# The Role of Acceptance Criteria and Developer Expertise in Enhancing the Quality of Robustness Diagrams in Agile Software Development

Francisco Antonio Mejía-Domínguez, Manuel A. Quintana, Ramón R. Palacio, Gilberto Borrego, and Samuel González-López

***Abstract*—In agile software development, user stories are a common tool for capturing functional requirements, valued for their simplicity and user-centric approach. Nevertheless, their inherent informality can introduce ambiguity, especially when acceptance criteria are lacking or poorly defined. Such ambiguity may hinder early design activities, including the development of UML robustness diagrams. This study investigates the impact of acceptance criteria and developer experience on the accuracy and efficiency of robustness diagram construction. A controlled experiment was conducted with thirty participants, divided into newbie and advanced developers, who were tasked with creating robustness diagrams from user stories both with and without acceptance criteria. Performance was assessed based on task duration and diagram completeness. Statistical analysis (Mann-Whitney U test) revealed that acceptance criteria significantly reduce errors and improve completion times, particularly among newbie developers. Experienced developers consistently produced higher-quality diagrams more efficiently, underscoring the role of expertise as a moderating factor. The findings suggest that well-specified acceptance criteria mitigate ambiguity, facilitating more accurate requirements interpretation and improving early design outcomes. Additionally, the results highlight the value of structured requirement practices in agile methodologies, especially in teams with varying levels of experience. This research advances the understanding of how requirement clarity and developer expertise collectively influence software modeling, providing actionable recommendations for enhancing agile design processes.**

**Link to graphical and video abstracts, and to code:**
**https://latamt.ieeer9.org/index.php/transactions/article/view/9950**

***Index Terms*—user stories, acceptance criteria, robustness diagram, agile, UML**

## I. Introduction

MANY software development methodologies, such as the Rational Unified Process [1] and ICONIX [2], include a phase dedicated to analyzing user requirements or use cases to bridge the gap with detailed design. In this phase, UML diagrams are commonly used to represent the structural and dynamic views of the system. Among these, robustness diagrams provide a visual representation that validates the structural integrity of software systems by illustrating interactions between analysis objects in response to user actions [3]. Consequently, the quality of these diagrams depends heavily on how accurately user requirements are captured and communicated during development. While using robustness diagrams as a case study, the importance of clear acceptance criteria extends to all early design artifacts in agile development.

In agile environments, user stories are widely used for capturing functional requirements due to their simplicity and user-centric focus. However, their brevity often introduces ambiguity, especially when acceptance criteria are unclear or omitted. Lucassen et al. [4] argue that the lack of explicit criteria can lead to misunderstandings and incomplete implementations, while Mahnič [5] notes that many agile teams prioritize speed over well-defined criteria, which can compromise quality. Reports such as VersionOne's State of Agile [6] reveal that only 58% of teams consistently include acceptance criteria,highlighting a gap between theory and practice. In complex systems, these criteria serve as a critical "contract" that aligns expectations and validates outcomes[7].

A well-structured user story typically includes the user's role, the intended action, and the expected outcome, supported by explicit acceptance criteria to confirm completeness and correctness [8]. Such criteria provide testable conditions that guide development and reduce ambiguity, facilitating the accurate translation of user stories into robustness diagrams [6]. Robustness diagrams, in turn, serve as a bridge between class and activity diagrams, modeling system behavior under varying workloads and validating business logic [1], [9], [2]. They also enhance communication between stakeholders and help detect inconsistencies early.

Prior research highlights the importance of developer experience in design tasks. Borrego et al. [10] show that advanced developers develop an intuitive understanding of architectural decisions, compensating for missing documentation. This expertise results in deeper comprehension of requirements and constraints, more efficient knowledge transfer, improved maintainability, and reduced time spent seeking external clarifications [10], [11]. These advantages contribute to more efficient and cost-effective development processes [12]. Moreover, experienced developers can interpret and apply acceptance criteria more effectively, inferring missing details and ensuring consistency in diagrams [7], [13]. Conversely, less experienced

The associate editor coordinating the review of this manuscript and approving it for publication was Carlos Thomaz (*Corresponding author: Francisco Antonio Mejía Domínguez*).

Francisco Antonio Mejía-Domínguez, M. A. Quintana, R. R. Palacio, and G. Borrego are with Sonora Institute Of Technology, Navojoa, Mexico (e-mails: francisco.mejia216677@potros.itson.edu.mx, manuel.quintana133313@potros.itson.edu.mx, ramon.palacio@itson.edu.mx, and gilberto.borrego@itson.edu.mx).

S. González-López is with Technological Institute of Nogales, Nogales, Mexico (e-mail: samuel.gl@nogales.tecnm.mx).

developers rely more heavily on explicit guidance, which helps reduce errors and time spent resolving ambiguities.

