Journal of Software Engineering Research and Development, 2026, 14:5, doi: 10.5753/jserd.2026.5919

Extending the Comparative Study of Anomaly Detection Tools in Software Requirements with ChatGPT

Fábio Rodrigues Pereira [**Federal University of Lavras** | ***fabiopereiragt@gmail.com***]
Heitor Augustus Xavier Costa [**Federal University of Lavras** | ***heitor@ufla.br***]
Paulo Afonso Parreira Júnior [**Federal University of Lavras** | ***pauloa.junior@ufla.br***]

Abstract. A software requirement indicates a capability or characteristic that a software system must possess to provide value to its stakeholders. It is essential to ensure that the description of the requirements is unambiguous to allow for proper understanding and facilitate its evolution. However, since most software requirements are described in natural language, they may contain subjectivity and inconsistencies in their descriptions, which are conventionally referred to as "Software Requirements Anomalies". Several studies propose tools to aid in the detection of requirements anomalies. However, it can be observed that few of these studies evaluate the effectiveness (recall and precision) of the proposed tools. Therefore, this work presents a comparative study of three anomaly detection tools (`RETA`, `Tactile Check`, and `Tiger Pro`), as well as the ChatGPT model, analyzed based on requirements documents from different domains containing over 85 anomalies. The results show that the `Tactile Check` tool produced the best performance. Although ChatGPT offers advantages in terms of information visualization and flexibility of interaction, its performance was not satisfactory compared to tools specifically designed for anomaly detection in software requirements. All analyzed tools, including ChatGPT, demonstrated unsatisfactory levels of recall and precision, averaging below 66% and 57%, respectively. These results highlight the need for further contributions in this research area.

Keywords: Requirements Engineering, Requirements Anomalies, Generative AI, ChatGPT

1 Introduction

Software is widely integrated into individuals' professional and personal environments (Fabbri *et al*. [2014]). Given its significance in the modern world, ongoing concerns exist about the quality of software developed and delivered to stakeholders (Machado [2018]). A software requirement specifies a capability, attribute, quality, or characteristic necessary for it to be valuable to stakeholders. Requirements Engineering encompasses the discovery, analysis, documentation, and verification of requirements for software development and maintenance (Sommerville [2011]). It includes a series of activities that can lead to a textual document outlining the software requirements, known as "Requirements Document" (Chitchyan *et al*. [2006]).

According to Valente [2020], Requirements Engineering can directly contribute to software quality by aligning the Requirements Document with stakeholders' expectations. However, producing a high-quality Requirements Document is not a straightforward task. Requirement engineers typically describe software requirements in natural language, which can lead to subjectivity and inconsistency, compromising understanding and interpretation (Nascimento *et al*. [2018]). For instance, it may result in incorrect or defective implementations, leading to rework, increased development costs, and team demotivation (Femmer *et al*. [2017]).

The concept of "Software Requirements Anomalies" (also known as Software Requirements Smells) emerges in the search for quality in Requirements Documents. In this paper, we use the term *requirements anomalies* to refer to specific types of defects in requirement descriptions, following the taxonomy initially proposed by Davis (1993). Although the terminology *defect* is more broadly used in the literature, we adopt *anomaly* to ensure consistency with our previous study Pereira *et al*. [2024], which this work extends. Requirements anomalies[1] are indicators that there is "something wrong" with the description of a requirement. Certain anomalies involve the use of comparative terms such as "better performance" or "lower response time" when specifying software requirements (Femmer *et al*. [2017]). We classify these terms as anomalies because they introduce subjectivity into the Requirements Documents, making them harder to understand and interpret.

Several studies in the literature have proposed tools for detecting requirements anomalies. According to (Nascimento *et al*. [2018])'s Systematic Literature Mapping, 41 studies on this topic were published between 2013 (when the "Software Requirements Smells" term was first introduced) and 2018, proposing 22 tools for anomaly detection. However, there is a lack of studies evaluating the effectiveness[2] of these tools by comparing them with one another and identifying their strengths and weaknesses (Nascimento *et al*. [2018]). Research in this area is relevant because it supports the development of more effective tools, enhances the performance of existing ones, and helps researchers and industry professionals make informed decisions about which tools best meet their needs.

This article conducted a comparative analysis of the tools for detecting requirements anomalies effectiveness while identifying their strengths and weaknesses. To achieve this objective, we employed the following research method: (i) updating Nascimento *et al*. [2018]'s Systematic Literature

[1] For simplicity, the "Software Requirements Anomalies" term will be referred to as "Requirements Anomalies".

[2] In this paper, the "effectiveness" term corresponds to tool recall and precision for detecting requirements anomalies.

Mapping; (ii) selecting the tools for comparative analysis based on predefined criteria; (iii) choosing a set of Requirements Documents with pre-cataloged anomalies; (iv) analyzing the effectiveness of the selected tools using recall and precision metrics; and (v) detailing the strengths and weaknesses of each tool based on the analysis results.

We selected three tools for the comparative analysis: (i) `RETA` from Arora *et al*. [2015], (ii) `Tactile Check` from Wilmink and Bockisch [2017], and (iii) `Tiger Pro` from Arendse and Lucassen [2016]). We applied these tools to three Requirements Documents from different domains, which contained over 85 anomalies of various types. Based on the combined analysis of recall and precision, the `Tactile Check` tool achieved the best result, standing out in evaluations compared to the other two tools. However, all three tools exhibited unsatisfactory levels of recall and precision, with averages below 66% and 57%, respectively. These findings highlight existing gaps for new contributions in this research area.

In addition to the tools originally designed for anomaly detection, we also evaluated the ChatGPT model—an advanced generative AI—by comparing its performance to that of the best-performing tool, `Tactile Check`. Although ChatGPT showed favorable results in reducing false positives for some anomaly categories, its overall performance was unsatisfactory in terms of recall and F-measure. As a general-purpose model, ChatGPT lacks the domain-specific adjustments needed to achieve higher effectiveness in this context.

It is worth mentioning that this article is an extended version of the work (Pereira *et al*. [2024]) published in the proceedings of the XXIII Brazilian Symposium on Software Quality (SBQS 2024). In this extended version, we introduce a third tool—ChatGPT—to explore the feasibility of using large language models (LLMs) for the same task. We redesigned and adapted the dataset to suit the prompt-based interaction with ChatGPT and conducted a new set of experiments to evaluate its performance. Furthermore, we revised and expanded key sections of the paper to reflect the broader scope, and included a more in-depth review of related work involving generative AI in Requirements Engineering. This extended study provides new insights into the capabilities and limitations of LLMs when applied to quality assessment of natural language requirements.

We structured the article as follows. Section 2 offers a brief theoretical background on requirements anomalies, their relation to software quality, and pertinent work. Section 3 details the selection of tools and artifacts used during the comparative analysis. We also discussed the procedures performed before the comparative analysis, including updating the Systematic Literature Mapping, selecting analysis approaches, identifying requirement sets to detect anomalies, and conducting the comparative analysis. Section 4 presents the findings of the anomaly detection process. Section 5 considers the final considerations, limitations, and suggestions for future work.

2 Background

2.1 Software Requirements Anomalies

According to Sommerville [2011], software requirements express the attributes and functionalities necessary for software to perform tasks that meet the stakeholders' goals. The most common approach to writing software requirements is to use Natural Language (Arendse and Lucassen [2016]; Zhao *et al*. [2020]) because it facilitates communication of the requirements with stakeholders and allows individuals not involved in software development to easily understand them (Kroth [2012]). However, ambiguity and inconsistency often arise when using Natural Language, leading to multiple interpretations and omissions of essential characteristics that the software should possess (Kiyavitskaya *et al*. [2008]; Kamsties *et al*. [2001]). When these negative aspects emerge in the requirements description, people recognize them as Software Requirements Anomalies (Femmer [2013]). These anomalies arise during the Requirements Documents creation and maintenance and share the following characteristics (Nascimento *et al*. [2018]; Femmer *et al*. [2017]):

- **They indicate violations of the Requirements Documents quality**: People understand the Requirements Documents quality in terms of its (potential) effects on subsequent activities in the software life-cycle that depend on the Requirements Documents;
- **They do not necessarily lead to defects and should, therefore, be assessed based on context**: Whether discovering an anomaly constitutes a problem should be decided individually, based on the context;
- **They have a specific location, such as a word or a sentence**: Requirements anomalies indicate a particular location where inspection should occur;
- **They have a concrete detection mechanism**: Strategies for detecting them are feasible due to their concrete nature. However, the accuracy of these strategies may vary depending on the context and the type of anomaly detected.

Recent literature on this topic describes a catalog of 11 requirements anomalies (Nascimento *et al*. [2021]). Table 1 presents the anomalies, their names, brief descriptions, and practical examples.

