ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.798

Students' Confidence and Challenges in Writing Software Requirements: A Preliminary Study

***Abstract*—This paper presents a preliminary study of undergraduate software engineering students' confidence and challenges in software requirements. The study was conducted in an introductory Requirements Engineering course with 86 junior-level students. A survey collected quantitative and qualitative data on students' self-reported confidence and difficulties in requirements engineering tasks. Results show moderate confidence in writing software requirements (mean = 3.58, SD = 0.76) and a strong reliance on examples and templates. The most common challenges reported were ambiguity (34.5%) and incompleteness (23.8%) in requirements specification. The findings highlight the importance of structured scaffolding in requirements engineering education and suggest practical improvements such as more accessible domains, earlier integration of supporting tools, and additional instructional examples to strengthen students' requirements specification skills.**

***Keywords*—software requirements, requirements engineering, requirements analysis, software engineering education, software development life cycle (SDLC).**

I. Introduction

Software requirements engineering forms a foundational yet challenging part of software development. Clear, precise, and well-structured requirements are crucial for successfully guiding the entire software development life cycle (SDLC) and delivering systems that meet expectations [1], [2]. According to industry reports, poorly defined requirements are frequently cited as a major contributor to project challenges leading to expensive rework, scope creep, or systems that do not satisfy user needs [3]. In academic environments, students often struggle to articulate requirements accurately, resulting in ambiguities, incompleteness, and vagueness in their specifications [6], [7]. These shortcomings ripple through the SDLC: during design, students may miss critical entities, actors, or functions; during implementation, code can become disconnected from evolving or unclear requirements; and during testing, incomplete or unverifiable requirements make effective verification and validation difficult [6].

Students commonly face challenges in requirements elicitation due to communication gaps with stakeholders, conflicting needs, and unstated (tacit) knowledge assumptions. They also struggle with specification—particularly avoiding ambiguity while ensuring specificity—and with achieving essential qualities such as completeness, consistency, and traceability [6], [7]. This study investigates students' self-reported confidence levels and perceived difficulties in understanding, analysing, and writing software requirements. By highlighting these gaps, the research seeks to guide the development of improved teaching strategies that better prepare students to produce high-quality requirements, while also strengthening their problem-solving and critical-thinking skills for professional software engineering practice.

The paper is structured as follows: Section 2 presents related studies; Section 3 describes the software requirements course; Section 4 introduces the methodology; Section 5 reports the survey results; Section 6 discusses the findings; Section 7 outlines future work; and Section 8 concludes the paper.

II. Related Work

Requirements engineering (RE) education plays a critical role in preparing students to handle the complexities of software development. Yet, it remains a challenging area due to persistent issues in teaching and learning practical requirements engineering skills. A comprehensive systematic literature review of RE education highlights ongoing student difficulties with key qualities such as ambiguity resolution, specificity, completeness, and consistency in requirements formulation [7], [5]. These challenges often stem from the interdisciplinary nature of RE, requiring technical skills, domain knowledge, and soft skills such as stakeholder communication.

Recent empirical studies have explored innovative approaches to address these gaps, particularly through the integration of large language models (LLMs). For instance, a cross-university research examined 179 students' perceptions of using LLMs in RE coursework, finding that while LLMs enhance engagement, motivation, and practical support for tasks like requirements drafting and analysis, students report concerns about over-reliance on the LLMs and the need for balanced integration with critical thinking [9]. Similarly, educators in Switzerland have identified challenges in higher education RE teaching, emphasizing the need for hands-on practice to build skills in elicitation and specification amid diverse student backgrounds [10].

Broader explorations of LLMs in software engineering education reveal mixed perceptions: students value their utility for productivity and learning but express concerns about effectiveness, detecting AI-generated work, and academic integrity [9], [11], [16]. These findings align with this study's focus on moderate student confidence and widespread recognition of ambiguity and specificity issues, underscoring the potential for targeted instructional enhancements.

While prior work has mapped general RE education landscapes and begun incorporating AI tools, few studies have

directly surveyed students' self-reported confidence in core writing skills. This paper builds on these efforts by providing descriptive insights into perceptions and challenges, aiming to guide pedagogical improvements for better alignment with software development life cycle (SDLC) needs.