Despite the recognized benefits of acceptance criteria, empirical evidence on their effect in robustness diagram creation is still limited. Low completeness during early stages can lead to costly rework [14]. Experience also moderates this effect: advanced developers, with their architectural knowledge and problem-solving skills, often require less detailed criteria [4], while novices need precise guidance to avoid misunderstandings [15].

This study empirically examines the influence of acceptance criteria on the quality of robustness diagrams, considering developer experience as a moderating factor. The analysis focuses on two dimensions: (1) diagram completeness and (2) the time required for their construction. By applying statistical analysis to these factors, this research aims to provide actionable insights to improve the integration of user stories and acceptance criteria in agile software design.

The rest of this article is structured as follows. Section 2 details the study description and key concepts. Section 3 presents the method, including experimental design, participant grouping, and instrumentation. Section 4 discusses results and findings on the effect of acceptance criteria and developer experience on robustness diagram quality. Section 5 addresses threats to validity, and Section 6 concludes with key contributions and directions for future work.

## II. Study Description

In this study, we investigate the impact of acceptance criteria in user stories on the completeness and the time required to complete robustness diagrams. We also examine how the varying levels of detail in user stories influence these outcomes.

### A. Objective

Empirically evaluate the effect of including acceptance criteria in user stories on the quality and efficiency of UML robustness diagram creation by measuring task completion time and diagram completeness and analyzing how developer experience moderates these outcomes.

## III. Method

### A. Scoping

This study empirically evaluates the impact of two factors on the construction of robustness diagrams: (i) the presence of acceptance criteria in user stories and (ii) the level of developer experience. Specifically, it focuses on two key outcome dimensions—task completion time and diagram completeness.

To ensure consistency across experimental conditions and enable precise measurement of variables, a web-based tool named ROBIX was used [16]. This platform facilitated standardized task presentation, automatic tracking of user interactions, and accurate recording of completion times, enabling detailed statistical analysis.

### B. Planning

*1) Context Selection:* The experiment was conducted in a controlled academic environment. Participants were grouped according to their level of experience in software development. The advanced group consisted of professionals currently working in the industry, while the newbie group was composed of students in the final year of a Software Engineering undergraduate program.

*2) Selection of Subjects:* Participants were selected and divided into two equally sized groups:

- **Advanced Group**: 15 participants with at least two years of professional experience working on web platforms, primarily using PHP and related technologies. These participants had experience in software design, coding, and UML diagramming.
- **Newbie Group**: 15 participants who were undergraduate students currently undertaking professional internships. They had received basic training in software development but lacked extensive practical experience. All participants had basic training in the fundamentals of UML and robustness diagrams prior to the experiment.

We refer to participants as developers because their main roles involve software development tasks like coding, debugging, and implementation. Although robustness diagramming is design-oriented, participants particularly those in the advanced group had UML and modeling experience, allowing them to perform the tasks without being formal software designers[17].

*3) Study Design:* This study follows a within-subjects design, in which each participant completed three tasks consisting of building robustness diagrams based on user stories, some with acceptance criteria and others without. Participants were randomly assigned to mixed groups, rather than being separated by experience level, to reduce potential group bias and better generalize the findings across varying skill levels.

Each participant accessed the ROBIX web tool remotely and received detailed instructions through a virtual meeting conducted via Google Meet. They were then provided with a set of three user stories in a fixed sequence, accompanied by login credentials and a link to the ROBIX platform. Participants were instructed to complete the tasks in the given order.

The entire experiment was completed in a single session lasting between 20 and 25 minutes. The order of tasks and the format of the user stories were standardized across all participants to ensure consistency. This standardization minimized potential biases related to task sequence or content variation, allowing observed performance differences to be attributed to the presence or absence of acceptance criteria.

*4) Variables Selection:* The variables selected in this study are not only theoretically relevant, but also highly applicable to real-world software development contexts. Their evaluation contributes to a deeper understanding of the interaction between human factors and process artifacts in agile environments. Therefore, these variables are significant for understanding how the clarity of requirements and the experience of developers influence software modeling in the early

stages of a software project. Moreover, to evaluate quality, we focus on the completeness of the produced robustness diagrams, defined as the inclusion of all essential components for accurately representing the system. Understanding how acceptance criteria and developer experience influence these outcomes can have direct implications for design quality [18]. A specification is considered complete when no information is left unstated, there are no undefined entities, and no critical details are missing [18]. Incomplete specifications can lead to uncertainty in estimation, miscommunication, and integration issues [19], which is why completeness is a key metric in assessing model quality.

