2025 IEEE/ACM 47th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER) | 979-8-3315-3711-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICSE-NIER66352.2025.00016

On the Impact of Requirements Smells in Prompts: The Case of Automated Traceability

Andreas Vogelsang*, Alexander Korn*, Giovanna Broccia†, Alessio Ferrari‡, Jannik Fischbach§, Chetan Arora¶
*University of Cologne, Cologne, Germany
{vogelsang,korn}@cs.uni-koeln.de
†CNR-ISTI, Pisa, Italy
giovanna.broccia@isti.cnr.it
‡ University College Dublin, Dublin, Ireland and CNR-ISTI, Pisa, Italy
alessio.ferrari@ucd.ie, alessio.ferrari@isti.cnr.it
§Netlight Consulting GmbH and fortiss GmbH, Munich, Germany
jannik.fischbach@netlight.com
¶Monash University, Melbourne, Australia
chetan.arora@monash.edu

***Abstract*—Large language models (LLMs) are increasingly used to generate software artifacts, such as source code, tests, and trace links. Requirements play a central role in shaping the input prompts that guide LLMs, as they are often used as part of the prompts to synthesize the artifacts. However, the impact of requirements formulation on LLM performance remains unclear. In this paper, we investigate the role of requirements smells—indicators of potential issues like ambiguity and inconsistency—when used in prompts for LLMs. We conducted experiments using two LLMs focusing on automated trace link generation between requirements and code. Our results show mixed outcomes: while requirements smells had a small but significant effect when predicting whether a requirement was implemented in a piece of code (i.e., a trace link exists), no significant effect was observed when tracing the requirements with the associated lines of code. These findings suggest that requirements smells can affect LLM performance in certain SE tasks but may not uniformly impact all tasks. We highlight the need for further research to understand these nuances and propose future work toward developing guidelines for mitigating the negative effects of requirements smells in AI-driven SE processes.**

***Index Terms*—LLMs, Requirements Eng., Smells, Traceability.**

I. Introduction

In the rapidly evolving field of AI, prompt engineering has become a key area of interest, focusing on developing and optimizing prompts for large language models (LLMs). Developers are increasingly relying on LLMs for a variety of software engineering (SE) tasks, such as generating code from requirements [1], deriving test cases from requirements [2], [3], or tracing requirements to code [4]. These tasks often hinge on the quality of the requirements used to prompt the LLM, yet the impact of the specific formulation of requirements on LLM performance is still poorly understood.

Our new idea is to examine the potential role of *requirements smells*—indicators of issues like ambiguity or inconsistency in natural language (NL) requirements—in prompt effectiveness. In SE, requirements smells have long been recognized as potential obstacles that can lead to misinterpretations and lower quality in the software development process [5]. If smells are present in the requirements used as input for LLMs, they could similarly impact the quality of the LLM's outputs in SE tasks.

We conducted experiments using two leading generative LLMs, GPT-4o and Llama 3.1, to explore this phenomenon in the context of automated trace link generation between requirements and code. This task is particularly relevant, as LLMs are increasingly used to automate traceability—a crucial process that ensures alignment between high-level requirements and their implementation in code [6], [4]. We compared the LLMs' performance on trace link generation using 94 requirements and associated code (809 LOC) from five projects (70 trace links), considering non-smelly and smelly requirements.

Our emerging results present a mixed picture. While requirements smells had a small but statistically significant negative impact on predicting if a requirement was implemented in a code segment, they had no significant effect on identifying the specific lines of implementation. This suggests that LLMs can cope with low-quality requirements for the investigated task and the limited complexity of the systems we tested. However, there are also indications that smells affect certain aspects of LLM performance but their impact varies by task and smell type.

These preliminary findings open several research avenues, such as exploring the relevance of smells in other requirements-centric SE tasks, understanding the influence of project scale and domain, developing methods to identify and mitigate smells, and investigating *prompt smells*—the broader implications of "smell" in prompt engineering [7].