2.2 Related Work

Several studies have proposed new approaches to ensure the quality of Requirements Documents concerning software requirements anomalies.

Kamsties *et al*. [2001] focus on the identification of ambiguity in natural language software requirements. Their work categorizes common sources of ambiguity and proposes linguistic indicators—referred to as "ambiguity classes"—that can serve as heuristics to detect potentially ambiguous expressions in requirements documents. Unlike our study, which evaluates the ability of automated tools to detect a broader set of ten requirement anomalies, their approach is based on a theoretical and linguistic analysis supported by examples, rather than a tool-based empirical evaluation.

Table 1. Description of requirements anomalies cataloged by Nascimento *et al*. [2021]

ID	Anomalies	Description
AN1	Ambiguous Adverbs and Adjectives	Refers to unspecific adverbs and adjectives by nature, such as "almost always", "very little", "significative", and "minimal". **Example**: If the quality (...) is too low, a failure should be recorded in the error memory.
AN2	Loopholes	Refers to words that may allow stakeholders to ignore requirements, such as "if possible", "as appropriate", and "if applicable". **Example**: To the extent possible, inputs should be verified.
AN3	Non-verifiable Terms	Refers to terms that are difficult to verify, as they offer several possibilities, for example, to developers. **Example**: The software should offer the user all possible forms of payment.
AN4	Superlative	Refers to adverbs and adjectives that express the relationship of software to all other systems, such as "best performance" and "shortest response time". **Example**: The software should provide the signal to the customer in the best resolution.
AN5	Comparative	Refers to adverbs and adjectives that express the relationship of software with another software or previous situation, such as "better than" e "greater than". **Example**: The new user interface should contain fewer usability problems than the previous one.
AN6	Negative Statements	Refers to functionality statements the software should not provide, which may lead to a lack of explanation about the actual behavior of software in such cases. **Example**: The software should not disconnect users due to timeouts.
AN7	Vague Pronouns	Refers to unclear relationships of a pronoun. **Example**: The software should implement services for applications, which should communicate with the controller applications deployed on other controllers.
AN8	Incomplete References	Refers to references the reader cannot find in the Requirements Document. **Example**: The architecture must follow the guidelines defined in Section 5.4 (Note: Section 5.4 does not exist in the document).
AN9	Subjective Language	Refers to words whose semantics are not objective, such as "friendly", "easy to use", and "economical". **Example**: The new software user interface should be friendly.
AN10	Passive Voice	Refers to situations in which it is unclear which actor performs a given action in the software. **Example**: The software should display a success message as soon as the record is saved.
AN11	Duplicate Functionality	Refers to the interaction repetition between software and actors across multiple software requirements. **Example**: The software checks whether the user is logged in before proceeding.

Nascimento *et al*. [2018] conducted a Systematic Literature Mapping to identify studies report approaches and techniques for detecting software requirement anomalies. They considered publications from January 2013 to March 2018. This study shares similarities with Systematic Literature Mapping in its focus on reporting software quality aspects, particularly emphasizing requirements anomalies. However, while Systematic Literature Mapping gathered and categorized approaches and tools available in the literature for detecting requirements anomalies, this study aimed to compare computational tools for anomaly detection against each other through a controlled experiment.

Femmer *et al*. [2017] provided an in-depth discussion and a precise definition of software requirements anomalies. Additionally, they proposed a computational tool for automatically detecting related anomalies. Their study aimed to (i) define the concept of "Requirements Smells" and explain its derivation from the ISO 29148 standard, (ii) present the `Smella` computational tool, and (iii) report on the empirical study conducted to evaluate the proposed tool. This study builds upon the Femmer *et al*. [2017]'s study as a theoretical foundation for requirements anomalies. However, it differed from their study and others in the same research line in that, instead of proposing a new approach, it evaluated existing ones to determine their effectiveness.

Arendse and Lucassen [2016] compared the performance of three tools for detecting defects and deviations in requirements documents written in Natural Language. They focused on two specific types of anomalies: ambiguity and atomicity[3]. Based on the results of their comparative analysis, they developed a tool that leverages the strengths identified in the evaluated tools. In total, they identified 16 areas for improvement. Their research served as a reference to guide the comparative analysis methodology employed in this study. As a distinguishing factor, this study verified additional types of anomalies (beyond ambiguity and atomicity) and evaluated different tools. Besides, it utilized a set of Requirements Documents obtained from an open and publicly available repository, ensuring the replicability of this study.

In recent years, the integration of Large Language Models (LLMs), such as ChatGPT, into Requirements Engineering (RE) has gained significant attention. Several studies have investigated how generative models can support key RE activities, including elicitation, analysis, and validation of requirements. For instance, Arora *et al*. [2024] present a comprehensive analysis of how LLMs can be applied across all RE phases, including a SWOT analysis that highlights both opportunities (such as automation, compliance checking, and support for traceability) and risks (such as overreliance, hallucinations, and lack of domain-specific understanding). Complementarily, Vogelsang [2024] explore the concept of *prompts* as artifacts within the RE lifecycle, arguing that prompt engineering itself can benefit from RE principles, as prompts can be interpreted as high-level representations of requirements.

[3]One ambiguous requirement can have multiple interpretations; one non-atomic requirement has more than one function (Arendse and Lucassen [2016]).

A recurring finding in recent literature is that ChatGPT, when properly used, can produce requirements that are highly *abstract*, *consistent*, *correct*, and *understandable*, as demonstrated by Ronanki *et al*. [2024]. In a comparative study involving human experts, requirements generated by ChatGPT frequently achieved similar or even higher scores in various quality attributes, although it performed worse in terms of *feasibility* and *unambiguity*. These results reinforce ChatGPT's potential as a valuable assistant in the early phases of RE, particularly for tasks such as requirement formulation, quality assessment, and ambiguity detection.

The present study builds upon our earlier work Pereira *et al*. [2024], which presented a comparative analysis of three traditional tools for anomaly detection in requirements, without including any form of generative AI or language model. Based on this, the present study explores the use of ChatGPT not to propose a new RE tool, but to evaluate its effectiveness in detecting anomalies in software requirements — a task traditionally performed by static analysis tools or rule-based engines. To this end, we designed a comparative experiment in which ChatGPT was tested against the best-performing anomaly detection tool identified in prior evaluations. The same anomaly categories covered by that reference tool were used as the evaluation basis for ChatGPT's output.

To the best of our knowledge, no previous studies have conducted a direct comparison between the performance of ChatGPT and tools specialized in anomaly detection in requirements documents. Most current research focuses on ChatGPT's ability to generate or classify requirements, with limited emphasis on defect detection. Therefore, this study fills an important gap by providing a systematic assessment of ChatGPT's capabilities in this scenario, using predefined taxonomies and structured analysis.

3 Pre-Comparative Analysis Procedure

The pre-comparative analysis procedures involve updating the Systematic Literature Mapping (Section 3.1), choosing analytical approaches (Section 3.2), selecting sets of requirements, identifying anomalies (Section 3.3), and conducting the comparative analysis (Section 3.4).

3.1 Updating Systematic Literature Mapping

When initiating the search for tools for comparative analysis, Nascimento *et al*. [2018] gained prominence for providing a comprehensive systematic literature mapping on requirements anomaly detection. Their study included 41 scientific publications and proposed 22 tools for anomaly detection. However, updating this systematic literature mapping became necessary since researchers published these tools between 2013 and 2018. The Garner *et al*. [2016]'s updating framework[4] of Systematic Literature Reviews guided this decision. This framework consists of three key steps:

- **Step 1**: Assessing the relevance of the existing Systematic Literature Review by determining whether its topic remains significant for research and practice;
- **Step 2**: Identifying whether existing new published methods or studies since the original Systematic Literature Review;
- **Step 3**: Evaluating whether these new methods or studies could influence the conclusions of the original Systematic Literature Review.

After applying these three steps, **it was evident that updating the systematic literature mapping was necessary**, as the topic remains relevant and new approaches and tools may have emerged, introducing novel findings and valuable insights. The protocol parameters used for this update—such as research questions, digital libraries, search strings, and inclusion/exclusion criteria—remained the same as those established by Nascimento *et al*. [2018][5]. The only modification was the publication period, which we extended to include studies from January 2013 to May 2023. As a result of this update, we added eight studies. Only three proposed computational tools for detecting requirements anomalies correspond to papers ID 21, 22, and 23 (Table 2).

3.2 Selecting Analysis Approaches

Based on the studies presented in Table 2, we selected the tools for comparative analysis according to the Arendse and Lucassen [2016]'s criteria:

- **Availability**. The tool had to be freely available for download or offer a free license for testing if it was a commercial product;
- **Automation**. We considered only fully automated tools to eliminate any influence of the researchers' prior knowledge in handling them;
- **Documentation**. The article (or webpage) describing the computational tool must contain clear and understandable information regarding its installation and use. The tool must provide documentation on its functions and operations to avoid penalization during the comparative analysis due to improper handling of its resources;
- **Requirements description format**. Since Natural Language is the primary format for requirements description, the tool had to support this format.