**Independent Variables**:

- *Experience level* (Advanced vs. Newbie) : Developer experience plays a key role in interpreting requirements and constructing design artifacts. In this study, participants were classified based on self-reported professional experience: those with more than two years of experience in software development were categorized as advanced, while those with two years or less were considered newbies. According to Borrego et al. [10], advanced developers tend to develop an intuitive understanding of architectural decisions, which helps them maintain consistency and completeness even in the absence of detailed documentation. Their expertise enables more efficient knowledge transfer and reduced time spent resolving uncertainties during modeling tasks [13].
- *Presence of acceptance criteria* (No Criteria vs. With Criteria) : Acceptance criteria are fundamental in clarifying user requirements, especially in agile contexts where minimal documentation is preferred [**?**]. Well-defined criteria help both newbie and advanced developers reduce ambiguity, validate assumptions, and ensure alignment with expected system behaviors [18]. For less advanced developers, acceptance criteria offer critical scaffolding to prevent errors and speed up task completion [4].

**Dependent Variables**:

- *Completion time* (in seconds): This variable captures the duration required by each participant to construct a robustness diagram based on a given user story. The timing began when the participant first dragged an element onto the diagram canvas within the ROBIX web application and ended when the participant clicked the save button to store the completed diagram. This approach ensured consistent and automated measurement of the actual modeling activity duration for each task.
- *Completeness* Score of Robustness Diagrams , evaluated manually using a predefined checklist of expected elements. This variable captures the quality and thoroughness of the produced diagrams, particularly in the identification of the main objects of this kind of diagram: boundary, control, and entity elements.

The rationale for selecting these variables is supported by findings that suggest advanced developers can leverage even minimal documentation to produce high-quality artifacts [10], [**?**]. Moreover, acceptance criteria contribute to reducing cognitive load and guiding modeling activities, thereby improving both performance and outcome [11].

*5) Hypotheses Formulation:* The following null hypotheses were defined to statistically analyze the influence of the studied variables:

- $\mathbf{H_{0TimeExperience}}$: There is no significant difference in the time required to complete a robustness diagram between advanced and newbie developers.
- $\mathbf{H_{0TimeCriteria}}$: There is no significant difference in the time required to complete robustness diagrams with acceptance criteria and without them.
- $\mathbf{H_{0CompletenessCriteria}}$: There is no significant difference in the completeness of robustness diagrams created with and without acceptance criteria.
- $\mathbf{H_{0CompletenessExperience}}$: There is no significant difference in the completeness of robustness diagrams created by advanced and newbie developers.

*6) Instrumentation:* To carry out the experiment, the following instruments and resources were used:

- **ROBIX web application**: A diagramming tool [16] installed on a private server, customized to capture interaction logs and completion times per task.
- **Training Material**: Documentation on how to use ROBIX and build robustness diagrams.
- **User Stories**: The selection of the three user stories was based on functional representativeness, controlled complexity, with different numbers of elements across diagrams, which led us to measure completeness as the raw count of correctly included elements, and balanced assignment of acceptance criteria. Each story reflects a common functionality in web applications (user authentication, order processing, and information search), ensuring relevance to real-world scenarios. The variation in acceptance criteria (0, 3, and 7) was designed to simulate different levels of requirement clarity, ranging from completely informal descriptions to well-structured specifications. Prior pilot testing ensured that no single story was significantly more complex than the others, maintaining experimental balance. This approach aligns with best practices in empirical software engineering research for designing valid and replicable tasks [10].

### C. Operation

The overall procedure followed in the experiment is illustrated in Fig. 1. To complement this visual outline, the key steps are described in detail below to highlight how participants engaged with the task of creating robustness diagrams from different user stories.

*1) Introduction and Training:* Participants received credentials to access ROBIX, along with detailed instructions and the training material described above. This material included documentation on how to use the ROBIX web application and guidelines for constructing robustness diagrams. Additionally, a 10-minute introductory explanation was provided at the beginning of the session. This session covered the fundamental concepts of robustness diagrams, illustrated examples of correct and incorrect diagrams, and clarified common modeling mistakes.

