Thesis Topic Elaboration and Validation

Version 1.0 – October, 2023

PA2560: ADVANCED TOPICS IN SOFTWARE ENGINEERING

February 16, 2026

Thesis	Tentative title	To What Extent Can AI Assist in Generating Additional Test Cases from User Stories?
	Classification	Software testing, Artificial intelligence, Agile software development
Student 1	Name	Pathipati Thanu Reddy
	e-Mail	thpa24@student.bth.se
	Social security nr	20040505-8546
	Visa expiration date	June 2026
Student 2	Name	Hari Uday Kiran
	e-Mail	udha24@student.bth.se
	Social security nr	20040407-9295
	Visa expiration date	June 2026
Student 3	Name	Viranchi Vilohith Siddulu
	e-Mail	visu24@student.bth.se
	Social security nr	20030823-7114
	Visa expiration date	June 2026
Student 4	Name	Sai Tharun Kuthadi
	e-Mail	sakt24@student.bth.se
	Social security nr	20030927-4496
	Visa expiration date	June 2026
Supervisor Proposals	Name and title	Daniel Mendez Fernandez
	e-Mail	daniel.mendez@bth.se
	Department	Software Engineering

**2012 ACM Computing Classification System: `www.acm.org/about/class/2012`.*

***Co-advisor from industry or a higher education institution (HEI).*

Structured Abstract

In Agile software development, user stories are widely used to specify functional requirements and to guide both implementation and verification. In practice, test cases are commonly derived manually from user stories and their acceptance criteria, which makes the resulting test suites sensitive to practitioner experience, time pressure, and the level of specification detail present in the story. As a consequence, teams often achieve adequate coverage of the primary success flow (the "happy path") while systematically under-covering negative paths, boundary-value cases, and cross-story or end-to-end integration scenarios. This report investigates the human baseline (RQ1) in a single Agile project by examining how testers and developers derive test cases from user stories and which scenario types they perceive as commonly missed or difficult

to identify. A qualitative case study approach is employed using semi-structured interviews and inductive thematic analysis to capture practitioner workflows, heuristics, and omission drivers. The findings highlight an acceptance-criteria-driven test design process, reinforced by schedule constraints and informal, experience-based reasoning, which collectively contributes to recurring gaps in scenario coverage. The report concludes with practical implications for improving test derivation through lightweight checklists, strengthened cross-role collaboration during refinement, and tool support that prioritizes "hard-to-think-of" scenarios rather than duplicating obvious success cases.

Keywords: Agile software development; user stories; test case design; requirements-based testing; qualitative case study; semi-structured interviews; thematic analysis; negative testing; boundary value analysis; integration testing; scenario coverage

1 Introduction

User stories are a common way of specifying functionality in Agile software development. From these stories, teams derive test cases that are executed during implementation and regression testing. This process is usually manual and depends heavily on the experience, domain knowledge, and available time of testers and developers. While the "happy path" is often covered, more subtle edge cases, invalid inputs, and interactions between stories are frequently missed, which can lead to defects that are only discovered late in the process or in production.

Recent work has shown that natural language processing (NLP) techniques and, more recently, large language models (LLMs) can generate test cases from natural-language requirements or user stories, often with promising coverage and effort reduction [8]. However, most existing evaluations focus on metrics such as number of generated tests, requirement coverage, or mutation score, rather than on how human testers perceive individual AI-suggested test cases or how much these suggestions actually improve defect detection in real projects.

This thesis therefore takes a human-in-the-loop perspective: testers remain responsible for test design, while an LLM-based assistant proposes additional test cases from the same user stories [9]. The central question is not "how many tests can the AI generate?" but "which of these AI-suggested tests are actually useful for testers, and do those useful tests help detect more defects?"

1.1 Problem Statement

In the selected Agile project, test cases are currently designed manually from user stories. This manual process tends to favour straightforward functional scenarios and leaves out some negative, boundary, and interaction scenarios that testers themselves report as difficult to think of or easy to forget [7]. LLMs can be used to automatically generate test cases from the same user stories, but it is unclear:

- how testers and developers in this context currently derive tests and where they see gaps,
- which AI-suggested test cases they would actually accept and consider useful additional tests, and

- whether those accepted suggestions help detect more defects when added to the existing human-only test suite.

