors of a specific type. For comparison, it is useful to employ a simple search based on shared terms alongside an analysis performed solely by the language model, without additional rules. Such a comparison makes it possible to determine the contribution of each stage of the procedure while also revealing how many candidates will require manual review. The evaluation should be conducted using documents that were not utilized during the selection of examples or the formulation of model queries.

During experimental validation, it is advisable to distinguish between the quality of the initial pair selection and the quality of the decision made regarding a selected pair. If the first stage fails to capture related requirements, subsequent analysis cannot detect a contradiction, regardless of the quality of the language model's response. Therefore, for a subset of control documents, an expert-generated list of pairs sharing a common regulatory subject should be created and compared against the selection results. Separately, the classification into the three designated states should be evaluated using the same set of candidates. This distinction makes it possible to pinpoint exactly where omissions occur: during the search for related provisions or during the interpretation of their content. The time an analyst spends reviewing candidates should be tracked alongside the number of unjustified notifications they reject.

Evaluation should not be limited to whether the final classification matches the expert decision. For each reported contradiction, it is necessary to verify whether the cited excerpts actually appear in the source requirements, whether the shared context is correctly identified, and whether an exception in an adjacent sentence has been overlooked. A situation may arise where the model correctly selects the classification category by chance but justifies it using a rule that does not exist in the document. Such a result does not provide the analyst with a reliable basis for correcting the specification. Consequently, within the control dataset, it is useful to separately record the accuracy of the source reference, the description of conditions, and the reason for the incompatibility. Comparing these components with the expert rationale will reveal whether the conclusion can be verified against the document without re-performing the entire analysis. The impact of context can be investigated by comparing the conclusions drawn for the same pairs given varying amounts of input information. In the first scenario, the model receives only two statements; in the second, it receives term definitions and associated business rules; in the third, it also receives data regarding the revisions and validity of the relevant clauses. The composition of the test pairs, the evaluation criteria, and the presentation order must remain consistent across all scenarios. Particular attention should be paid to instances where, following the addition of information, the label changes from “contradiction” to “consistent requirements” or “insufficient context”. For each such change, the analyst must identify the specific provision that altered the interpretation of the pair. This approach makes it possible to assess the value of context without assuming that a larger volume of text automatically leads to a better decision.

Pairwise comparison has limitations that must be considered when interpreting results. Sometimes, the scope of two requirements is defined by a third provision—such as a general rule, a term definition, or an exception description. If such a provision is omitted from the check, the pair might appear contradictory simply because part of the specification was overlooked. The related provision should be taken into account as context clarifying the applicability of the specific pair and noted in the check card. If this context is insufficient, a conclusion regarding the pair is deferred, and the relevant provisions are referred to an analyst for separate review. Incorporating a third provision to interpret a pair does not imply that the procedure checks for contradictions within a group of requirements. Contradictions arising only from the simultaneous application of three or more requirements—and not detectable when checking any single pair—fall outside the scope of the proposed approach. Similarly, a wording change might resolve a previous conflict or create a new one for the same role and operation. This limitation does not negate the value of pairwise selection; rather, it defines when local analysis must be supplemented by a review of the broader context.

It is advisable to link the practical application of the procedure to the specification change approval process. Following the analyst's decision, the card records the pair's status, a reference to the agreed-upon business rule, and a list of items requiring clarification; the requirement text is modified only in accordance with established project procedures. If one of the requirements is revised, the associated cards must be reopened to verify the validity of the previous conclusion. It is important to retain both rejected messages and confirmed contradictions in order to track the causes of errors and avoid revisiting previously reviewed cases without new grounds. This approach defines the potential role of automated analysis in requirements management, though it requires validation using real-world documents and analyst workflows.

The primary scenarios encountered during requirements comparison, along with the information needed to verify them, are summarized in Table 1. The status of a pair depends on the overlap of the conditions under which the provisions apply; therefore, the cases presented here should not be interpreted as ready-made rules for the automatic correction of specifications.