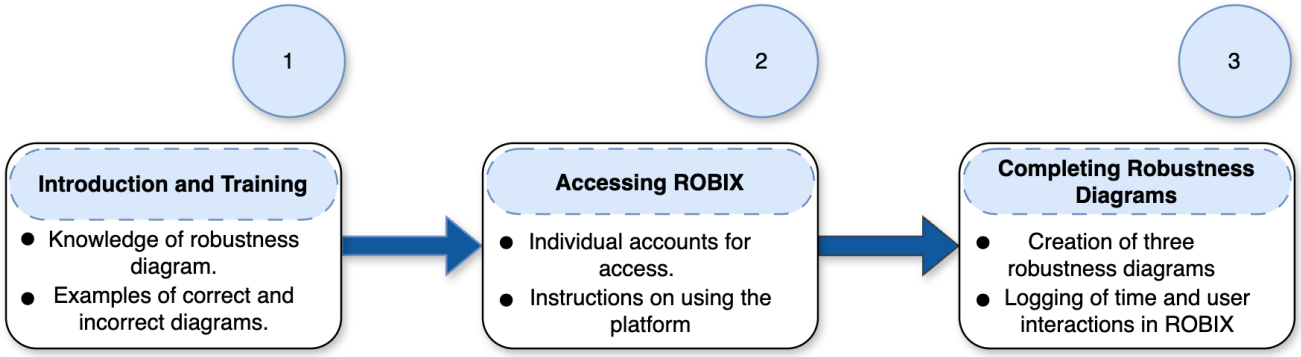


Fig. 1. Outline of the experiment procedure. The process consists of three stages: (1) introduction and training with examples of robustness diagrams, (2) accessing the ROBIX platform through individual accounts, and (3) completing three robustness diagrams while recording time and user interactions.

To ensure accurate data collection, the ROBIX application was also modified to incorporate features that enabled precise tracking of task completion time and user interactions.

*2) Accessing ROBIX:* Participants used the ROBIX web application to create robustness diagrams. The experiment was carried out remotely through Google Meet. At the beginning of the session, each participant received a set of three user stories presented in a predefined order—from the first to the third—with clear instructions indicating that they should address them sequentially. The instructions also included the link to access the ROBIX tool and a unique account for each participant to log in to the platform.

*3) Completing Robustness Diagrams:* Once they had read the instructions, participants proceeded to log into ROBIX using their assigned credentials. From there, they began constructing the corresponding robustness diagrams for each user story, starting with the first one. The environment was configured so that each participant worked independently, while the researchers monitored the session in real time via Google Meet to provide support in case of technical issues or clarifications.

The experiment was carried out over two days. On a given day, a group could consist of seven advanced developers and eight newbie developers, or a different combination. The arrival of each participant was scheduled according to their individual availability. Each session lasted approximately 20 minutes, during which all participants completed the same set of exercises.

### D. Data Analysis

The collected data was processed using MATLAB. For each participant, a CSV file generated by ROBIX[1] was retrieved. These files included the recorded timestamps, interaction logs, and task durations. The data provided insight into the participants' behavior during the execution of the tasks.

Before conducting hypothesis tests, the normality of the data distributions was assessed using the Shapiro-Wilk test. Results indicated that the assumption of normality was violated for multiple variables, especially within the smaller subgroups. Therefore, the Mann-Whitney U test, a non-parametric alternative, was applied to evaluate the statistical significance of differences between groups. This test is particularly suitable for comparing independent groups when the data are not normally distributed and the sample size is limited, as was the case for this study.

Additionally, various visualizations such as boxplots and bar graphs were created to support the statistical findings and highlight trends in the data:

The time by experience level and by acceptance criteria. The completeness scores by experience level and by acceptance criteria. To ensure the reliability of the results, the logged interaction data were cross-referenced with the automatically recorded task durations and completeness scores. This triangulation helped validate the participants' actual performance and supported the interpretation of the statistical outcomes.

Results were analyzed to assess the validity of each null hypothesis, focusing on how acceptance criteria and developer experience influenced robustness diagram quality and task efficiency. Statistical significance was determined using a standard threshold of $\alpha = 0.05$.

## IV. Results and Discussion

This section presents the results of the hypothesis tests regarding the impact of acceptance criteria and developer experience on the time and completeness of robustness diagrams. Along with statistical significance, we discuss the practical implications and emerging patterns observed in developer performance.

### A. Time Comparison Between Newbie and Advanced Developers ($H0_{TimeExperience}$)

Across all conditions, newbie developers exhibited a longer average completion time ($\overline{x} = 432.02$ seconds) than advanced developers ($\overline{x} = 398.45$). The variability was also greater for newbies ($\sigma = 49.43$) compared to the advanced group ($\sigma = 43.32$), indicating a less consistent performance. This pattern reflects the typical learning curve observed in tasks involving modeling, especially when using structured notations such as robustness diagrams.

[1]https://github.com/ITSONRobustnessDiagram/acceptance-criteria-dataset

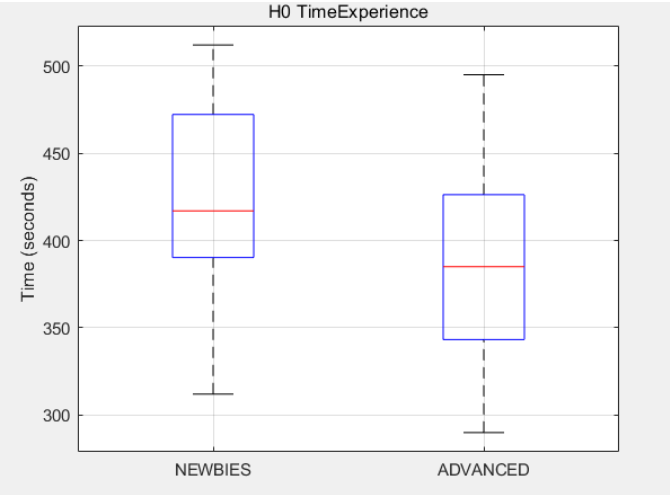


Fig. 2. Comparison of time (in seconds) to create robustness diagrams based on developer expertise. Newbies tend to require more time, showing a higher median and greater variability, while advanced developers complete the task faster on average with lower dispersion.