II. Background and Related Work

Requirements Smells: "Requirements smells" refer to patterns or characteristics in software requirements that indicate potential issues, leading to problems in downstream development activities [5]. These smells can signify ambiguous, incomplete, inconsistent, or overly complex requirements, resulting in increased costs, delays, or defects in the final

2832-7632/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE-NIER66352.2025.00016

product [8]. Frattini et al. [9] published a catalog of 206 requirements quality indicators (aka *smells*) extracted from a systematic mapping study on 105 relevant primary studies [10]. The authors also categorized requirements smells into three categories: (1) *lexical* smells describe issues in single words or terms, e.g., code = "program source" or "set of rules"?; (2) *syntactic* smells describe issues in word or sentence structures, e.g., *When the system sends a message to the receiver, it shall provide an acknowledgment* (it = "system" or "receiver"?); and (3) *semantic* smells describe issues in interpreting the requirements within its context, e.g., *The system shall generate a report at the end of each day*, (strictly at midnight or at the end of the business hours?). We sampled three types of requirements smells from each of the three categories of this catalog and used them in our study.

Automated Traceability: The ability to trace relevant software artifacts to support reasoning about the quality of the software and its development process plays a crucial role in requirements and software engineering, particularly for safety-critical systems [11]. LLMs have led to considerably better results in solving trace link recovery (TLR) as a classification task [6]. Rodriguez et al. [4] tested the capability of Claude, a generative AI model, to solve TLR tasks directly by prompting it with NL instructions. North et al. [12] use explainable AI techniques to trace input requirements to LLM generated code.

Our paper is the first to analyze the effect of requirements smells on trace link generation. Additionally, our roadmap extends beyond traceability, exploring the impact of smells on other generative SE tasks.

III. Study Design

We aim to assess the impact of requirements smells in prompts on downstream SE tasks. We analyze automated traceability as a representative task, considering the following research questions (RQs):

RQ1: How well can LLMs trace high-quality requirements to existing code?

RQ2: How does the presence of requirements smells in prompts impact the tracing performance?

RQ3: How do different requirement smell categories impact the tracing performance?

To answer the RQs, we perform a benchmark study on five manually curated projects. We evaluate the tracing performance of LLama 3.1 and GPT-4o on well-written requirements (RQ1) and in smelly ones, considering the impact of the number of smells (RQ2) and the smell category, i.e., lexical, syntactic, semantic (RQ3).

A. Study Objects and Data Collection

We experiment on five exemplary cases with 94 requirements of which 70 were implemented in 809 LOC. All five cases represent simple games, which are more or less popular. One of the authors created a list of requirements for all five cases and a Java code file that implements a fraction of the requirements. Another author has reviewed the list of requirements and the code to ensure correctness and sufficient quality. We also created a smelly version for a subset of the requirements. Each smelly requirement represents exactly one type of smell. We sampled nine smell types from a recently published catalog of requirements smells (see Table II) [9]. Table I gives an overview of the considered cases including their number of requirements, how many requirements are implemented in the code, how many requirements have a smelly variant, and how large the code is.

TABLE I: Study Objects

Game	#req.	#implemented req.	#req. with smelly variant	#LOC
Dice	25	19	18	141
Arkanoid	19	14	15	152
Snake	14	7	11	142
Scopa	16	15	13	220
Pong	20	15	15	154
Sum	94	70	72	809

Ground truth creation: Two authors created a ground truth by independently reviewing the smell-free requirements and deciding which requirements are implemented in the code in which LOC ($\kappa = 0.9$). Similarly, two authors reviewed the smelly requirements and decided which should be considered as implemented in the code ($\kappa = 0.74$). Any remaining differences were resolved in two meetings.

Ratio of Smelly Requirements: We investigate the effect of diverging levels of requirements quality by assessing the tracing performance. We define the level of requirements quality by the ratio of smelly requirements.

Smell Category: In RQ3, we are interested in the impact of different smell categories. We test the differences between three smell categories: lexical, syntactic, and semantic smells. Each category is represented by three smell types associated with the category (see Table II).