In the first scenario presented in Table 1, two provisions prescribe opposing actions: one permits or mandates an action, while the other prohibits it. Simply finding the words “may” and “prohibited” in the text is insufficient to deem such a pair contradictory. It is necessary to determine whether the same user, object, and operation are involved, and whether both provisions apply under the same system state and during the same time period. For instance, a patient's right to cancel their own appointment and an administrator's prohibition against cancelling others' appointments do not

constitute a conflict. The conclusion must be based on a specific shared scenario where it is impossible to comply with both provisions simultaneously.

Table. Summary of the check for potential contradictions between requirements

Situation	What needs to be checked	Possible conclusion
Conflicting provisions	Identity of the role, action, and object; simultaneous validity of conditions.	Conflict due to overlapping scopes of application.
Time or numerical limits	Shared interval, units of measurement, and inclusion of the boundary value.	Conflicting or consistent requirements.
General rule and exception	Service type, object status, and the scope of the special rule.	Conclusion following clarification of the exception.
Different versions of clauses	Effective dates and whether the new clause replaces the previous one.	Insufficient context to compare the versions.
Indefinite conditions	Role definition, associated business rules, and missing conditions.	Insufficient context for clarification.

Source: compiled by the author

Time and numerical limits require separate verification, as their incompatibility is not always apparent from the wording. In the doctor's appointment example, permission to cancel 12 hours before the visit falls within a timeframe where another requirement prohibits cancellation less than 24 hours in advance. However, this conclusion holds true only if both requirements apply to the same type of appointment and define the appointment time in the same way. The analyst must clarify whether the boundary value is included in the permitted interval, the units used to specify the timeframe, and how terms such as “before”, “after”, and “no later than” are interpreted. This allows for distinguishing a genuine conflict of constraints from imprecise wording.

The relationship between a general rule and an exception presents a different scenario. A general provision might establish a timeframe or procedure for the majority of services, whereas a special provision defines a different procedure for a specific category of records. If the text specifies the scope of the exception, the two provisions can be reconciled: each applies to its own set of cases. However, when a special rule is formulated without reference to the general one, the system may erroneously flag the pair as contradictory. Therefore, it is necessary to verify the service type, user role, object status, and the condition under which the exception takes effect. In the absence of this information, it is more appropriate to request clarification than to arbitrarily prioritize one provision over the other.

Different versions of requirements can also create the appearance of a conflict. Following the approval of a change, the previous text sometimes remains in another section of the specification, an appendix, or a related document. Differing timeframes or user rights may then reflect the evolution of the rule rather than the simultaneous application of incompatible provisions. Verification requires the effective date, the source of each version, and information on whether the old provision has been replaced by the new one. If both versions are marked as active, the pair should be referred to an analyst along with the associated change decisions. When the change history is undetermined, a status of “insufficient context” more accurately reflects the state of knowledge than a categorical assertion of contradiction.

Finally, some requirements do not include all the conditions necessary for an unambiguous comparison. They may omit the user role, service type, record status, or the period during which the rule applies; A necessary definition may sometimes appear in a different section of the document. In such a case, the language model should not fill the gap with an assumption and present it as a specification provision. The verification record should specify exactly what information is missing, which related clauses have already been reviewed, and what question needs to be addressed to the stakeholder. Once a response is received, the pair can be re-evaluated and its status updated. This process preserves the traceability of the decision and reduces the risk of making unjustified changes to requirements.

Table demonstrates that the same apparent discrepancy may have different explanations depending on the validity of the provisions and their scope of application. Specifically, differing numerical limits do not prove a contradiction unless they share a common interval, and a general rule can coexist with a specific exception. Therefore, for each pair, the conditions verified and the basis for the conclusion must be documented; when this information is insufficient, the correct course of action is to request clarification rather than automatically assigning a "contradiction" status.

Examining these scenarios helps clarify the distinction between contradiction, consistency, and a lack of information. The decisive question is whether a single situation can exist where both active requirements apply to the same actor and object yet prescribe incompatible actions or constraints. If the scopes of application do not overlap, a difference in wording does not constitute a contradiction. Conversely, if a role, exception, or validity period is not specified, there are no grounds to classify the pair as either consistent or contradictory. This distinction is necessary for both manual analysis and the creation of a test set: an expert must explain not only the classification label but also the conditions that justify it.