III. Software Requirements Course

A. Course Context

This study was conducted within an undergraduate Software Requirements Engineering course offered as part of a Bachelor's degree program in Software Engineering. The course enrolled 86 junior-level students during a 16-week semester. A core component was a semester-long team project introduced in Week 2, centered around developing a software system for a community pharmacy—an unfamiliar domain for most students. Students were organized into teams of 4–5 members and worked collaboratively on the project for the remaining 15 weeks.

The project was structured in five phases, emphasizing progressive refinement of requirements while adhering to established quality attributes for acceptable software requirements: atomic, clear and unambiguous, complete, consistent, correct, feasible, independent, necessary, modifiable, prioritized, testable (verifiable), and traceable [12]

B. Phase 1: Domain Learning and Elicitation (Weeks 2–5)

Students focused on building a foundational understanding of community pharmacy operations, including functions, services, customer types, and relevant regulatory laws - those governed by pharmacy councils and health authorities. Due to limited access to dedicated stakeholders (such as practicing pharmacists or technicians for formal interviews), however, two pharmacy technicians were students in the class. They provided practical information on pharmacy workflows and processes. Overall, students relied primarily on secondary sources:

- Research from official pharmacy council websites and regulatory documents.
- Viewing publicly available video interviews of pharmacists in the USA and UK describing daily operations.
- Examination of pharmacy labels, packaging, and existing pharmacy management software (accessed via free trials).
- Independent field observations, where students were encouraged to visit local pharmacies and supermarkets with pharmacy counters to observe workflows and informally ask questions for clarification.

Each team produced a comprehensive domain report summarizing their findings. This report served as a shared reference document for each team and provided the foundation for subsequent requirements activities.

C. Phase 2: Initial Writing Software Requirements (Weeks 6–9)

Using the domain report, students began writing requirements in "shall" statement format. Teams were instructed to produce at least ten requirements per category, for example, medications, supplies, and insurance, based on the information collected. The primary focus was on achieving atomicity, clarity, unambiguity, consistency, testability, completeness, and correctness [12]. Given the constraints on stakeholder access, students produced only a subset of the total possible requirements for a pharmacy.

D. Phase 3: Requirements Refinement and Categorization (Weeks 10–12)

Students received written instructor feedback and an in-person meeting with each team to discuss their Phase 2 requirements and produced a revised version. In addition to further improving atomicity, clarity, unambiguity, consistency, testability, completeness, and correctness, students were instructed to assign priorities to each requirement and categorize them as functional or non-functional [12].

E. Phase 4: Tool-Assisted Analysis (Week 12)

Students were introduced to the NASA Automated Requirements Measurement (ARM) tool as an objective third-party analyser [13], [14]. The NASA ARM tool evaluates requirements statements linguistically and structurally, identifying issues such as weak phrases, ambiguity, imperfection, and excessive depth in hierarchical numbering, thus ensuring requirements are at the appropriate system/subsystem level. Ongoing peer reviews were also encouraged throughout all phases to reinforce adherence to quality rules and standards.

F. Phase 5: Final Requirements Documentation (Weeks 13–15)

Teams consolidated all requirements into a final report, incorporating corrections from instructor feedback (written and one-on-one team meeting with the instructor), peer reviews, and NASA ARM analysis. Requirements were clearly classified as functional or non-functional, prioritized, and presented in a structured format.

This phased, iterative project design—combined with limited interaction with real stakeholders—provided a realistic yet controlled environment for observing common student difficulties in requirements engineering practice.

IV. Methodology

The participants consisted of 86 junior-level students enrolled in an undergraduate Software Requirements Engineering course as part of a Bachelor's degree program in Software Engineering. The survey was administered at the end of the semester to all 86 enrolled students. It combined Likert-scale questions assessing self-reported confidence in understanding, analyzing, and writing software requirements using a 5-point scale: 1 = Not confident at all, 5 = Very confident, structured multiple-choice questions (including reliance on examples/templates and most challenging aspect of requirements engineering), a multi-select question on aids that help understand requirements, and open-ended questions capturing students' perceived challenges, common mistakes, and suggestions for improvement in writing software requirements. The instrument was designed to gather descriptive data on perceptions rather than to evaluate the objective correctness of student work. Participation was voluntary.