Language Model: We ran our experiments on two generative LLMs, GPT-4o (gpt-4o-2024-08-06) and Llama 3.1 (70B). Both models have a 128k token context window. We selected these models as they represent the most advanced closed- and open-source LLMs at the time of the study.

B. Tracing Performance Measures

We measure tracing performance by examining an LLM's ability to predict whether a requirement is implemented and in which LOC.

Binary tracing: We define binary tracing accuracy (BTA) as the accuracy of an LLM's prediction on the question of whether or not a requirement is implemented in the code:

$$BTA = \frac{|implemented \wedge traced| + |\neg implemented \wedge \neg traced|}{N}$$

where N is the total number of considered requirements.

LOC tracing: We assess the ability to trace a requirement to locations in the code by measuring the precision, recall, and F_1 of the traced LOC compared with the true lines of code implementing a requirement.

$$LOC\ precision = \frac{|LOC\ is\ traced \wedge LOC\ implements\ req|}{|LOC\ is\ traced|}$$

$$LOC\ recall = \frac{|LOC\ is\ traced \wedge LOC\ implements\ req|}{|LOC\ implements\ req|}$$

TABLE II: Number and types of studied smells (categorization based on Frattini et al. [9])

Smell categories and types	Description	Dice	Arkanoid	Snake	Scopa	Pong	Sum
Lexical smells		**7**	**4**	**4**	**4**	**5**	**24**
subjective language	Words of which the semantics are not objectively defined, such as *user-friendly*, *easy to use*, *cost effective*, etc. Sentences expressing personal opinions or feelings.	5	2	2	2	3	14
optional parts	Sentences containing optional parts, e.g., by using the words *possibly*, *eventually*, *if possible*, *if needed*, etc.	1	1	1	0	2	5
weak verbs	Weak verbs, such as *can*, *could*, *may*, etc.	1	1	1	2	0	5
Syntactic smells		**6**	**4**	**3**	**3**	**4**	**20**
vague pronouns	Pronouns that refer back to a previous part of the text for which the reference is unclear.	1	1	1	1	1	5
passive voice	Sentences using passive voice such that it is unclear who is performing a certain action.	3	2	1	2	2	10
negative phrases	Sentences containing negative modifiers (e.g., *not*) or negative expressions.	2	1	1	0	1	5
Semantic smells		**5**	**7**	**4**	**6**	**6**	**28**
logical inconsistencies	Two requirements, which are connected to the same concepts, contradicting each other.	2	3	1	1	1	8
numerical discrepancies	Two requirements connected to the same concepts, containing inconsistent and/or contradicting numerical information.	1	1	1	1	1	5
ambiguities	Unclear/imprecise sentence parts that can be misunderstood if read by different people.	2	3	2	4	4	15
Sum		**18**	**15**	**11**	**13**	**15**	**72**

F_1 is the harmonic mean between precision and recall. LOC precision, recall, and F_1 are computed for each requirement. To answer our RQs, we compute the mean LOC precision, recall, and F_1[1] of all requirements implemented in the code; we neglect the LOC tracing for requirements not implemented.

C. Study Execution

We crafted a prompt by following the experience with automated traceability prompts reported by Rodriguez et al. [4]. When manually creating the ground truth, we created and followed specific tracing instructions, which we also included in the prompt. The prompt is included in our online material[2]. After executing the prompt, the LLM outputs a JSON object containing the requirement ID, a yes/no field indicating whether the model thinks the requirement is implemented, and a list of numbers indicating the LOCs the model associates the requirements with.

To control any non-deterministic behavior by the LLMs, we set the temperature to zero [13]. We executed each prompt five times and used majority voting to determine the final answer. For a game with n requirements and m smelly versions available, we replaced p of the requirements with their smelly counterparts in each run. We did $2 + 120$ runs, one for each extreme case ($p = 0$ and $p = m$), and 120 for the intermediate cases, with a random selection of p. The quality level of each run was represented by the percentage of smelly requirements in the total set (p/n). The 122 runs allowed us to cover different quality levels while keeping the workload manageable. In total, we ran 3,050 prompts on each LLM, covering five games with 122 samples per game and five runs (majority voting) per sample.