The Mann-Whitney U test yielded a statistically significant result ($U = 2434.50, p = 0.00181$), indicating that experience level does indeed impact completion time. With a cliff-delta $\delta = 0.3822$ which means Newbies takes more time than Advanced developers.

Fig. 2 illustrates this effect clearly. The boxplot shows that newbie developers both with and without acceptance criteria tend to have higher median times and wider interquartile ranges. In contrast, advanced developers complete the task faster and with more consistency.

Given that the p-value is lower than the 0.05 threshold, $\text{H0}_{\text{TimeExperience}}$ is rejected. Developer experience has a statistically significant effect on completion time, with advanced developers completing robustness diagrams faster and with less variability.

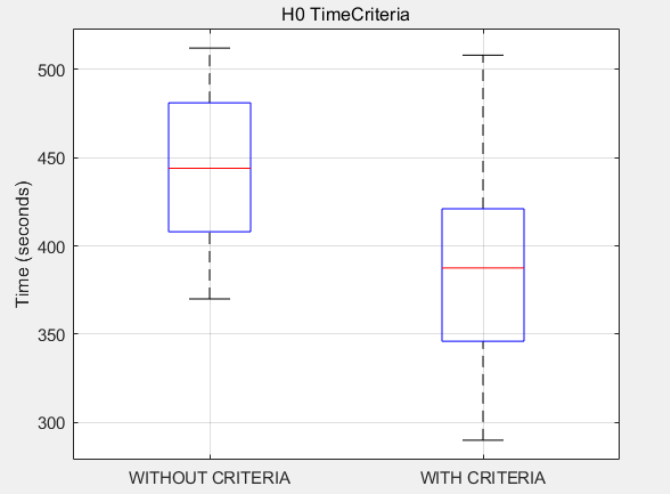


Fig. 3. Comparison of time (in seconds) to create robustness diagrams with and without acceptance criteria, indicating longer times without criteria and shorter, more consistent times with criteria.

### *B. Time Comparison Between Developers With and Without Acceptance Criteria ($H0_{TimeCriteria}$)*

Participants who received acceptance criteria showed reduced completion times ($\overline{x} = 387.97$, $\sigma = 50.64$), compared to those without ($\overline{x} = 442.50$ seconds, $\sigma = 42.47$). The reduction was observed in both experience groups, though it was more pronounced among newbies developers.

The Mann-Whitney U test confirmed the significance of this difference ($U = 1866.50, p = 0.00002$). This result confirms the theory that acceptance criteria function as a cognitive scaffold, reducing ambiguity and effectively guiding developers during the modeling process. And a cliff delta $\delta = 0.7101$ which indicates User Stories with Acceptance Criteria have shorter creation time for the diagrams compared to those without.

This trend is illustrated in Fig. 3, where both groups (newbies and advanced) show lower median completion times when acceptance criteria are included. The spread of the data is also narrower, particularly among advanced developers.

Since the p-value falls well below 0.05, $\text{H0}_{\text{TimeCriteria}}$ is rejected. The presence of acceptance criteria significantly reduces the time required to complete robustness diagrams, confirming their utility as a supportive design artifact.

### *C. Completeness Comparison Between Newbie and Advanced Developers ($H0_{CompletenessExperience}$)*

The third null hypothesis, $\text{H0}_{\text{CompletenessExperience}}$, asserts that diagram completeness does not significantly differ between newbie and advanced developers.

The average incompleteness score among newbies was $\overline{x} = 7.05$ ($\sigma = 2.32$), compared to $\overline{x} = 3.35$ ($\sigma = 1.46$) among advanced developers. This indicates that advanced developers tend to include more expected elements (e.g., boundary, control, and entity objects), achieving more complete and structured diagrams.

The Mann-Whitney U test revealed a statistically significant difference between groups ($U = 2766.50, p < 0.00001$). While cliff delta $\delta = 0.5572$ where newbies have more errors than advanced developers.

Fig. 4 confirms this pattern. It shows that newbie developers frequently omit components, leading to higher incompleteness scores and more dispersed values. Advanced developers show lower medians and more concentrated distributions.

Since the p-value is considerably lower than 0.05, $\text{H0}_{\text{CompletenessExperience}}$ is rejected. The completeness of robustness diagrams is significantly influenced by the developer's level of experience.