Data was collected using an end-of-semester survey. Although participation was voluntary, 84 students provided usable responses to the key items, yielding a response rate ≈97.7%. Responses were collected non-anonymously (to allow potential follow-up if needed). Still, they were treated confidentially and aggregated for analysis and reporting to protect student privacy and encourage honest self-reflection. The questions did not focus on students' perceptions of the instructor, as this information is collected through the university-administered SPOI survey. Quantitative responses were summarized using descriptive statistics (frequencies, percentages, means, and standard deviations where appropriate). The qualitative responses to open-ended questions were analyzed using thematic analysis [15] to identify recurring patterns, with particular attention to issues such as ambiguity, specificity, completeness, and testability. Percentages show how many respondents mentioned each theme.

V. Results

Students rated their confidence in writing precise software requirements on a five-point Likert scale (1 = Not at all confident, 5 = Very confident). The mean self-reported confidence was 3.58 (SD = 0.76), indicating moderate overall confidence (see **Fig. 1**). Students reported high reliance on examples or templates, with over 76% indicating they use them "Often" or "Always" (see **Fig. 2**). Specification emerged as the most frequently cited difficulty in requirements engineering (see **Fig. 3**). This reflects the demanding nature of specification, which involves translating informal and often ambiguous stakeholder needs into precise and verifiable requirements. Errors at this stage can propagate to later phases of system development.

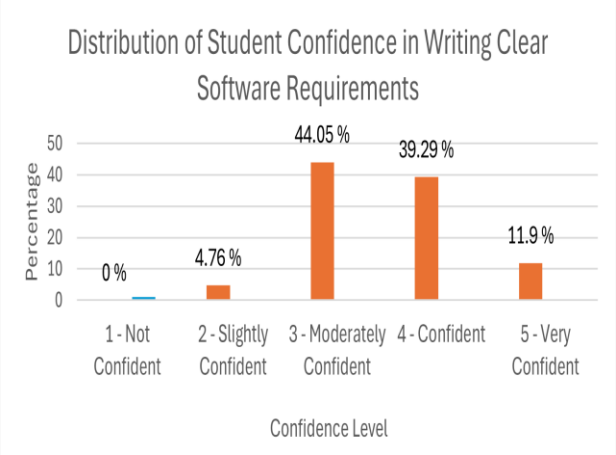

Fig. 1. Distribution of student confidence in writing software requirements.

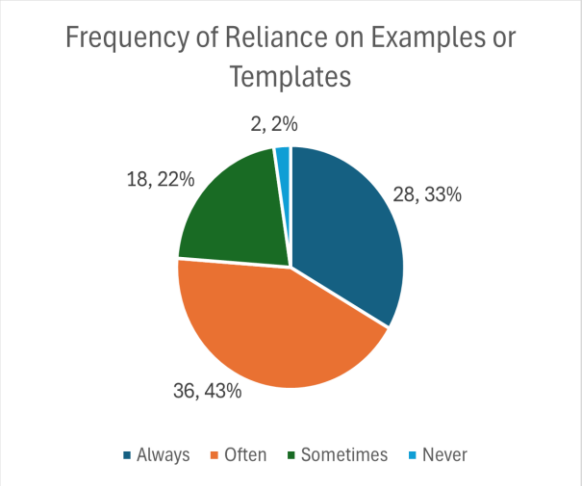

Fig. 2. Frequency of reliance on examples or templates when writing software requirements.