From the 23 identified articles, 22 distinct tools were extracted and analyzed based on these criteria. Table 3 presents the results of this analysis. The first column (ID) represents the tool identification number, the second column (Tools) lists its name, and the third column (ID Paper) indicates the papers mention it. The fourth to seventh columns correspond to the evaluation criteria (Availability - C1, Automation - C2, Documentation - C3, and Requirements Description Format - C4). The last column indicates whether we accepted the tool for evaluation based on these criteria. As shown in the table, only three tools met the criteria: `RETA`, `Tactile Check`,

[4] Although this framework was developed focusing on Systematic Literature Reviews, according to the authors, it can be used for Systematic Literature Mapping.

[5] The search string adopted the term *anomaly* to maintain alignment with the previous study. We acknowledge, however, that including synonyms such as *defect* or *requirements defect* might lead to a broader retrieval of related works.

Table 2. Studies that use/propose tools for detecting requirements anomalies

ID	Title	References
1	*Automated Checking of Conformance to Requirements Templates using Natural Language Processing*	Arora *et al*. [2015]
2	*Flexible Ambiguity Resolution and Incompleteness Detection in Requirements Descriptions via an Indicator-Based Configuration of Text Analysis Pipelines*	Bäumer and Geierhos [2018]
3	*Rapid Requirements Checks with Requirements Smells: Two Case Studies*	Femmer *et al*. [2014]
4	*Rapid quality assurance with Requirements Smells*	Femmer *et al*. [2017]
5	*Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain*	Rosadini *et al*. [2017]
6	*On the Ability of Lightweight Checks to detect Ambiguity in Requirements Documentation*	Wilmink and Bockisch [2017]
7	*Toward tool mashups comparing and combining NLP RE Tools*	Arendse and Lucassen [2016]
8	*Initial Investigation on the influence of requirement smells on test case design*	Beer *et al*. [2017]
9	*Automated Extraction and Clustering of Requirements Glossary Terms*	Arora *et al*. [2017]
10	*Using Argumentation to Explain Ambiguity in Requirements Elicitation Interviews*	Elrakaiby *et al*. [2017]
11	*Automatic Requirements Reviews - Potentials, Limitations and Practical Tool Support*	Femmer [2017]
12	*Quality Assurance of Requirements Artifacts in Practice: A Case Study and a Process Proposal*	Femmer *et al*. [2016]
13	*Detecting Domain-Specific Ambiguities: An NLP Approach Based on Wikipedia Crawling and Word Embeddings*	Ferrari *et al*. [2017a]
14	*Interview Review: Detecting Latent Ambiguities to Improve the Requirements Elicitation Process*	Ferrari *et al*. [2017c]
15	*Natural Language Requirements Processing: From Research to Practice*	Ferrari [2018]
16	*Detecting requirements defects with NLP patterns: an industrial experience in the railway domain*	Ferrari *et al*. [2018]
17	*PURE: A Dataset of Public Requirements Documents*	Ferrari *et al*. [2017b]
18	*Using human error information for error prevention*	Hu *et al*. [2018]
19	*Improving agile requirements: the Quality User Story framework and tool*	Lucassen *et al*. [2016]
20	*Detecting requirements defects with NLP patterns*	Ferrari *et al*. [2018]
21	*Quality Requirements and the Requirements Quality*	Calazans *et al*. [2019]
22	*Ambiguous requirements A semi-automated approach to identify and clarify ambiguity in large-scale projects*	Asadabadi *et al*. [2020]
23	*Hidden fuzzy information Requirement specification and measurement of project provider performance using the best worst method*	Asadabadi *et al*. [2019]

and `Tiger Pro`. The selection process was carried out by the first author and subsequently reviewed by the last two authors of this study.

We excluded most tools due to failing the "Availability (C1)" and "Documentation (C3)" criteria. For instance, the `Scout` tool, developed by the Qualicen company, does not provide academic licenses for its products. While it provided rule configurations and explanations from its developers via email, the `GATE` tool lacked publicly available documentation on how to set up and execute these rules, thereby failing the "Documentation" criterion. Similarly, the `Teamscale` tool, which exclusively handles requirements linked to source code, did not meet the "Requirements Description Format (C4)" criterion.

3.3 Selecting Requirements Documents and Identifying Anomalies

The Requirements Document (RD) plays a crucial role in analyzing the effectiveness of tools for detecting software requirements anomalies. To compare the tools in a scenario closer to the real world, the selection of RDs followed specific criteria:

- **Include multiple domains**. The effectiveness of anomaly detection tools may vary depending on the domain. To avoid this bias, we included RDs from different domains;
- **Be in English**. Most academic and commercial tools for detecting requirements anomalies use English (Arendse and Lucassen [2016]). A cataloged tools analysis by Nascimento *et al*. [2018] confirmed they all support only English-written requirements;
- **Contain multiple types of anomalies**. A key characteristic of an anomaly detection tool is its ability to handle a variety of anomalies (Arendse and Lucassen [2016]). Therefore, the selection process prioritized RDs that included multiple types of anomalies.

The selection process retrieved the RDs from the `PURE`[6] (PUblic REquirements) dataset, adhering to the criteria

[6]https://fmt.isti.cnr.it/nlreqdataset/

Table 3. Selection of tools for comparative analysis

ID	Tools	ID Paper	C1	C2	C3	C4	Sel.
1	RETA	1,15,17	x	x	x	x	Yes
2	Prototype*	2	x			x	No
3	ConQAT	3,18	x			x	No
4	Smella	4,15,16,17,20				x	No
5	GATE	5,15,16,20	x			x	No
6	Tactile Check	6	x	x	x	x	Yes
7	Holmes - Qualicen	7				x	No
8	RAQ	7			x	x	No
9	Tiger Pro	7	x	x	x	x	Yes
10	Teamscale	8		x	x		No
11	QuaRS Express	10,14,19	x			x	No
12	SREE	10,14,15,20			x	x	No
13	Scout - Qualicen	11			x	x	No
14	AQUSA	12,19		x	x	x	No
15	Ambiguity Detection	14	x			x	No
16	Poirot	19					No
17	Dowser	19	x			x	No
18	RAI	19					No
19	ARM tool	20	x			x	No
20	Nvivo	21		x		x	No
21	No name given	25		x	x	x	No
22	No name given	26		x	x	x	No

Legend: C1 – Availability; C2 – Automation; C3 – Documentation; C4 – Requirements Description Format.

above. This dataset provides 79 public requirements documents written in Natural Language, covering various domains (Ferrari *et al*. [2017b]). Researchers frequently use PURE because most studies on requirements anomaly detection rely on private documents and rarely share their requirements databases, making replicating experiments or generalizing results difficult (Ferrari *et al*. [2017b]).

Researchers with over ten years of experience teaching and researching Requirements Engineering reviewed the selected RDs and identified anomalies. They manually validated the anomalies to create an oracle, which served as an evaluation benchmark during the comparative analysis. The selection process identified three RDs for evaluating the tools, considering cataloged anomalies by Nascimento *et al*. [2021] (Table 1).

Table 4 lists the selected RDs, their identifiers, titles, and the number of requirements. DR01 describes the functional and non-functional requirements for an "Integrated Library System" Administration Module. The document organizes requirements into General, Consoles and Dashboards, Business Rules, Data Recovery, Security, Maintenance, Customer Management, Queries, and Reports. Table 5 provides examples of requirements from the "General" category from DR01. The selection process preserved the original English text to maintain semantic accuracy when testing the tools. In REQ03, the analysis identified a "Loopholes" anomaly. The "reasonable" term introduces ambiguity and subjectivity since its interpretation can vary (Femmer *et al*. [2017]).

Table 4. Selected Requirements Documents

ID	Title	Number of Requirements
DR01	Integrated Library System Specification	51
DR02	DigitalHome Software Requirements Specification	92
DR03	Crime And Criminal Tracking Network And Systems	122

Table 5. Examples of Requirements from DR01 - Adapted from PURE

ID	Requirement
RE01	System runs on a fully relational SQL-based database system. Ability to run SQL queries against any table in the database. Ability to access database as an ODBC source. All data tables and data storage are fully accessible.
RE02	The system provides real-time processing. For example, pull lists are up to date at the time of viewing or printing; the system supports live shelf reading and weeding.
RE03	Ability for multiple staff members and patrons to simultaneously access and update patron and item records, including on-staff check-in and check-out terminals, on self check-out stations, through SIP2/NCIP2 and similar protocols and APIs, and in OPAC. Depending on assigned privileges, staff can view all patron and item fields; patrons can access only selected files. Record changes are applied in a **reasonable** way, with prompts to warn when a record has been changed since it was displayed.