IV. Study Results and Discussion

A. RQ1: General Tracing Performance

Table III shows the binary tracing accuracy (BTA) and the LOC tracing precision, recall, and F_1-score achieved in the runs with 0% smelly requirements. The average BTA is 0.95 for GPT-4o and 0.96 for Llama 3.1, indicating high reliability of the LLMs for this task when considering relatively small, yet realistic, projects like the one included in our study. For a more complex task such as LOC tracing, the performance decreases, but remains within acceptable boundaries, with GPT-4o outperforming Llama 3.1 by 0.08 in terms of F_1, making it the preferred model for this task.

TABLE III: Tracing performance with 0% smelly reqs.

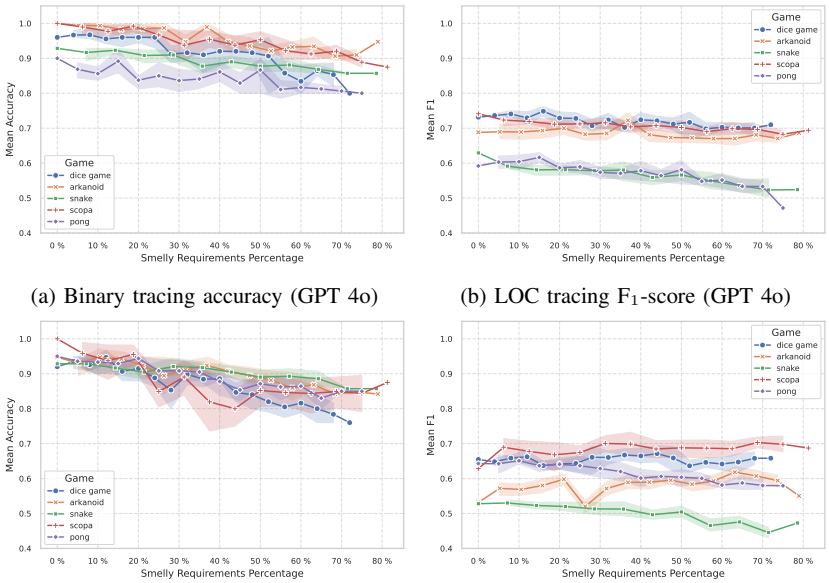
	GPT-4o				Llama 3.1			
Game	**BTA**	**LOC prec.**	**LOC rec.**	**F_1**	**BTA**	**LOC prec.**	**LOC rec.**	**F_1**
dice	0.96	0.71	0.83	0.73	0.92	0.68	0.73	0.65
arkanoid	1.00	0.68	0.80	0.69	0.95	0.64	0.61	0.53
snake	0.93	0.61	0.75	0.63	0.93	0.49	0.67	0.53
scopa	1.00	0.74	0.84	0.74	1.00	0.64	0.70	0.63
pong	0.90	0.59	0.73	0.59	0.95	0.65	0.71	0.64
Mean	0.96	0.67	0.79	0.68	0.95	0.62	0.68	0.60

B. RQ2: Tracing Performance with Smells

Fig. 1 shows the BTA and the LOC tracing F_1 score achieved in the runs with increasing ratios of smelly requirements. To quantify the relation between the ratio of smelly requirements and the performance, we have fitted a generalized linear mixed model (GLMM) on our data with *game* and *LLM* as random effects. This means the model is fitted to predict the performance (BTA, F_1) based on the percentage of smelly requirements and conditioned on the corresponding game and the used language model. The model for BTA converges and shows a good fit. A summary of the model can be found in our online material[2]. The model coefficient is -0.001, meaning that for every 10% increase in smelly requirements, the BTA decreases by 0.01. A p-value < 0.001 indicates that this effect is statistically significant. The confidence interval $[-0.002, -0.001]$ suggests that this negative impact is consistent across the games and language models. The model for F_1 converged but the model coefficient of -0.001 was not statistically significant ($p = 0.055$).