In the proposed procedure, each stage contributes a specific type of information to the final conclusion. Structuring preserves the link between requirement elements and the source text; pair selection determines which provisions should be compared; the language model offers a semantic explanation; and explicit rules verify boundaries, negations, and temporal relationships. The analyst compares these results against existing business rules and documents the final decision. Even when the model's output aligns with the rule-based check, it remains necessary to review the source excerpts, as both mechanisms may be operating with incomplete context. Retaining the verification record makes it possible to identify exactly where a discrepancy arose and what requires clarification following a specification change. Thus, the proposed result should be viewed as a description of a reproducible verification process rather than as evidence of the accuracy of automated detection already achieved. Its practical utility will depend on the ability to identify related pairs without excessive omissions, provide the correct textual basis for the conclusion, and avoid overwhelming the analyst with false alerts. This requires evaluation against independently annotated requirements, comparison with simpler search methods, and a separate analysis of cases where context alters the status of a pair. Only after such a comparison can one determine which stages of the procedure genuinely aid in detecting contradictions and which require refinement before being applied to actual requirements work.

Conclusions and prospects for further research. The study clarifies the nature of contradictions between textual requirements for a software system: the decisive factor is not the external opposition of statements, but the impossibility of simultaneously fulfilling their mandates in the same situation. Reaching this conclusion requires establishing a commonality of actor, action, and object, as well as determining whether the conditions under which the respective rules apply overlap. Different document versions, specific exceptions, or requirements for different user categories may explain a discrepancy without an actual conflict existing. Thus, identifying a contradiction requires comparing the content of provisions with the context of their application; this conclusion aligns with research on the classification and formal verification of contradictions [6], [7].

An analysis sequence is proposed that encompasses structuring the initial requirements, selecting related pairs, performing semantic comparison using a large language model, verifying explicit constraints via rules, and having an analyst review the results. A key aspect of the procedure is preserving the full text, identifier, source, and version of each requirement. This allows explanations of potential conflicts to be linked to specific specification fragments and enables any disagreement between the model's conclusion and the rule-based verification result to be referred for expert review. While combining semantic analysis with logical verification is supported by existing literature [7, 10], the effectiveness of the specific procedure proposed here remains to be experimentally established. The example of a doctor's appointment cancellation discussed here demonstrates why comparing individual words or numbers is insufficient for evaluating time-based

constraints. A rule allowing cancellation 12 hours prior to an appointment and another prohibiting it less than 24 hours in advance create a contradiction only if both rules apply to the same type of appointment and remain in effect. If one requirement governs a specialized service, the conclusion changes once the scope is clarified. This leads to a practical rule: the system should flag a suspicious pair, present the shared context, and indicate what information is missing; the final decision and the agreed-upon correction to the specification rest with the analyst and the stakeholders.

Further research involves creating a corpus of Ukrainian-language requirement pairs, independently annotated by experts across three categories: “contradiction”, “consistent requirements”, and “insufficient context”. The corpus should include explicit prohibitions, numerical and temporal constraints, exceptions, different wording variations, and semantically similar yet compatible provisions. Disagreements between experts should be resolved through a review of source materials and business rules, with the rationale for the final decision documented. The accuracy and completeness of contradiction detection, the false positive rate, and the time required for manual verification of a single pair will be evaluated using documents not utilized for example selection or query tuning. Such an evaluation is necessary to distinguish the quality of the proposed method from the impression created by specific illustrative examples [9].

A distinct avenue for future work involves comparing the full procedure against alternative approaches: searching for shared terms, using only a language model for analysis, and omitting the transfer of business rule context. This will help determine the contribution of each stage—particularly explicit condition checking—to identifying actual conflicts and reducing the number of false-positive alerts. Following this validation, it is advisable to examine how requirement versions are handled: specifically, whether the rationale for a conclusion remains valid after a specification item changes, and whether linked pairs can be re-verified to promptly notify the analyst of any change in conflict status. Adapting the procedure to other domains must be accompanied by a review of that domain's specific terms and rules, as results obtained for one set of requirements cannot be directly generalized to all software systems. Another future direction is extending the analysis to groups of three or more requirements—specifically cases where individual pairs appear consistent, yet the collective set of provisions cannot be satisfied simultaneously; this requires defining a method for selecting such groups and establishing rules for verifying their joint conditions.