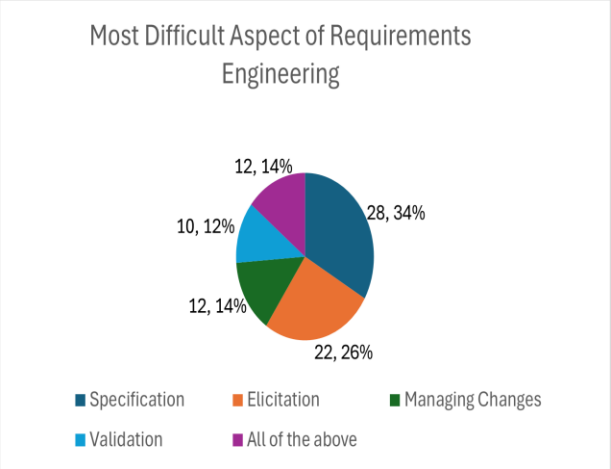

Fig. 3. Most difficult aspect of requirements engineering reported by students.

Responses to the open-ended question about common mistakes and challenges in writing software requirements were analyzed thematically. Ambiguity was the most prevalent challenge, followed by incompleteness and inappropriate mixing of functional and non-functional requirements. Other notable issues included limited domain knowledge, missing stakeholder input, and miscellaneous concerns (see **Fig. 4**). The multi-select question on aids to understanding requirements revealed a strong preference for user stories and diagrams (see **Fig. 5**).

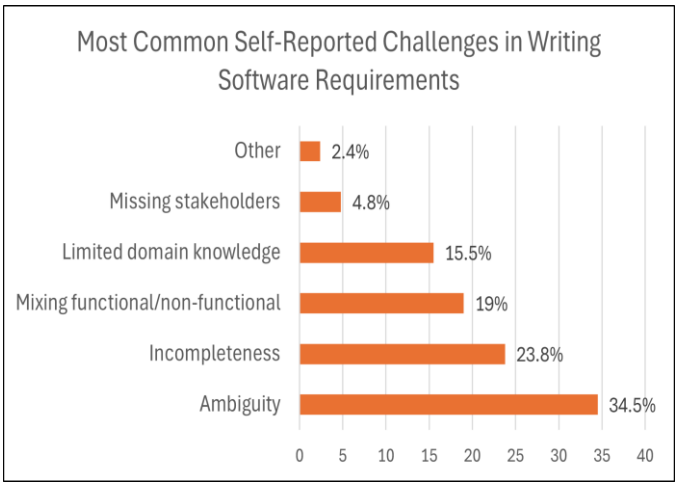

Fig. 4. Most common self-reported challenges in writing software requirements.

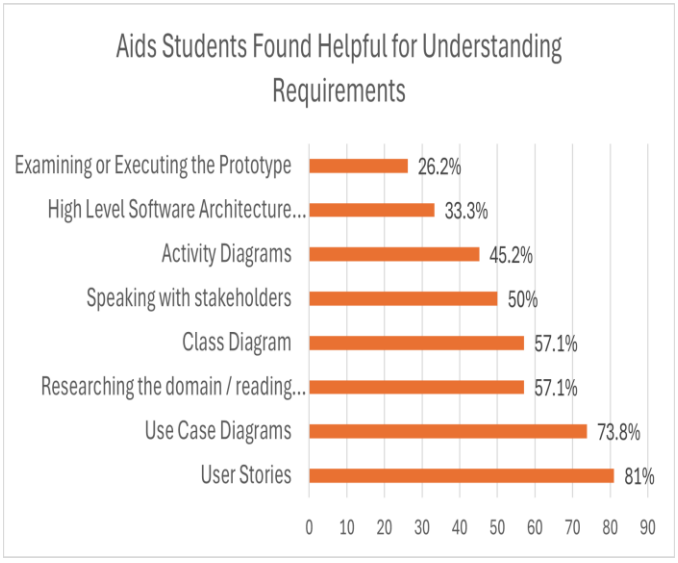

Fig. 5 Aids students found helpful for understanding requirements.

Thematic analysis of open-ended suggestions (≈80 responses) yielded key recommendations: more practice with real-world examples (≈35%), better/more feedback from instructors or stakeholders (≈22%), templates, checklists, or cheat sheets (≈18%), deeper domain knowledge/research (≈15%), and increased real stakeholder interaction (≈10%). These align closely with identified challenges and reliance on examples and templates.