[1]Mean F_1 may lie outside the range of mean precision and recall due to averaging differences and the harmonic mean's sensitivity to extremes.

[2]https://doi.org/10.6084/m9.figshare.27153441



(a) Binary tracing accuracy (GPT 4o)

(b) LOC tracing F_1-score (GPT 4o)

(c) Binary tracing accuracy (Llama 3.1)

(d) LOC tracing F_1-score (Llama 3.1)

Fig. 1: Results for RQ2: Tracing performance with increasing ratio of smelly requirements.

TABLE IV: Tracing performance for smell categories

Smell category	GPT-4o				Llama 3.1			
	BTA	LOC prec.	LOC rec.	F_1	BTA	LOC prec.	LOC rec.	F_1
lexical	0.90	0.69	0.76	0.69	0.83	0.65	0.68	0.69
syntactic	0.98	0.74	0.84	0.73	0.91	0.76	0.75	0.73
semantic	0.83	0.63	0.76	0.63	0.86	0.64	0.70	0.63

C. RQ3: Tracing Performance for Smell Categories

Table IV shows the mean BTA and the mean LOC tracing precision, recall, and F_1-score achieved for the smelly requirements over all samples of our experiments grouped by the three categories. While syntactic smells (e.g., vague pronouns, passive voice, negative phrases) seem less problematic, the tracing performance for requirements with semantic smells (e.g., inconsistencies, ambiguities) was generally worse. This implies that one should prioritize strategies to avoid these types of smells when using LLMs for trace link generation.

D. Discussion and Conclusion

We conclude that current LLMs show good performance in the analyzed tracing tasks on high-quality requirements. Moreover, they can also cope with low-quality requirements for the investigated task and the limited complexity of the systems we tested. On the other hand, the significant effect of smelly requirements on BTA and the differences in smell types and games suggest that investigating the effect on larger systems and other SE tasks is worthwhile.

V. Future Plans

We aim to study the impact of requirements smells in prompts on software development processes where LLMs are used to generate artifacts. To understand this relationship, we will explore several key research avenues.

Effect of Smells in Requirements-Centric Tasks: We plan to adapt our experiments to other SE tasks, such as code generation, model synthesis, and test case derivation, to examine if and how requirements smells affect these activities. We will assess the correlation between requirements smells and generated artifact smells, such as code or test smells [14], [15]. In addition to quantitative analysis, we will gather qualitative insights to identify emerging problems in generated artifacts. Expert opinions will complement performance results to ensure LLM outputs are measured and explained, following the approach of Ferrari et al. [16].

Effect of Project Scale and Domain: We plan to replicate our experiments on larger, more complex systems. We hypothesize that the effect of smells on downstream tasks will be stronger in complex systems due to the increased difficulty LLMs face handling complex contextual information [17]. Additionally, domain specificity may affect LLM performance, especially if the model lacks domain-specific training [18].

Smell Identification and Correction: To mitigate the impact of smells, prompt issues must be addressed. We will investigate self- and human-assisted correction techniques, such as self-correction by LLMs or clarification queries from the analyst. This will guide the development of practical LLM-based solutions for requirements review, scaling beyond individual requirements to application-level scenarios [19].

Beyond Requirements Smells: Prompts express requirements [20] and, like requirements, may suffer from quality issues. It is unclear how linguistic quality affects LLM performance or how *requirement smells* translate to *prompt smells* [7]. We aim to assess whether 'smelly' prompts impact benchmark tasks (e.g., math reasoning, counterfactual evaluation) or if new *prompt smells* are needed.

REFERENCES

[1] H. Liu, M. Shen, J. Zhu, N. Niu, G. Li, and L. Zhang, "Deep learning based program generation from requirements text: Are we there yet?" *IEEE Transactions on Software Engineering*, vol. 48, no. 4, pp. 1268–1289, 2022.