DR02 outlines the requirements for developing a "Smart House" system called DigitalHome (DH), managed by the DigitalHomeOwner Division of HomeOwner Inc. This requiremetns document describes a system that allows homeowners to manage environmental control devices. DR02 lists the system's key features and categorizes requirements into Functional and Non-Functional Requirements, dividing them into General, Security, Reporting, and Performance sections.

DR03 describes the Crime And Criminal Tracking Network And Systems (CCTNS), an Indian government initiative to develop a nationwide, integrated software system for policing. The system includes an online tracking program connecting thousands of police stations nationwide. DR03 provides a detailed description of the functional requirements for the first version of the CCTNS. The document categorizes the requirements into Functional and Non-Functional Requirements and organizes them into modules such as Navigation, Configuration, Registration, Investigation, and Indictment.

Table 6 presents the number of requirements with anomalies in each RD and specifies the types of identified anomalies. Some requirements contain more than one type of anomaly. It is worth emphasizing that the PURE dataset contains only raw requirement sentences, without any predefined anomaly labels. The identification of anomalies in this study was performed manually by the authors, using the taxonomy proposed by Nascimento *et al*. [2021] as a reference.

This manual analysis served as the oracle for the evaluation of the tools.

Table 6. Anomalies identified in the DR

ID	Requirements with anomalies	Types found
DR01	24	Loopholes, Incomplete References, and Ambiguous Adverbs and Adjectives
DR02	10	Loopholes, Superlative, Vague Pronouns, and Non-verifiable Terms
DR03	39	Ambiguous Adverbs and Adjectives, Loopholes, Non-verifiable Terms, and Negative Statements

3.4 Setting up the Environment

This comparative analysis aims to evaluate the quality of the selected tools based on precision and recall (as defined in Section 4). The analysis conducted in this study was intentionally limited to the level of individual requirement sentences, as represented in the available datasets. Consequently, only the anomaly categories applicable to single-sentence evaluation were considered. Broader issues such as inconsistency, redundancy, or omission across multiple requirements were not addressed. For instance, AN11 (*Duplicate Functionality*) requires examining relationships among several sentences or documents, which falls outside the intended scope of this study. Future work will expand this analysis by incorporating additional criteria to establish a more comprehensive benchmark for requirements anomaly detection tools.

The process began with setting up the environment, which included selecting the operating system, installing necessary plugins and frameworks, verifying user license requirements, and checking the minimum hardware prerequisites for each tool. The analysis focused on the RETA, Tactile Check, and Tiger Pro tools. Among them, only the Tiger Pro tool requires a commercial license, which unlocks advanced configuration options, tool customization, and additional features. However, anomaly detection is a built-in feature available in its free version. The other tools do not necessitate a proprietary user license, granting access to all features.

Each tool supports a specific set of anomalies, often using different names for the same type of issue. Additionally, an anomaly in one tool may correspond to multiple categories in another. To standardize the comparison, we created a "From-To" mapping, aligning the anomalies analyzed in this study[7] with those identified by the tools. Each cataloged anomaly was assigned a unique ID (Table 1). We excluded anomalies that did not fit into predefined categories to ensure consistency and uphold the integrity of the comparative analysis.

The RETA tool is an application within the GATE Developer (GATE) work environment; users need to install it before importing the RETA tool. This open-source tool follows the GNU LGPL 3.0; users can access it from the authors' website[8]. The GATE tool supports multiple platforms, including Linux, Windows, and macOS, and requires Java 8 or later. The RETA tool processes 10 types of anomalies (Arora *et al*. [2015]), but only five align with the cataloged anomalies. Table 7 shows the "From-To" relationship between the detected anomalies by the RETA tool and the cataloged anomalies. For example, the Warn Adverb in Verb Phrase anomaly corresponds to three types of cataloged anomaly (AN1, AN2, and AN4). Since adverbial verb phrases can introduce imprecision and leave crucial details implicit (e.g., The S&T module should periodically query the database for EDTM CSI information), the authors determined the classification of each instance. The RETA tool was tested on Microsoft Windows 11 using the GATE tool version 8.1.

Table 7. Cataloged Anomalies X Detected Anomalies by the RETA Tool

Sigla	Anomalias catalogadas	Anomalias da ferramenta *RETA*
AN1	Ambiguous Adverbs and Adjectives	Warn Adverb in Verb Phrase
AN2	Loopholes	Warn Adverb in Verb Phrase
AN3	Non-verifiable Terms	Warn Vague Terms
AN4	Negative Statements	Warn Adverb in Verb Phrase
AN10	Passive Voice	Warn Passive Voice
AN6	Subjective Language	Warn Quantifier
AN7	Vague Pronouns	Warn Pronoun

The Tactile Check tool[9], created in 2016, operates as a Visual Basic for Applications macro within Microsoft Word 2013. It identifies anomalies by adding annotations directly to sentences in the document. As it integrates with Word, it allows users to detect anomalies within the same environment where the requirements are present. Although the Tactile Check tool is free, it requires a commercially licensed version of Microsoft Word. This tool detects five types of anomalies [31, 38]. Table 8 presents the "From-To" mapping between the detected anomalies by the Tactile Check tool and the cataloged anomalies. The Subjective Language anomaly, for instance, corresponds to two types of cataloged anomaly (AN1 and AN6). The Subjective Language anomaly includes terms with ambiguous meanings, such as "friendly", "easy to use", and "economical". We tested the Tactile Check tool in Word (version 17) on a Windows 11, both licensed Microsoft products.

Table 8. Cataloged Anomalies X Detected Anomalies by the Tactile Check tool

ID	Cataloged Anomalies	Anomalies *Tactile Check*
AN1	Ambiguous Adverbs and Adjectives	Subjective Language
AN2	Loopholes	Non-verifiable Terms
AN3	Non-verifiable Terms	Loopholes
AN4	Negative Statements	Negative Statements
AN5	Incomplete References	Incompleteness
AN6	Subjective Language	Subjective Language

[7] These anomalies will be described as "cataloged anomalies".
[8] https://gate.ac.uk/download
[9] https://github.com/mwmk67/TactileCheck

The `Tiger Pro` tool[10], developed in 2006 using Borland's Delphi programming language (`Visual Pascal`), serves as a front-end tool to assist Requirements Engineers in writing effective requirements (Arendse and Lucassen [2016]). It runs exclusively on the `Microsoft Windows` platform. The `Tiger Pro` tool is shareware, requiring a license for use beyond 14 days, except for academic purposes (the owner provides a free license). Currently, two versions of the `Tiger Pro` tool are available: 2.0 and 2.1 (beta). The beta version offers additional features, such as color highlighting for each anomaly type and the capability to fix certain defects. However, since the research focuses on anomaly detection, we considered version 2.0 a stable and consolidated version. The `Tiger Pro` tool handles seven types of anomalies (Kasser *et al*. [2003]), but only one - `Unverifiable` - could be mapped to the cataloged anomalies. This anomaly refers to subjective terms that do not express an exact value for a function, such as "easy", "fast", or "friendly". Table 9 shows the "From-To" relationship between the cataloged anomalies and detected anomalies by the `Tiger Pro` tool. We observed that the `Unverifiable` anomaly mapped to all four types of cataloged anomalies (`AN1`, `AN2`, `AN3`, and `AN4`). The authors decided on the classification in these cases and reviewed it with two other researchers. The `Tiger Pro` tool ran in a `Microsoft Windows` environment (version 11).

Table 9. Cataloged Anomalies X Detected Anomalies by the `Tiger Pro` tool

ID	Cataloged Anomalies	Anomalies *Tiger Pro*
AN1	Ambiguous Adverbs and Adjectives	Unverifiable
AN2	Loopholes	Unverifiable
AN3	Non-verifiable Terms	Unverifiable
AN6	Subjective Language	Unverifiable

Table 10 shows the results of the correspondence between the cataloged anomalies and those detected by each selected tool. The anomalies `AN8`, `AN9`, and `AN11` anomalies, which correspond to the "Comparative", "Superlative", and "Duplicate Functionality" anomalies, respectively, fall outside the detection scope of the tools.