### *D. Completeness Comparison Between Developers With and Without Acceptance Criteria ($H0_{CompletenessCriteria}$)*

The final null hypothesis, $\text{H0}_{\text{CompletenessCriteria}}$, posits that the inclusion of acceptance criteria does not significantly impact the completeness of robustness diagrams.

Developers using acceptance criteria achieved an average incompleteness score of ($\overline{x} = 3.13$, $\sigma = 1.94$), in contrast to ($\overline{x} = 6.77$, $\sigma = 2.01$) for those without. This reduction

TABLE I
P-VALUES FOR EACH NULL HYPOTHESIS

| **Null Hypothesis** | **p-value** |
|---|---|
| $H0_{TimeExperience}$: No significant difference in completion time between advanced and newbie developers. | 0.00181 |
| $H0_{TimeCriteria}$: No significant difference in completion time between tasks with and without acceptance criteria. | 0.00002 |
| $H0_{CompletenessExperience}$: No significant difference in completeness between advanced and newbie developers. | 0.00001 |
| $H0_{CompletenessCriteria}$: No significant difference in completeness between tasks with and without acceptance criteria. | 0.00001 |

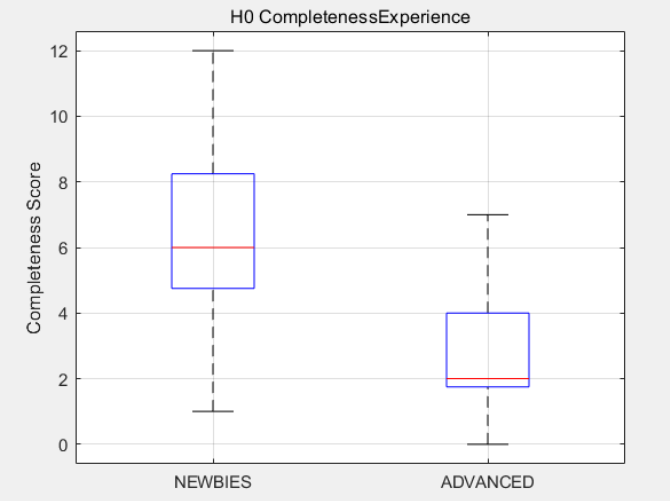


Fig. 4. Completeness score distribution in robustness diagrams based on developer expertise, with higher median values for newbies and lower but more consistent scores for advanced developers.

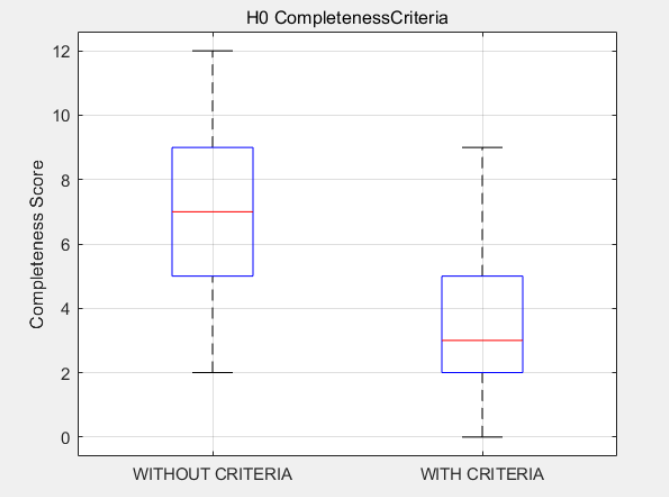


Fig. 5. Completeness score distribution in robustness diagrams with and without acceptance criteria, showing higher median values without criteria and lower but more consistent values with criteria.

in omitted elements was consistent across both experience levels, indicating that acceptance criteria provide a structured reference that supports completeness.

Statistical testing via the Mann-Whitney U test yielded $U = 1918.00, p < 0.00001$, confirming the difference as statistically significant. Cliff delta $\delta = 0.6144$ that means without criteria has higher completeness than with criteria.

Fig. 5 visually supports this conclusion. The groups with acceptance criteria consistently show lower median incompleteness scores and tighter distributions, indicating that criteria help developers avoid missing critical components.

Given that the p-value is below 0.05, $H0_{CompletenessCriteria}$ is rejected. Acceptance criteria significantly improve the completeness of robustness diagrams by helping developers identify and represent all required elements more consistently. In Table I all the $p$ values are summarized.