The problem is thus to empirically assess the *added value* of AI-generated test cases from user stories, under human control and judgement, rather than just counting the raw number of generated tests.

1.2 Aim and Objectives

The overall aim of this thesis is to understand and quantify the role of an LLM-based assistant in supporting test design from user stories in an Agile project [9].

More concretely, the objectives are:

- To characterise how testers and developers currently derive test cases from user stories and which types of scenarios they perceive as commonly missed.
- To configure an LLM-based assistant that generates test cases from user stories and to evaluate, with human testers, which suggestions are useful additional tests.
- To measure the impact of these useful AI-suggested test cases on defect detection when they are added to the existing human-only test suite.

1.3 Contributions

The expected contributions of this work are:

- **Human baseline description:** An empirical description of how testers and developers in the selected project currently design test cases from user stories and which scenarios they perceive as frequently missed.
- **Usefulness of AI-generated tests:** Quantitative and qualitative data on how testers judge AI-generated test cases from user stories, including the proportion of suggestions that are considered useful additional test cases and common reasons for rejection.
- **Impact on defect detection:** An assessment of how much defect detection can be improved by adding only tester-accepted AI-suggested test cases to a human baseline suite.
- **Practical guidance:** Initial guidelines on how to integrate LLM-based tools into manual test design from user stories in a human-in-the-loop manner.

2 Related Work

Early work on deriving tests from natural language focused on requirements-based and model-based testing. More recently, NLP-based approaches such as Chinnaswamy *et al.* automatically extract information from user stories and generate positive and negative test cases using keyword classification and templates, showing that user stories can be a viable source for systematic test generation [1]. Albdeiwi uses transformer-based models to generate test cases from feature descriptions and compares AI-generated tests to human-written ones, reporting that AI can match humans in some scenarios but often lacks contextual detail and precision [2].

With the emergence of LLMs, several studies evaluate them as general-purpose test generators. Li and Yuan evaluate large language models as test case generators across multiple programming tasks and report that current models can generate high-quality tests for many scenarios but still struggle on harder tasks [3]. Wang *et al.* propose TESTEVAL, a benchmark for comparing LLMs on test generation tasks, and find that line and branch coverage remain challenging for many models [4]. Dakhel *et al.* show that pre-trained LLMs can produce effective unit tests but note that coverage alone is not a reliable indicator of fault detection [5]. Celik and Mahmoud survey LLM-based automated test case generation and observe that most evaluations rely on technical metrics such as coverage and diversity, with limited attention to human acceptance and workflow integration [6].

Practice-oriented reports discuss using LLMs or GenAI tools to generate test cases directly from requirements or user stories. Korraprolu *et al.* describe test case generation for natural-language requirements and note that generated tests may still fail adequacy criteria for some requirements [7]. Fraunhofer IESE report on using LLMs to speed up test design from textual requirements and emphasise the importance of human validation before adoption [8]. Thoughtworks has also explored using generative AI to generate test cases from user stories, discussing both the potential for speed and the need for human oversight [9].

3 Research Questions

Based on the problem statement and related work, this thesis investigates the following research questions:

- **RQ1 – Human baseline:** How do testers and developers currently derive test cases from user stories in the selected project, and what kinds of test scenarios do they perceive as commonly missed or difficult to come up with?

- **RQ2 – Perceived usefulness of AI suggestions:** "To what extent do human testers accept AI-generated test cases as useful additions (i.e., valid, unique, and understandable) to their manually designed test suites?

- **RQ3 – Effect on defect detection:** "How much does the defect detection capability improve when these human-verified AI test cases are added to the original manual test suite?"

4 Selected Research Question

This report focuses exclusively on RQ1. Out of the above questions, RQ1 was selected for in-depth investigation as it establishes the human baseline for test design practices and gaps. RQ1 examines the current process by which testers and developers derive test cases from user stories and identifies which test scenarios are commonly overlooked or hard to conceive. RQ2 and RQ3 are beyond the scope of this study and are left for future work. By concentrating on RQ1, we aim to gain a detailed understanding of existing practices and challenges in manual test design from user stories, which can inform subsequent research on potential improvements or tools.