Table 10. Type of cataloged anomalies Detected by the tools

ID	Anomalies	*RETA*	*Tactile Check*	*Tiger Pro*
AN1	Ambiguous Adverbs and Adjectives	X	X	X
AN2	Loopholes	X	X	X
AN3	Non-verifiable Terms	X	X	X
AN4	Negative Statements	X	X	
AN5	Incomplete References		X	
AN6	Subjective Language	X	X	X
AN7	Vague Pronouns	X		
AN8	Comparative			
AN9	Superlative			
AN10	Passive Voice	X		
AN11	Duplicate Functionality			

[10]https://www.scenarioplus.org.uk/vendors.htm#TigerPro

3.5 Choosing the Comparative Analysis Metrics

We evaluated the performance of the anomaly detection tools using Recall, Precision, and F-measure metrics (Arora *et al*. [2015]; Nakache *et al*. [2005]; Herrera *et al*. [2012]).

An **oracle** was created by manually labeling each requirement sentence in the dataset with the presence or absence of specific anomalies, following the taxonomy proposed by Nascimento et al. Nascimento *et al*. [2021]. This manual labeling served as the ground truth for comparison.

Each tool was evaluated against this oracle, resulting in the classification of predictions into the following categories:

- **True Positive (TP):** A requirement sentence correctly identified by the tool as containing a specific anomaly that is also labeled as such in the oracle.
- **False Positive (FP):** A requirement sentence identified by the tool as containing an anomaly, but not labeled as such in the oracle.
- **False Negative (FN):** A requirement sentence labeled in the oracle as containing an anomaly, but not identified by the tool.

For example, suppose the oracle labels the sentence "The system must sometimes respond in less than 2 seconds" as ambiguous due to the word "sometimes". If a tool also flags this sentence as ambiguous, it is a TP. If it does not, it is a FN. If the tool incorrectly flags "The user must press the login button to proceed" as ambiguous, this is a FP.

Based on this classification, we compute:

- **Precision (P)** = $\frac{TP}{TP+FP}$, representing the proportion of identified anomalies that are correct.
- **Recall (R)** = $\frac{TP}{TP+FN}$, representing the proportion of actual anomalies that were correctly identified.

F-measure, in turn, represents the harmonic mean of **Precision** and **Recall** into a single value (Sasaki *et al*. [2007]):

$$F = \frac{2 * P * R}{P + R}$$

4 Results of the Anomaly Detection

We recorded the anomaly detection results for each tool in an electronic spreadsheet. Each spreadsheet includes four tabs: one for each Requirements Document and another for the consolidated results. Each tab describes the requirements, the number and types of anomalies detected by the tools, the oracle data, and the total count of identified anomalies.

After collecting the data, we grouped the anomalies by type and Requirements Document. We calculated the number of false positives and false negatives and correctly detected anomalies. Using this data, we computed the Precision and Recall of each tool. At the end, we summarized the results in tables that present the overall performance, where the `AN1`, `AN2`, ..., and `ANN` acronyms refer to the cataloged anomalies.

All data used in this study is publicly available in the following research repository: `https://zenodo.org/`

records/15212993. This includes the list of anomalies, the requirement documents analyzed, the raw and consolidated evaluation spreadsheets, and other supporting materials.

4.1 Results of the RETA, Tactile Check, and Tiger Pro tools

Tables 11, 12, and 14 summarize the overall test results for the RETA, Tactile Check, and Tiger Pro tools based on their analysis of the three Requirements Documents. We organized each table into three sections: (a) The first three rows display the detection results, showing anomalies identified by the tool versus the manually validated oracle, including False Negatives (FN), False Positives (FP), and Correct Detections (OK); (b) The last two rows present Precision and Recall percentages for each anomaly type; and (c) The last column sums the values across anomaly types. We marked the cells with no applicable data with a dash ("-"). Table 15 shows the F-measure of each tool, representing the balance between Precision and Recall.

4.2 Strengths and Weaknesses

We analyzed the results to identify the strengths and weaknesses of each tool during anomaly detection.

The RETA tool achieved an overall Recall of 65.43% but had a low Precision of 14.29%. The significant gap is due to many false positives, particularly related to anomaly AN3 (Non-verifiable Terms). This tool often misidentifies certain terms, such as "*support*, *include*, and *use*" as anomalies without considering their context. Although the RETA detects many cataloged anomalies, its false positives increase the workload for Requirements Engineers. This issue stems from its lack of contextual analysis, often incorrectly marking logical connectors like "*OR*" as anomalies. Although the dictionary of the RETA tool is not publicly accessible, it demonstrated the highest correlation recall with cataloged anomalies among the three tools. However, its low accuracy suggests that the incorporation of contextual analysis could significantly improve its performance in distinguishing actual anomalies from false positives.

The Tactile Check tool delivered an total Recall of 65.63% and Precision of 56.25%, offering the most balanced performance. Ideally, recall and precision should be the same, making this the most effective scenario among the three tools. The highest number of false positives was related to AN6 anomaly (Subjective Language) across all Requirements Documents, likely because it does not account for context or grammar (Wilmink and Bockisch [2017]). In terms of recall, the Tactile Check tool ranked second after the RETA tool. Although it accurately detected the AN1, AN2, and AN4 anomalies, false positives reduced its overall performance. Contextual analysis could further enhance its precision.

The Tiger Pro tool had the lowest overall recall (46.88%) and a precision of 43.27%. It detected the fewest cataloged anomalies among the three tools. Like the others, it does not analyze context before marking a term as an anomaly, which leads to inaccuracies. This tool produces many false positives without a structured dictionary or contextual analysis, complicating the Requirements Engineer's analysis. Despite this, it exhibited the lowest correlation coverage with cataloged anomalies compared to the RETA and Tactile Check tools. The Tiger Pro tool maintained a relatively high accuracy. Like the other tools, integrating contextual analysis into its anomaly detection process could enhance its accuracy.

The F-measure values were 23.45% for the RETA tool, 60.68% for the Tactile Check tool, and 45% for the Tiger Pro tool. Among them, the Tactile Check tool achieved the best overall performance. However, despite its superior performance, none of the tools demonstrated satisfactory recall and precision. Requirements engineers would still need to manually validate anomalies, limiting their professional applicability. Improving the reliability of anomaly detection techniques remains a key challenge.

We also compared the tools in three qualitative aspects: (a) setup and installation of the environment, (b) operation and usability, and (c) presentation of the results. It is important to note that some criteria assessed during installation, configuration, and operation are inherently subjective. We used terms such as "complex", "intuitive", "easier", "just", and "simple" to categorize processes based on the level of research effort and steps required for execution.

4.2.1 Environment Setup and Installation

We identified several advantages of the RETA tool. It is a multi-platform tool successfully tested on Windows and Linux. Additionally, the only prerequisite for its execution is Java 9 or higher, which simplifies the installation process. Another positive aspect is the availability of comprehensive installation and configuration documentation. In this regard, we did not identify any significant drawbacks.

We observed some notable strengths regarding the Tactile Check tool. It operates within Microsoft Word, a widely used software for requirements document creation, which enhances its accessibility and ease of use. Furthermore, it does not require installation and includes detailed documentation covering its operation and parameterization. However, a significant drawback is that it requires macro execution, which forces users to turn off all security settings related to macro execution, potentially raising security concerns.

As for the Tiger Pro tool, we found two main advantages. A single executable installs all necessary dependencies, streamlining the setup process. Additionally, it provides installation and configuration documentation to assist users. However, its main limitation is that it is only compatible with Windows, which restricts its usability for users of other operating systems.

4.2.2 Operation and Usability

The usability experience varied significantly depending on the underlying technology of each tool. However, all three tools accepted Requirements Documents in plain text format using the "ID - Description" structure.

Table 11. Results: Anomalies Detection - `RETA` Tool

	AN1	AN2	AN3	AN4	AN5	AN6	AN7	Total
FN	12	10	5	0	0	0	1	28
FP	7	2	76	13	119	64	37	318
TP	5	5	41	2	0	0	0	53
ORACLE	17	15	46	2	0	0	1	81
Recall	29.41%	33.33%	89.13%	100%	-	-	0%	65.43%
Precision	41.67%	71.43%	35.04%	13.33%	0%	0%	0%	14.29%

Table 12. Results: Anomalies Detection - `Tactile Check` Tool

	AN1	AN2	AN3	AN4	AN5	AN6	Total
FN	16	7	8	2	0	0	33
FP	0	20	2	0	0	27	49
TP	1	16	42	2	0	2	63
ORACLE	17	23	50	4	0	2	96
Recall	5.88%	69.57%	84%	50%	-	100%	65.63%
Precision	100%	44.44%	95.45%	100%	-	6.90%	56.25%

Table 13. Results: Anomalies Detection - `ChatGPT`

	AN1	AN2	AN3	AN4	AN5	AN6	Total
FN	16	22	25	4	0	2	69
FP	1	2	13	2	3	6	27
TP	1	1	25	0	0	0	27
ORACLE	17	23	50	4	0	2	96
Recall	5.88%	4.35%	50%	0%	-	0%	28.13%
Precision	50%	33.33%	65.79%	0%	0%	0%	50%