### *E. Practical Implications*

The findings of this study provide actionable insights for improving software development practices:

1) **Completeness Improvement**: Acceptance criteria significantly enhance the completeness of robustness diagrams by offering explicit guidance. This helps developers—especially early in the design phase—identify and include essential elements that might otherwise be overlooked, resulting in more accurate system representations. Incorporating well-defined acceptance criteria into requirements documentation improves model quality, reduces ambiguity, and strengthens the foundation for downstream development tasks.
2) **Efficiency Gains**: The clarity provided by acceptance criteria reduces cognitive effort and accelerates modeling tasks. In projects involving numerous user stories, this efficiency can lead to substantial time savings. Systematically including acceptance criteria streamlines the design process, improves predictability, and supports faster delivery—particularly valuable in agile environments where speed and adaptability are key.
3) **Onboarding Support**: Understanding performance differences in the diagram design between novice and experienced developers enables better onboarding strategies. Providing structured templates, examples, or training based on empirical insights can shorten ramp-up times and help new developers produce higher-quality models from the start.
4) **Enhanced Collaboration**: Awareness that both expertise and guidance influence outcomes encourages teams to develop shared conventions and knowledge bases. This alignment fosters clearer communication, minimizes mis-

understandings during early design phases, and facilitates smoother transitions from analysis to implementation—especially within cross-functional agile teams.

This study addresses a critical gap in the literature on early software design phases in agile environments: how requirement clarity (represented by the inclusion of acceptance criteria) and developer experience affect the quality and efficiency of robustness diagram creation.

The results confirm that including acceptance criteria has a significant impact on both completeness and execution time of robustness diagrams, particularly for novice developers. These findings support existing theoretical recommendations [5], but provide empirical validation of their practical application in real-world modeling contexts. Acceptance criteria act as cognitive scaffolding that reduces ambiguity and enhances understanding of the functional scope of each user story, which is essential for producing coherent and complete UML models [**?**].

## V. Threats to Validity

To ensure the reliability and generalizability of our findings, we analyzed potential threats to validity following Wohlin et al.'s classification [20], covering internal, external, construct, and conclusion validity. We also applied mitigation strategies such as controlled settings, standardized task design with ROBIX, pre-testing, and statistical validation.

1) **Internal Validity** The remote setup limited control over participants' environment, attention, and motivation, introducing potential biases affecting execution time and diagram quality. External assistance (e.g., peers or online resources) could not be ruled out. Performing tasks in a fixed order risked learning effects, especially for novices, where later tasks may have benefited from practice rather than treatment. Although stories were selected for similar complexity, a counterbalancing design was not applied. Additionally, the short training session (10 minutes) may have caused interpretation bias despite providing examples of correct and incorrect diagrams.
2) **External Validity** Participants were recruited from Mexican academic and industrial contexts, limiting cultural and methodological diversity and restricting generalizability to distributed teams or organizations with mature agile cultures. Focusing exclusively on web applications also limits applicability to other domains. Replications with more diverse teams and environments are recommended.
3) **Construct Validity** Diagram completeness was evaluated with a checklist based on UML and ICONIX guidelines. While structured, it introduces subjectivity since a diagram could meet checklist criteria but still be conceptually incorrect. Task completion time, captured automatically by ROBIX, improved accuracy, but may reflect factors unrelated to modeling skill, such as tool familiarity or internet speed, introducing a technological bias.
4) **Conclusion Validity** Non-parametric tests (Mann-Whitney U) suited the data and sample size, but the small sample reduced statistical power, limiting detection of smaller effects. The between-subjects design, while avoiding fatigue or learning, increased variance and reduced sensitivity. Future studies should explore within-subject or mixed designs and larger samples to improve robustness and practical applicability.

## VI. Conclusion

This study provides empirical evidence of the positive impact of acceptance criteria and developer expertise on the quality and efficiency of robustness diagrams. Integrating well-defined criteria into user stories reduces ambiguity, improves completeness, and strengthens communication within agile teams, offering clear benefits for novice developers.

Developers using acceptance criteria completed tasks faster and produced fewer errors, while experienced developers achieved greater consistency and efficiency, highlighting the value of combining expertise with structured guidance.

From a practical perspective, agile teams should adopt acceptance criteria as a collaborative practice. Before a story enters a sprint, it should include clear, verifiable criteria agreed upon by all stakeholders to ensure shared understanding and reduce rework. Integrating these criteria into daily workflows and tools such as JIRA or Azure DevOps, with simple templates to guide modeling tasks, enhances their effectiveness.

While this study primarily focused on completeness as a quality measure, future research should explore other quality dimensions and validate these findings with larger, more diverse samples to deepen understanding of how acceptance criteria and expertise jointly enhance software design.