Overall, the proposed procedure allows a suspected conflict to be formulated as a verifiable statement based on the document: identifying which two provisions are being compared, the shared conditions under which they cannot both be satisfied, and the specific specification fragments that substantiate this conclusion. If the necessary condition is absent from the text, the outcome is a specific question addressed to those responsible for the requirements, rather than an assumption being recorded in the document as a fact. This format can be useful when reviewing specifications and approving changes, as it keeps the rationale behind decisions available for future review. Subsequent evaluation should determine whether the resulting traceability and conflict detection capabilities justify the costs of setting up the procedure and the analyst's effort.

Contradictions arising from a combination of three or more requirements—even when pairwise checks reveal no incompatibilities—warrant separate study. To identify them, it is advisable to first establish connections between provisions based on shared actors, actions, objects, and conditions of application, and then examine the groups of requirements linked in this way. Identifying the minimal set of provisions that cannot be satisfied simultaneously is particularly important; this enables the analyst to pinpoint the cause of the contradiction without reviewing the entire specification. However, increasing the number of requirements in a group complicates candidate selection and raises the risk of erroneous conclusions due to overlooked exceptions or outdated versions. Consequently, expanding the procedure requires specific rules for presenting the shared context, maintaining references to each provision, and ensuring analyst verification of the results. An empirical comparison of pairwise versus group analysis will help determine which

contradictions remain undetected under the current approach and whether the effort to find them justifies the additional verification costs.

Conflict of interest: The author declares no conflict of interest—whether financial, personal, authorship-related, or of any other nature—that could influence the research or the results published in this article

Funding: The study was conducted without financial support

Data availability: All data are available in digital or graphical form in the main text of the manuscript

Use of artificial intelligence (AI) tools: The author confirms that she did not use AI tools to write this work. In this paper, large language models are considered a component of the proposed requirements analysis procedure

REFERENCES

1. “ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering”. Geneva: ISO. 2018. – Available from: <https://www.iso.org/standard/72089.html>. – [Accessed: 28.06.2026].
2. Gervasi, V. & Zowghi, D. “Reasoning about inconsistencies in natural language requirements”. *ACM Transactions on Software Engineering and Methodology*. 2005; 14 (3): 277–330. DOI: <https://doi.org/10.1145/1072997.1072999>.
3. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M., Chioasca, E.-V. & Batista-Navarro, R. T. “Natural language processing for requirements engineering: A systematic mapping study”. *ACM Computing Surveys*. 2021; 54 (3): 55. DOI: <https://doi.org/10.1145/3444689>.
4. Fantechi, A., Gnesi, S., Passaro, L. & Semini, L. “Inconsistency detection in natural language requirements using ChatGPT: A preliminary evaluation”. *2023 IEEE 31st International Requirements Engineering Conference*. 2023. DOI: <https://doi.org/10.1109/RE57278.2023.00045>.
5. Mahbub, T., Dghaym, D., Shankarnarayanan, A., Syed, T., Shapsough, S. & Zualkernan, I. “Can GPT-4 Aid in detecting ambiguities, inconsistencies, and incompleteness in requirements analysis? A comprehensive case study”. *IEEE Access*. 2024; 12: 171972–171992. DOI: <https://doi.org/10.1109/ACCESS.2024.3464242>.
6. Gärtner, A. E., Fay, T.-A. & Göhlich, D. “Fundamental research on detecting contradictions in requirements: Taxonomy and semi-automated approach”. *Applied Sciences*. 2022; 12 (15): 7628. DOI: <https://doi.org/10.3390/app12157628>.
7. Gärtner, A. E. & Göhlich, D. “Automated requirement contradiction detection through formal logic and LLMs”. *Automated Software Engineering*. 2024; 31: 49. DOI: <https://doi.org/10.1007/s10515-024-00452-x>.
8. Fazelnia, M., Koscinski, V., Herzog, S. & Mirakhorli, M. “Lessons from the use of natural language inference (NLI) in requirements engineering tasks”. *2024 IEEE 32nd International Requirements Engineering Conference*. 2024. DOI: <https://doi.org/10.1109/RE59067.2024.00020>.
9. Montgomery, L., Fucci, D., Bouraffa, A., Scholz, L. & Maalej, W. “Empirical research on requirements quality: a systematic mapping study”. *Requirements Engineering*. 2022; 27: 183–209. DOI: <https://doi.org/10.1007/s00766-021-00367-z>.
10. Ferrari, A. & Spoletini, P. “Formal requirements engineering and large language models: A two-way roadmap”. *Information and Software Technology*. 2025; 181: 107697. DOI: <https://doi.org/10.1016/j.infsof.2025.107697>.