Table 14. Results: Anomalies Detection - *Tiger Pro* Tool

	AN1	AN2	AN3	AN6	Total
FN	13	8	28	2	51
FP	1	11	40	7	59
TP	4	17	24	0	45
ORACLE	17	25	52	2	96
Recall	23.53%	68%	46.15%	0%	46.88%
Precision	80%	60.71%	37.50%	0%	43.27%

Table 15. Results: Anomalies Detection - Three Tools

Tool	Recall	Precision	*F-measure*
RETA	65.43%	14.29%	23.45%
Tactile Check	65.63%	56.25%	60.58%
Tiger Pro	46.88%	43.27%	45.01%

For the `RETA` tool, we identified both advantages and limitations. One of its key strengths is its support for multiple input formats, including `.txt`, `.xml`, and `.HTML`, providing flexibility in document handling. However, we observed several limitations that impact usability. First, it lacks dedicated operational documentation. While the `GATE` tool, which the `RETA` tool relies on, offers extensive documentation, the `RETA` does not include a user manual, making it difficult for users to navigate its functionalities. Additionally, it does not allow users to modify anomaly detection parameters, limiting customization. Another significant drawback is the `GATE` interface, which is not intuitive and makes it unclear how to execute the necessary commands for anomaly detection. Moreover, the detection process is time-consuming and requires extensive configuration, further complicating its usability.

We found several positive aspects regarding the `Tactile Check` tool. It features an intuitive graphical interface that is easy to understand and use, reducing the learning curve for new users. Additionally, it allows users to configure anomaly detection settings easily, providing greater flexibility. One of its most notable advantages is its seamless integration with Microsoft Word, enabling users to detect anomalies directly within the document without switching to an external environment. Despite these benefits, we identified some drawbacks. The tool does not allow direct import of Requirements Documents for anomaly detection, requiring users to manually transcribe requirements into a Microsoft Word document containing the necessary macros. Furthermore, its usage is restricted to users with a Microsoft Word license, which may pose a barrier for those without access to the software.

As for the `Tiger Pro` tool, we observed several advantages. It supports the `.txt` and `.CSV` file formats, offering compatibility with commonly used document formats. It also includes user documentation, facilitating the learning process, and features a user-friendly graphical interface that enhances usability. However, we identified two main drawbacks. The tool is limited to the Windows platform, restricting its accessibility to users of other operating systems. Additionally, the student license imposes restrictions by not allow-

ing users to configure anomaly detection settings, reducing its adaptability for different use cases.

4.3 ChatGPT Analysis

ChatGPT, developed by OpenAI, is one of the most popular Large Language Models (LLMs) at the time of writing this paper. The use of ChatGPT as a supporting tool for requirements engineering activities has been investigated in several studies (Arora *et al*. [2024]; Vogelsang [2024]; Ronanki *et al*. [2024]). However, to the best of our knowledge, no studies have presented evidence regarding the use of ChatGPT for detecting anomalies in software requirements. This section aims to evaluate the accuracy of ChatGPT in supporting the task of anomaly detection in software requirements, following the methodology used by Felizardo *et al*. [2024].

To achieve the proposed objective, we used the same dataset employed in the evaluation of the other tools, the requirements documents `DR01`, `DR02`, and `DR03` (see Table 6), which contain a total of 85 requirements with anomalies previously detected and validated by software engineering exports. We chose to compare ChatGPT's performance in anomaly detection exclusively with `Tactile Check`, the tool that demonstrated the best results among those evaluated. Accordingly, only the anomalies effectively covered by this tool were considered in the analysis (`AN1` to `AN6`). This decision ensures a fair comparison based on a high-quality benchmark, avoiding the inclusion of anomalies not recognized by the reference tool.

As presented in Figure 1, we followed three main steps to evaluate the accuracy of using ChatGPT for detecting anomalies in software requirements, as explained below.

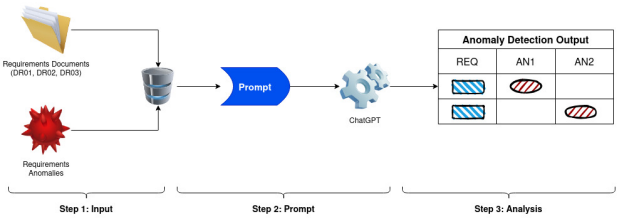

Figure 1. Steps followed to evaluate ChatGPT's accuracy in detecting requirement anomalies.

Step 1: Input — To perform anomaly detection in software requirements using ChatGPT, we provided the following as input: the requirements documents, as well as the list of anomalies `AN1` to `AN6` along with their descriptions, as presented in Table 1.

Step 2: Prompt — We designed a specific input prompt tailored to the context under investigation. The prompt included: (i) the role to be assumed by ChatGPT, for instance, "act as a senior requirements analyst"; (ii) the title of each requirements document (Section 3.3), in order to provide contextual information about the domain to which the requirements were related; (iii) the list of requirements from the document, separated by blank lines; and (iv) the list of anomalies to be identified in each requirements, also separated by blank lines.

We asked ChatGPT to return a list in which each item corresponds to a requirement from the document, followed by the codes of the anomalies identified for that requirement. It was requested that a justification be provided for the selection of the specific anomalies considered in the analysis. Although this choice does not affect the results themselves, it helps to contextualize and better explain ChatGPT's performance by clarifying the criteria used to define which anomalies were relevant for evaluation.

Based on the description provided above, the prompt template proposed in this study is presented below:

Prompt Template

Assume that you are a senior requirements analyst conducting an anomaly detection task on the requirements document entitled *<document title>*.

Consider the following list of requirements, separated by a blank line:

<Requirements list>

Also consider the following list of anomalies separated by a blank line. For each anomaly, there is a code, a name, and a description.

Code: <Anomaly code>
Name: <Anomaly name>
Description: <Anomaly description>

Check for the presence of these anomalies in the given requirements and return the result as a list of items separated by a blank line, following the format bellow:

Requirement: <Requirement text>
Anomalies: <Codes of the anomalies affecting the requirement, separated by commas>
Justification: <Reasons for the selection of the specific anomalies considered in the analysis>

The data required to execute the prompt, such as the list of anomalies and requirements in the appropriate format and all ChatGPT interactions performed during the experiments, can be found in the database provided along with this paper.

Step 3: Analysis — The summary of the results is presented in Table 13, which follows the same format as the tables reporting the results of the other tools. Regarding anomaly `AN1` (*Ambiguous Adverbs and Adjectives*), the results obtained with ChatGPT were similar to those of the `TactileCheck` tool, both showing a high incidence of false negatives. In other words, several occurrences manually identified by experts were not recognized by either automated approach. This outcome highlights the ongoing challenge that linguistic ambiguity poses, even for advanced Natural Language Processing (NLP) technologies.

The prevalence of false negatives was observed across most of the analyzed anomaly categories, suggesting that merely providing the anomaly descriptions is insufficient for generative models like ChatGPT to achieve broad recall. This limitation is partly due to the fact that ChatGPT is a general-purpose AI not specifically trained for anomaly detection in requirements. The difficulty is particularly evident

in the categories *Non-verifiable Terms* and *Loopholes*, whose detection depends on contextual interpretation and domain-specific normative knowledge—dimensions that generalist models may not capture effectively.

Conversely, with respect to false positives, ChatGPT yielded more favorable results. Specifically, for anomalies AN6 (*Subjective Language*) and AN2 (*Loopholes*), the model exhibited a significantly lower rate of false positives when compared to TactileCheck. This difference may be attributed to ChatGPT's ability to interpret expressions in context, avoiding premature classifications based solely on the presence of potentially problematic terms. TactileCheck, in contrast, relies on a rule-based approach using fixed patterns or keyword lists, often disregarding the communicative intent or semantic function of terms within the requirement. In this regard, ChatGPT demonstrated greater parsimony in anomaly marking, reducing misclassifications in contexts where subjectivity or unverifiability does not necessarily compromise clarity or testability.

From a quantitative standpoint, the comparison of overall metrics reveals significant differences between the two approaches. ChatGPT achieved a **recall** of 28.13%, whereas TactileCheck reached 65.63%, underscoring the generative model's limited ability to detect the full range of anomalies. However, **precision** for ChatGPT (50.00%) was close to that of TactileCheck (56.25%), suggesting that, although it identifies fewer anomalies, it tends to classify them more conservatively. This higher selectivity aligns with the previously discussed reduction in false positives. Lastly, the **F-measure**, which balances recall and precision, was also higher for TactileCheck (60.58%) compared to ChatGPT (36.00%), indicating a more robust overall performance by the specialized tool.