VI. Discussion

The findings from this preliminary study align closely with established and recent literature on requirements engineering (RE) education, reinforcing persistent gaps in student preparation [7]. The most frequently reported challenges—ambiguity (34.5%) and incompleteness (23.8%)—mirror longstanding issues identified in systematic reviews, where students struggle to achieve clarity, specificity, completeness, and consistency in requirements formulation [7], [12]. These difficulties are compounded by the course's limited access to real stakeholders (as described in Section 3), which restricted authentic elicitation practice and contributed to the 15.5% of students citing limited domain knowledge as a primary barrier. Some students were not motivated to go out into the field to pharmacies to ask questions and get the information they needed clarification on and preferred that stakeholders be accessible to them or that stakeholders visit the class to answer questions. This is not feasible because class time is limited, and each team may require more or less one-on-one time with a given stakeholder. The addition of pharmacy visits encouraged real-world elicitation, though unfamiliarity increased challenges—future iterations will use accessible domains like bookstores to balance practicality and focus.

Students' moderate self-reported confidence (mean = 3.58, SD = 0.76; Fig. 1) and high reliance on examples or templates (76% "Often" or "Always"; Fig. 2) suggest an experiential deficit: learners recognize the value of structured guidance and scaffolds but lack sufficient opportunities to develop independent mastery. These findings suggest possible metacognitive awareness, as students identify their own limitations, learning strategies, and challenges in requirements engineering tasks. This is consistent with educator perspectives highlighting the need for more hands-on, practice-oriented approaches in RE teaching [10]. While the NASA ARM tool provided objective feedback in Phase 4, its late introduction may have limited its impact on self-reported confidence. Future studies could explore earlier or iterative use of such tools to determine potential improvements in learning outcomes. The strong preference for user stories (81%) and use case diagrams (73.8%; Fig. 5) further indicates that agile-inspired artifacts resonate with students and may enhance engagement and comprehension—potentially offering a pathway to address specification difficulties, which emerged as the most challenging aspect of RE for 33.3% of respondents (Fig. 3).

The fact that specification is the main difficulty supports earlier findings that students often think they're good at validation but struggle with basic specification [7]. The open-ended suggestions—such as using real-world examples,

soliciting feedback, using templates, and increasing domain exposure—reflect the course constraints in Section 3 and highlight that RE challenges are both social and technical [8].

The results suggest adding more hands-on activities—like stakeholder role-plays, prototype reviews, and industry case studies—to RE courses. Tools such as the NASA ARM and greater use of user stories and diagrams can help connect theory to practice. AI-assisted tools may also support learning by giving scalable feedback on ambiguity and completeness.

VII. Future Work

This study is preliminary and limited to self-reported data from a single course offering in an introductory undergraduate Software Requirements Engineering class, where practical constraints, such as limited resources and stakeholder access, are typical. Future work will address these limitations through the following feasible directions:

- Expanding the sample across multiple semesters of the same course, incorporating pre- and post-course assessments to measure actual learning gains and changes in confidence/perceptions over time.
- Analysing correlations between specific course elements (e.g., the domain report, user stories, NASA ARM tool usage, peer reviews) and student outcomes to identify which activities most effectively improve writing software requirements skills.
- Experimenting with the timing of tool introduction: while the NASA ARM tool provided valuable objective feedback when introduced in Phase 4, future iterations could test earlier exposure, balanced against the risk of over-reliance, ensuring students first develop core critical thinking and requirements engineering skills before using automated analysis.
- Replacing or supplementing the community pharmacy domain with a more accessible, familiar one, such as a hybrid bookstore system modeled after major retailers like Amazon (primarily online) and Barnes & Noble (with both online and in-store operations). This dual-focus domain would leverage students' everyday experiences with online browsing, searching, purchasing, and in-store interactions (e.g., browsing shelves, using in-store pickup, or loyalty programs), potentially reducing barriers related to limited domain knowledge without requiring real stakeholder access, which remains infeasible in this introductory setting. This approach is expected to be particularly effective in addressing the domain-understanding barrier identified by 15.5% of students, as familiar, consumer-facing systems like bookstores typically do not require expert stakeholder input to elicit meaningful requirements. Students can draw on personal experience and publicly available information, enabling more confident and accurate requirements formulation while maintaining the educational focus on core RE skills. To further support domain understanding and requirements elicitation, students will be provided with additional scaffolding materials, including a set of pre-written user stories covering various bookstore scenarios (e.g., searching for books online or in-store, managing shopping carts, handling payments and returns, applying discounts, and reserving items for in-store pickup) and a sample use case diagram illustrating key actors (e.g., online customer, in-store customer, staff) and interactions across both channels. These resources will give students concrete examples to reference as they elicit, analyse, and refine their own requirements.