[2] J. Fischbach, J. Frattini, A. Vogelsang, D. Mendez, M. Unterkalmsteiner, A. Wehrle, P. R. Henao, P. Yousefi, T. Juricic, J. Radduenz, and C. Wiecher, "Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study," *Journal of Systems and Software*, vol. 197, p. 111549, 2023.

[3] C. Arora, T. Herda, and V. Homm, "Generating test scenarios from NL requirements using retrieval-augmented LLMs: An industrial study," in *IEEE 32nd International Requirements Engineering Conference (RE)*, 2024, pp. 240–251.

[4] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Prompts matter: Insights and strategies for prompt engineering in automated software traceability," in *31st IEEE International Requirements Engineering Conference Workshops (REW)*, K. Schneider, F. Dalpiaz, and J. Horkoff, Eds. IEEE, 2023, pp. 455–464.

[5] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, vol. 123, pp. 190–213, 2017.

[6] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained BERT models," in *43rd IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 324–335.

[7] K. Ronanki, B. Cabrero-Daniel, and C. Berger, "Prompt smells: An omen for undesirable generative AI outputs," in *IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (CAIN)*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 286–287.

[8] H. Femmer and A. Vogelsang, "Requirements quality is quality in use," *IEEE Software*, vol. 36, no. 3, pp. 83–91, 2019.

[9] J. Frattini, L. Montgomery, J. Fischbach, M. Unterkalmsteiner, D. Mendez, and D. Fucci, "A live extensible ontology of quality factors for textual requirements," in *30th International Requirements Engineering Conference (RE)*. IEEE, 2022.

[10] L. Montgomery, D. Fucci, A. Bouraffa, L. Scholz, and W. Maalej, "Empirical research on requirements quality: a systematic mapping study," *Requirements Engineering*, vol. 27, no. 2, pp. 183–209, 2022.

[11] J. L. C. Guo, J.-P. Steghöfer, A. Vogelsang, and J. Cleland-Huang, "Natural language processing for requirements traceability," in *Handbook of Natural Language Processing for Requirements Engineering*, A. Ferrari and G. Deshpande, Eds. Cham: Springer International Publishing, 2024.

[12] M. North, A. Atapour-Abarghouei, and N. Bencomo, "Code gradients: Towards automated traceability of LLM-generated code," in *IEEE 32nd International Requirements Engineering Conference (RE)*, vol. 378. IEEE, 2024, pp. 321–329.

[13] K. Peng, L. Ding, Q. Zhong, L. Shen, X. Liu, M. Zhang, Y. Ouyang, and D. Tao, "Towards making the most of ChatGPT for machine translation," in *Findings of the Association for Computational Linguistics (EMNLP)*. Association for Computational Linguistics, 2023.

[14] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia, "Detecting code smells using machine learning techniques: Are we there yet?" in *25th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2018, pp. 612–621.

[15] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli, "On the relation of test smells to software code quality," in *IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2018, pp. 1–12.

[16] A. Ferrari, S. Abualhaija, and C. Arora, "Model generation with LLMs: From requirements to UML sequence diagrams," in *IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, vol. 24. IEEE, 2024, p. 291–300.

[17] F. Shi, X. Chen, K. Misra, N. Scales, D. Dohan, E. H. Chi, N. Schärli, and D. Zhou, "Large language models can be easily distracted by irrelevant context," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 210–31 227.

[18] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, "Don't stop pretraining: Adapt language models to domains and tasks," *arXiv preprint arXiv:2004.10964*, 2020.

[19] F. Mu, L. Shi, S. Wang, Z. Yu, B. Zhang, C. Wang, S. Liu, and Q. Wang, "ClarifyGPT: A framework for enhancing LLM-based code generation via requirements clarification," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 2332–2354, 2024.

[20] A. Vogelsang, "From specifications to prompts: On the future of generative large language models in requirements engineering," *IEEE Software*, vol. 41, no. 5, pp. 9–13, 2024.