These findings suggest that although LLMs are effective in generating text with good understandability, grammatical correctness, and even consistency, they exhibit significant difficulties with aspects such as unambiguity and feasibility of requirements, as reported by Ronanki *et al.* [2024]. Another relevant factor is that traditional tools like Tactile Check are based on explicitly defined rules and well-established criteria to detect anomalies such as ambiguity, subjectivity, and lack of atomicity. These tools are specifically designed and tuned for this purpose, whereas ChatGPT was not originally trained for anomaly detection in requirements. As pointed out by Vogelsang [2024], LLMs like ChatGPT may produce coherent reasoning and plausible explanations, but they do not necessarily adhere to formal rules or standardized taxonomies — which undermines their accuracy in structured verification tasks.

4.4 Final Analysis

Detecting requirements anomalies typically involves identifying and highlighting terms that may indicate potential issues in the Requirements Document and categorizing them accordingly. Various techniques can be employed for this process, and their effectiveness can either facilitate or hinder the Requirements Engineer's analysis.

In terms of result presentation, the RETA tool offers several advantages. It highlights each anomaly using a distinct color, allows users to select one or multiple anomalies simultaneously, and provides a color-coded legend that specifies the anomaly type. However, it also has notable limitations. The tool does not generate a summary report of the detection results, lacks the option to export results to an external file, and, when multiple anomalies are selected, overlapping colors can make visualization confusing or even incomprehensible, especially when a single term is associated with multiple anomalies.

The Tactile Check tool demonstrates key strengths in its result presentation. It highlights anomalies based on their type and explains each anomaly directly within the document using a "tag" system, ensuring clarity and ease of interpretation. We did not observe any significant drawbacks.

The Tiger Pro tool also offers several advantages. It summarizes results by anomaly type, displaying the detected term, count, and type of anomaly directly below each requirement. It also allows users to export results in the .CSV format and present them in an on-screen report. We did not observe any significant drawbacks.

Considering the F-measure results and the strengths and weaknesses related to environment setup, installation, usability, and result presentation, the Tactile Check tool demonstrated the best overall performance among the evaluated tools. However, its requirement to turn off all security mechanisms related to macro execution may present a critical limitation, potentially making it unsuitable for deployment in real-world projects where security concerns are a priority.

The use of the ChatGPT model, although not originally designed for anomaly detection in requirements, presents notable advantages regarding ease of use and flexibility. It requires no installation or environment configuration and can be accessed via a web interface or API, reducing setup effort significantly. Furthermore, its natural language interface enables iterative refinement of prompts, allowing users to adapt the detection process to different document styles and project contexts without relying on fixed rules or predefined interfaces. However, despite these usability benefits, the empirical results revealed important performance limitations. ChatGPT achieved the lowest recall and F-measure among the evaluated tools, which indicates a limited ability to comprehensively detect anomalies. Using ChatGPT as-is, without any adaptation or fine-tuning for the specific task, may lead to the omission of critical issues in requirements documents, potentially compromising the quality of the software development process. Therefore, while the model shows promise as a flexible and accessible tool, its direct application to anomaly detection should be approached with caution.

The findings of this study have implications for both researchers and industry practitioners. For researchers, the results highlight the need for continued investigation into how different detection mechanisms—rule-based, pattern-based, and generative—perform when applied to natural language requirements. The limitations observed in ChatGPT's performance also point to valuable research opportunities in prompt engineering, fine-tuning, and interpretability of generative models within the RE domain. For practitioners, the analysis provides evidence-based insights into the strengths and weaknesses of existing tools. While dedicated tools like RETA and Tactile Check demonstrated better precision and

recall, the flexibility and usability of ChatGPT suggest its potential as a lightweight, accessible option for early-stage reviews or educational contexts. Understanding these trade-offs can help practitioners make informed decisions when selecting tools for anomaly detection in real-world projects.

Finally, we acknowledge that the "traditional" tools and the LLM may have identified anomalies not originally marked in the oracle, which could correspond either to genuine issues that were missed during manual labeling or to spurious detections. This situation affects precision but not necessarily recall, and it highlights the inherent subjectivity of manual annotation in requirements analysis.

5 Final Remarks

Despite various tools for detecting requirement anomalies, few studies provide a comparative analysis of existing solutions. This study aimed to bridge this gap by conducting a comparative evaluation of three tools that automatically detect requirement anomalies in natural language: `RETA`, `Tactile Check`, and `Tiger Pro`. Given the critical role of high-quality requirements documents and the impact of anomalies on subsequent stages of software development, this research contributes in several ways. Specifically, it (i) updated an MSL on requirements anomalies, (ii) addressed a research gap by analyzing tools for anomaly detection, (iii) created a database of three requirements documents with identified and categorized anomalies, and (iv) compiled a comprehensive assessment of the strengths and weaknesses of the analyzed tools.

Additionally, we evaluated the ChatGPT model — an advanced generative AI — against the tool with the best overall performance, `Tactile Check`. While ChatGPT was not among the tools originally designed for anomaly detection, its inclusion aimed to explore the potential of general-purpose AI models for this task. The results revealed promising aspects, such as reduced false positives in certain categories, but also significant limitations in recall and overall effectiveness. The main contributions of this study are threefold: (i) a comparative analysis between a general-purpose LLM and traditional requirement anomaly detection tools; (ii) the provision of an open, reproducible dataset; and (iii) a discussion on the limitations of LLMs for this specific task.

However, this study has limitations and potential threats to validity: 1) The number of tools examined was constrained. Although many approaches and tools exist for anomaly detection, some are unavailable or discontinued. Additionally, the tools were developed in different periods, which presents a challenge, as more recent tools may detect a broader range of anomalies; 2) The study considered only three requirements documents, and some anomalies detectable by the tools were absent from these documents, potentially affecting the evaluation of the tools; 3) This study is limited to the analysis of individual requirement sentences. Consequently, it does not detect anomalies that arise from relationships between multiple requirements, such as redundancy, inconsistency, or omission; 4) We recognize that other types of anomalies, such as the use of quantifiers like "all" are also relevant. However, since the objective of this study was to compare the performance of existing detection tools rather than define a new taxonomy, we intentionally maintained the same list proposed by Pereira *et al*. [2024] to preserve comparability.; 5) Researchers manually defined and developed the oracle for anomaly detection. Multiple researchers reviewed the process to mitigate potential errors; 6) We manually mapped the cataloged anomalies and those detected by the tools. However, experts in Requirements Engineering rigorously reviewed this process; 7) The evaluation of the tools did not include end-users' feedback. Subjective criteria were assessed solely from the authors' perspectives; 8) Finally, a noteworthy threat to validity concerns the evaluation of ChatGPT in its default configuration, without any task-specific tuning or prompt engineering. As such, the results may not fully reflect the model's potential when properly adapted for anomaly detection.

Future research should explore several directions to build upon these findings. First of all, future work involves conducting a broader systematic mapping study on requirement anomalies, including more established terms in the requirements engineering community, such as "requirement defect". In addition to expanding the set of keywords used, it is also important to extend the time frame of the mapping. The update performed in this study was limited to the period from January 2013 to May 2023, which may have excluded relevant tools or approaches proposed outside this range. A broader and more recent mapping could provide a more comprehensive view of existing techniques and trends in the field. Another important improvement would be to involve industry professionals in the evaluation of the tools, ensuring that the study better reflects real-world practices and addresses practical usability concerns. Lastly, further investigation into the use of generative AI models—such as ChatGPT—for anomaly detection is warranted, particularly through fine-tuning or the design of domain-specific prompts. Given the flexibility and accessibility of such models, there is considerable room for improvement that could make them viable alternatives or complements to existing solutions. In this study, however, we designed the prompt to return only the identifiers of the detected anomalies, without requesting ChatGPT to explain or justify its choices. This decision was made to standardize the output format and facilitate a fair comparison with other tools. Nevertheless, analyzing the model's reasoning—by prompting it to explain its classifications—represents a valuable direction for future work, as it may uncover patterns of misclassification and support more effective adaptations of generative models for the domain of requirements engineering.

References

Arendse, B. and Lucassen, G. (2016). Toward tool mashups: Comparing and combining nlp re tools. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 26–31. DOI: 10.1109/REW.2016.019.

Arora, C., Grundy, J., and Abdelrazek, M. (2024). *Advancing Requirements Engineering Through Generative AI: As-*

sessing the Role of LLMs, pages 129–148. Springer Nature Switzerland, Cham. DOI: 10.1007/978-3-031-55642-5$_6$.

Arora, C., Sabetzadeh, M., Briand, L., and Zimmer, F. (2015). Automated checking of conformance to requirements templates using natural language processing. *IEEE Transactions on Software Engineering*, 41:1–1. DOI: 10.1109/TSE.2015.2428709.

Arora, C., Sabetzadeh, M., Briand, L., and Zimmer, F. (2017). Automated extraction and clustering of requirements glossary terms. *IEEE Transactions on Software Engineering*, 43(10):918–945. DOI: 10.1109/TSE.2016.2635134.