## References

[1] R. Klimek and P. Szwed, "Formal analysis of use case diagrams," *Computer Science*, vol. 11, p. 115, 2013.
[2] J. A. R. Castillo and J. A. Malinao, "Model decomposition of robustness diagram with loop and time controls to petri net with considerations on resets," in *ITS 2024*, pp. 193–201, Springer, 2024.
[3] N. Seyff, F. Graf, and N. A. M. Maiden, "End-user requirements blogging with irequire," in *2010 18th IEEE International Requirements Engineering Conference*, 2010.
[4] G. Lucassen, F. Dalpiaz, and J. M. van der Werf, "Improving agile requirements: The case for user stories," *Requirements Engineering*, vol. 21, no. 4, pp. 447–465, 2016.
[5] V. Mahnič and T. Hovelja, "On using planning poker for estimating user stories," *Journal of Systems and Software*, vol. 85, no. 9, pp. 2086–2095, 2012.
[6] R. Hoda, N. Salleh, and J. Grundy, "The rise and evolution of agile software development," *IEEE Software*, vol. 35, no. 5, pp. 58–63, 2018.
[7] S. Fatima, U. Maqbool, and G. Rasool, "Improving software requirements reasoning by novices: a story-based approach," *IET Software*, vol. 13, no. 6, pp. 546–556, 2019.
[8] F. Gilson, M. Galster, and F. Georis, "Generating use case scenarios from user stories," in *International Conference on Software and System Processes (ICSSP '20)*, (New York, NY, USA), pp. 10–11, ACM, Oct. 2020.
[9] S. Natarajan, M. Gunathilake, and P. Hettiarachchi, "Generating uml use case models from software requirements using natural language processing," in *2019 ICACSIS*, pp. 77–82, 2019.
[10] G. Borrego, A. L. Morán, R. R. Palacio, A. Vizcaíno, and F. O. García, "Towards a reduction in architectural knowledge vaporization during agile global software development," *Information and Software Technology*, vol. 111, pp. 101–115, 2019.
[11] S. Sonnentag, C. Niessen, and J. Volmer, "Expertise in software design," in *Handbook of Software Engineering and Knowledge Engineering*, pp. 373–387, 2006.

[12] J. S. Edwards, "Managing software engineers and their knowledge," in *Managing Software Engineering Knowledge*, pp. 5–27, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.
[13] D. Hallmann, ""i don't understand!": Toward a model to evaluate the role of user story quality," *Agile Processes in Software Engineering and Extreme Programming*, pp. 103–112, 2020.
[14] X. Wang, K. Conboy, and O. Cawley, ""leagility" in information systems development: An analysis based on agility and lean concepts," *Communications of the Association for Information Systems*, vol. 30, no. 1, pp. 67–83, 2012.
[15] S. Huang and S. Tilley, "Towards a documentation maturity model," in *Proceedings of the 21st Annual International Conference on Documentation*, pp. 93–99, 2003.
[16] F. A. M. Domínguez, B. R. C. Esquer, R. R. P. Cinco, G. Borrego, and M. A. Q. García, "Evaluation of the robix uml web modeler: Assessing the level of usability," in *2023 12th International Conference On Software Process Improvement (CIMPS)*, (Cuernavaca, Morelos, Mexico), pp. 61–64, 2023.
[17] F. F. J. Münch, "Developer experience: Concept and definition," in *2012 ICSSP*, pp. 73–77, 2012.
[18] M. Kamalrudin, "Improving requirements quality using essential use case," in *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference*, pp. 307–312, IEEE, 2011.
[19] C. Lange and M. Chaudron, "An empirical assessment of completeness in uml designs," in *26th International Conference on Software Engineering*, 2004.
[20] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer, 2024.

**Francisco Antonio Mejía Domínguez** was born in Ciudad Obregon, Mexico, in 2000. He earned his Bachelor's degree from the Sonora Institute of Technology in 2023. Currently, he is pursuing a Master's degree in Engineering Sciences at the Sonora Institute of Technology.

**Manuel Quintana** was born in Huatabampo, Mexico in 1996. He received his Master's degree from the Sonora Institute of Technology in 2020. He is currently pursuing a Ph.D. in Engineering Sciences at the Sonora Institute of Technology. His research is primarily focused on the use of Natural Language Processing (NLP) in Software Engineering.

**Ramón R. Palacio** was born in Navojoa, Mexico in 1977. He received his Ph.D. from the Autonomous University of Baja California in 2010. He worked in software engineering from 1999 to 2000 in a national company. Since 2003, he has been a research professor at the Sonora Institute of Technology. He conducts studies to gain a better understanding of current work practices and, based on this understanding, designs, develops, and evaluates appropriate technologies for various work environments.

**Gilberto Borrego** was born in Hermosillo, Mexico in 1979. He received his Ph.D. from the Autonomous University of Baja California in 2018. He worked in software engineering from 2002 to 2005 and from 2007 to 2014 in various national and international companies. Since 2019, he has been a research professor at the Sonora Institute of Technology. His research interests include agile software development and knowledge management.

**Samuel González-López** is a full professor of the Research Department of the Technological Institute of Nogales. He obtained his PhD in Computer Science at the National Institute of Astrophysics, Optics and Electronics INAOE. His works are focused on improving undergraduate students' writing with the project Linguistic Analysis of Research Project Drafts of Undergraduate Students. He also conducts research under the project of 'Dysfunctional consumer behavior.