Received 28.08.2026

Received after revision 25.09.2026

Accepted 09.10.2026

DOI: <https://doi.org/10.15276/ict.03.2026.52>

УДК 004.41:004.8

Виявлення суперечностей у вимогах до програмних систем засобами штучного інтелекту

Попова Марія Олександрівна

ORCID: <https://orcid.org/0000-0001-6836-3690>; popova.m.o@op.edu.ua

Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

АНОТАЦІЯ

У роботі розглянуто задачу виявлення змістових суперечностей у текстових вимогах до програмних систем. Такі суперечності виникають, коли різні положення специфікації встановлюють несумісні правила для однієї дії за однакових умов. Їх виявлення ускладнюється тим, що вимоги можуть бути сформульовані різними словами, містити винятки або посилалися на неявні бізнес-правила. Мета дослідження полягає у формуванні процедури аналізу, яка допомагає аналітику знаходити й перевіряти потенційно суперечливі пари вимог. Запропоновано виокремлювати в кожній вимозі актора, дію, об'єкт, модальність, умови застосування та числові обмеження, після чого відбирати пари, пов'язані спільною функцією. Під час відбору враховують також належність вимог до одного процесу та чинність їхніх редакцій, оскільки схожі за змістом положення можуть описувати різні стани системи. Для їх змістового зіставлення передбачено використання великої мовної моделі, а для перевірки числових меж і заперечень – явних правил. Особливу увагу приділено випадкам, коли несумісність дозволу й заборони виявляється лише після зіставлення часових умов або граничних значень. Висновок має супроводжуватися посиланнями на вихідні фрагменти, зазначенням спільної умови та поясненням причини можливого конфлікту. Такий спосіб подання результату забезпечує простежуваність аналізу і дає змогу перевірити, чи не було втрачено важливого контексту під час зіставлення. На прикладі вимог до скасування запису на прийом показано, що подібність формулювань не доводить суперечності без перевірки сфери їх дії. Якщо відомостей про сферу дії або винятки бракує, пару доцільно позначити як таку, що потребує уточнення, не зараховуючи її до підтверджених суперечностей. Остаточне рішення щодо узгодженості вимог належить аналітику. Після розгляду підтвердженого конфлікту аналітик має зафіксувати погоджене правило та визначити, які саме положення специфікації потребують уточнення. Для подальшої емпіричної перевірки процедури запропоновано використати експертно розмічений корпус пар вимог і оцінити точність, повноту, частку хибних спрацювань та час перегляду одного кандидата. Кількісних результатів застосування процедури на цьому етапі не наведено.

Ключові слова: інженерія вимог; суперечності у вимогах; програмні системи; великі мовні моделі; семантичний аналіз; специфікація вимог; логічна перевірка; експертний аналіз

ABOUT THE AUTHOR



Mariia O. Popova - PhD, Associate Professor, Department of Software Engineering, Odesa Polytechnic National University, 1, Shevchenko Av. Odesa, 65044, Ukraine

ORCID: <https://orcid.org/0000-0001-6836-3690>; popova.m.o@op.edu.ua

Research field: software engineering

Попова Марія Олександрівна - кандидат екон. наук, доцент, доцент каф. Інженерії програмного забезпечення. Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

Галузь досліджень: програмна інженерія