Asadabadi, M., Chang, E., Zwikael, O., Saberi, M., and Sharpe, K. (2019). Hidden fuzzy information: Requirement specification and measurement of project provider performance. *Fuzzy Sets and Systems*. DOI: 10.1016/j.fss.2019.06.017.

Asadabadi, M. R., Saberi, M., Zwikael, O., and Chang, E. (2020). Ambiguous requirements: A semi-automated approach to identify and clarify ambiguity in large-scale projects. *Computers & Industrial Engineering*, 149:106828. DOI: https://doi.org/10.1016/j.cie.2020.106828.

Beer, A., Junker, M., Femmer, H., and Felderer, M. (2017). Initial investigations on the influence of requirement smells on test-case design. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 323–326. DOI: 10.1109/REW.2017.43.

Bäumer, F. and Geierhos, M. (2018). Flexible ambiguity resolution and incompleteness detection in requirements descriptions via an indicator-based configuration of text analysis pipelines. DOI: 10.24251/HICSS.2018.720.

Calazans, A. T. S., Paldês, R. Á., Canedo, E. D., Masson, E. T. S., de Albuquerque Guimarães, F., Rezende, K. M. F., de Souza Gonçalves, F., and Mariano, A. M. (2019). Quality requirements and the requirements quality: The indications from requirements smells in a financial institution systems. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 472–480.

Chitchyan, R., Sampaio, A., Rashid, A., and Rayson, P. (2006). In *A Tool Suite for Aspect-Oriented Requirements Engineering*, EA '06, page 19–26, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/1137639.1137644.

Elrakaiby, Y., Ferrari, A., Spoletini, P., Gnesi, S., and Nuseibeh, B. (2017). Using argumentation to explain ambiguity in requirements elicitation interviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 51–60. DOI: 10.1109/RE.2017.27.

Fabbri, S. C. P. F., Ferrari, F. C., and Camargo, K. G. (2014). A atividade de teste sob a perspectiva de qualidade de software. *Revista TIS*, 2(3):164–166.

Felizardo, K. R., Lima, M. S., Deizepe, A., Conte, T. U., and Steinmacher, I. (2024). Chatgpt application in systematic literature reviews in software engineering: an evaluation of its accuracy to support the selection activity. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '24, page 25–36, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3674805.3686666.

Femmer, H. (2013). Reviewing natural language requirements with requirements smells – a research proposal –.

Femmer, H. (2017). Automatic requirements reviews - potentials, limitations and practical tool support. pages 617–620. DOI: 10.1007/978-3-319-69926-4$_5$3.

Femmer, H., Fernández, D. M., Juergens, E., Klose, M., Zimmer, I., and Zimmer, J. (2014). Rapid requirements checks with requirements smells: Two case studies. RCoSE 2014, page 10–19, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2593812.2593817.

Femmer, H., Hauptmann, B., Eder, S., and Moser, D. (2016). Quality assurance of requirements artifacts in practice: A case study and a process proposal. pages 506–516. DOI: 10.1007/978-3-319-49094-6$_3$6.

Femmer, H., MÃ©ndez FernÃ¡ndez, D., Wagner, S., and Eder, S. (2017). Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213. DOI: https://doi.org/10.1016/j.jss.2016.02.047.

Ferrari, A. (2018). Natural language requirements processing: from research to practice. pages 536–537. DOI: 10.1145/3183440.3183467.

Ferrari, A., Donati, B., and Gnesi, S. (2017a). Detecting domain-specific ambiguities: An nlp approach based on wikipedia crawling and word embeddings. pages 393–399. DOI: 10.1109/REW.2017.20.

Ferrari, A., Gori, G., Rosadini, B., Trotta, I., Bacherini, S., Fantechi, A., and Gnesi, S. (2018). Detecting requirements defects with nlp patterns: an industrial experience in the railway domain. *Empirical Software Engineering*, 23(6):3684–3733.

Ferrari, A., Spagnolo, G., and Gnesi, S. (2017b). Pure: A dataset of public requirements documents. pages 502–505. DOI: 10.1109/RE.2017.29.

Ferrari, A., Spoletini, P., Donati, B., Zowghi, D., and Gnesi, S. (2017c). Interview review: Detecting latent ambiguities to improve the requirements elicitation process. pages 400–405. DOI: 10.1109/RE.2017.15.

Garner, P., Hopewell, S., Chandler, J., MacLehose, H., Schünemann, H., Akl, E., Beyene, J., Chang, S., Churchill, R., Dearness, K., Guyatt, G., Lefebvre, C., Liles, B., Marshall, R., García, L., Mavergames, C., Nasser, M., Qaseem, A., Sampson, M., and Wilson, E. (2016). When and how to update systematic reviews: Consensus and checklist. *BMJ*, 354:i3507. DOI: 10.1136/bmj.i3507.

Herrera, J., Macia, I., Salas, P., Pinho, R., Vargas, R., Garcia, A., Araújo, J., and Breitman, K. (2012). Revealing crosscutting concerns in textual requirements documents: an exploratory study with industry systems. In *2012 26th Brazilian Symposium on Software Engineering*, pages 111–120. IEEE.

Hu, W., Carver, J., Anu, V., Walia, G., and Bradshaw, G. (2018). Using human error information for error prevention. *Empirical Software Engineering*, 23. DOI: 10.1007/s10664-018-9623-8.

Kamsties, E., Berry, D., and Paech, B. (2001). Detecting ambiguities in requirements documents using inspections.

Kasser, J., Tran, X.-L., Matisons, S., *et al*. (2003). *Prototype educational tools for systems and software (pets) engineering*. PhD thesis, Australasian Association for Engineering Education.

Kiyavitskaya, N., Zeni, N., Mich, L., and Berry, D. M. (2008). Requirements for tools for ambiguity identification and mea-

surement in natural language requirements specifications. *Requirements engineering*, 13(3):207–239.

Kroth, E. (2012). Emprego de técnicas de representação do conhecimento como forma de apoio à engenharia de requisitos. In *XVIII Congreso Argentino de Ciencias de la Computación*.

Lucassen, G., Dalpiaz, F., Van der Werf, J. M., and Brinkkemper, S. (2016). Improving agile requirements: the quality user story framework and tool. *Requirements Engineering*, 21. DOI: 10.1007/s00766-016-0250-x.

Machado, F. (2018). *Análise e Gestão de Requisitos de Software – Onde nascem os sistemas*. Saraiva Educação S.A.

Nakache, D., Metais, E., and Timsit, J. F. (2005). Evaluation and nlp. In *International Conference on Database and Expert Systems Applications*, pages 626–632. Springer.

Nascimento, R., Aranha, E., Kulesza, U., and Lucena, M. (2018). Requirements smells como indicadores de má qualidade na especificação de requisitos: Um mapeamento sistemático da literatura. In *WER*.

Nascimento, R., Guimarães, E., and Lucena, M. (2021). Requirements smells como indicador de qualidade para histórias de usuários: Estudo exploratório. In *WER*.

Pereira, F. R., Costa, H. A. X., and Parreira, P. A. (2024). A comparative study of tools for anomaly detection in software requirements. In *Proceedings of the XXIII Brazilian Symposium on Software Quality*, SBQS '24, page 11–21, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3701625.3701641.

Ronanki, K., Cabrero-Daniel, B., Horkoff, J., and Berger, C. (2024). *Requirements Engineering Using Generative AI: Prompts and Prompting Patterns*, pages 109–127. Springer Nature Switzerland, Cham. DOI: 10.1007/978-3-031-55642-5_5.

Rosadini, B., Ferrari, A., Gori, G., Fantechi, A., Gnesi, S., Trotta, I., and Bacherini, S. (2017). Using nlp to detect requirements defects: An industrial experience in the railway domain. volume 10153, pages 344–360. DOI: 10.1007/978-3-319-54045-0_24.

Sasaki, Y. *et al*. (2007). The truth of the f-measure. *Teach tutor mater*, 1(5):1–5.

Sommerville, I. (2011). *Engenharia de software*. Pearson Education, 9th edition.

Valente, M. T. (2020). *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. Independente.

Vogelsang, A. (2024). From specifications to prompts: On the future of generative large language models in requirements engineering. *IEEE Software*, 41(5):9–13. DOI: 10.1109/MS.2024.3410712.

Wilmink, M. and Bockisch, C. (2017). On the ability of lightweight checks to detect ambiguity in requirements documentation. volume 10153, pages 327–343. DOI: 10.1007/978-3-319-54045-0_23.

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K., Ajagbe, M., Chioasca, E.-V., and Batista-Navarro, R. (2020). Natural language processing (nlp) for requirements engineering: A systematic mapping study.