These enhancements aim to strengthen the evidence for effective, scalable teaching strategies in resource-limited RE courses, with a focus on iterative improvement and skill-building within the existing course structure.

VIII. Conclusion

This preliminary study shows that undergraduate students in a required Requirements Engineering course report moderate confidence in writing software requirements but continue to face significant challenges, particularly with ambiguity and incompleteness. The findings highlight the importance of domain familiarity, structured scaffolding, and practical support tools in improving requirements specification. These insights provide a foundation for iterative improvement of course design and better preparation of students for professional software development practice.

References

[1] I. Sommerville, *Software Engineering*, 9th ed. Boston, MA, USA: Pearson, 2011.

[2] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, Version 3.0. Los Alamitos, CA, USA: IEEE Computer Society, 2014.

[3] The Standish Group, *CHAOS Report*. Boston, MA, USA: The Standish Group International, 2020.

[4] C. Ghezzi and D. Mandrioli, "The challenges of software engineering education," in *Software Engineering Education in the Modern Age*, S. B. Seidman and P. J. Burton, Eds. Berlin, Germany: Springer, 2006, pp. 115–127.

[5] Y. Sedelmaier and D. Landes, "Software engineering body of knowledge with a view to education," *Requirements Engineering*, vol. 19, no. 4, pp. 1–20, Nov. 2014.

[6] A. Davis *et al.*, "Effectiveness of requirements elicitation techniques: An experiment," *Requirements Engineering Journal*, vol. 11, no. 3, pp. 189–202, Aug. 2006, doi: 10.1109/RE.2006.17.

[7] M. Daun, A. M. Grubb, V. Stenkova, and B. Tenbergen, "A systematic literature review of requirements engineering education," *Requirements Engineering*, vol. 28, no. 2, pp. 145–170, May 2023.

[8] M. G. Christel and K. C. Kang, *Issues in Requirements Elicitation*, Carnegie Mellon Univ., Pittsburgh, PA, USA, CMU/SEI-92-TR-012, 1992.

[9] M. Bernabei *et al.*, "Students' use of large language models in engineering education: A case study on technology acceptance,

perceptions, efficacy, and detection chances," *Computers & Education: Artificial Intelligence*, vol. 5, Art. no. 100172, 2023.
[10]A. Moravánszky, "Challenges of requirements engineering education in higher education: Insights from an interview study with educators in Switzerland," in *Proc. Int. Working Conf. Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2025.
[11]S. Rasnayaka *et al.*, "An empirical study on usage and perceptions of LLMs in a software engineering project," in *Proc. ICSE Workshop on LLMs for Code (ICSE@LLM4Code)*, 2024.
[12]P. A. Laplante and M. Kassab, *Requirements Engineering for Software and Systems*, 4th ed. Boca Raton, FL, USA: CRC Press, 2023.
[13]N. Carlson and P. A. Laplante, "The NASA automated requirements measurement tool: A reconstruction," *Innovations in Systems and Software Engineering*, vol. 10, no. 2, pp. 77–91, June 2013, doi: 10.1007/s11334-013-0225-8.
[14]W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated analysis of requirement specifications," in *Proc. 19th Int. Conf. Software Engineering (ICSE '97)*, Boston, MA, USA, May 1997, pp. 161–171.
[15]V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006, doi: 10.1191/1478088706qp063oa.
[16]S. Guardado *et al.*, "Students' perceptions of the use of LLMs in requirements engineering education: A cross-university empirical study," *arXiv* preprint arXiv:2509.05995, 2025.