5 Research Methodology

To address RQ1, we employed a qualitative, interview-based case study approach. The study was conducted in the context of a single Agile software project (the "selected project"), focusing on the practices of its testing and development team. We chose semi-structured interviews as the primary data collection method in order to gather in-depth insights into how practitioners derive test cases from user stories and what scenarios they might miss. Each interview followed a predefined guide (see Appendix A) but allowed flexibility for participants to elaborate on their experiences and perceptions.

All interviews were transcribed and analyzed using thematic analysis. We followed an inductive coding process: first, we familiarized ourselves with the transcripts and open-coded statements relevant to test design approaches or missed scenarios. These codes were then iteratively refined and grouped into higher-level themes reflecting common practices and challenges. Using a thematic analysis approach ensured that recurring patterns and ideas emerged from the data without imposing a rigid predetermined framework. The qualitative analysis was conducted carefully to maintain credibility, including peer discussion on code definitions and revisiting transcripts to validate interpretations. The result is a set of themes that directly address RQ1, providing a rich description of current test derivation processes and perceived gaps.

Method at a glance.

- **Design:** Single-case qualitative case study in an Agile software project.
- **Data source:** Semi-structured interviews with practitioners involved in deriving or consuming test cases from user stories.
- **Analysis:** Inductive thematic analysis with iterative coding and theme refinement.
- **Output:** Practitioner-derived themes describing (i) test-case derivation workflows and (ii) commonly missed or difficult scenario types.

6 Execution of the Interview Study

6.1 Participants and Context

We interviewed a total of 8 participants from the selected project, which is a mid-sized software development project following Agile methodologies. The participants comprised 4 testers (including one QA lead) and 4 developers (including one senior developer), covering a range of roles involved in writing or using test cases. Their experience in software testing/development ranged from approximately 2 to 10 years (with an average around 5 years), providing a variety of perspectives from junior to senior team members. All participants were familiar with using user stories as the basis for development and testing in their daily work. Participation was voluntary and all individuals were assured of anonymity; thus, in our results we refer to them by codes (P1 through P8) rather than by name or role when quoting their responses.

6.2 Procedure

Each participant took part in a one-on-one semi-structured interview of about 45 minutes. The interviews were conducted in a mix of in-person and video conference settings, depending on

participant availability. At the start of each interview, we obtained consent for audio recording and explained the purpose of the study (to learn about their test design process and challenges). The interview questions (Appendix A) covered topics such as: how the participant typically derives test cases from a given user story, their strategy for identifying different types of scenarios (e.g. normal vs. edge cases), any categories of tests they find difficult to come up with, and examples of scenarios they recalled missing in the past. Participants were also encouraged to think of recent user stories and discuss what tests they designed and whether any important tests were initially overlooked.

During the interviews, we probed with follow-up questions to clarify how they decide a test case is necessary or how they handle ambiguous acceptance criteria. We also collected any relevant artifacts that participants were willing to share, such as example user stories and their associated test cases, to provide context and triangulate the interview statements. However, the core analysis was based on the interview transcripts. All recordings were transcribed verbatim.

6.3 Data Analysis

Using the transcripts, we performed thematic coding as described in the methodology. Initially, one researcher reviewed the transcripts line by line and assigned open codes to segments of text describing a distinct idea (for example, a code for "uses acceptance criteria to design tests" or "forgotten edge case example"). After coding all transcripts, similar codes were grouped under candidate themes, and these themes were refined through an iterative review process. For instance, codes related to missing input validations or error conditions were grouped under a theme concerning "Negative path scenarios missed." We employed techniques to enhance reliability, such as cross-checking code assignments on select transcript sections and discussing any discrepancies until consensus was reached on the theme definitions (the final coding scheme is provided in Appendix B). The outcome of the analysis was a set of prominent themes characterizing (a) how test cases are derived from user stories in practice, and (b) what kinds of test scenarios tend to be missed or considered challenging. In the next section, we present these results with illustrative examples and direct quotes from participants.

7 Results

After analyzing the interview data, we identified several key themes that shed light on RQ1: How testers and developers derive test cases from user stories, and which scenarios they commonly miss or find difficult. The findings are organized into two main areas: (1) the practices and strategies participants use to derive test cases from user stories, and (2) the types of test scenarios that are frequently overlooked or challenging for them. Within each area, we describe the themes in detail and provide representative quotes from participants (labeled P1–P8) to illustrate each point.
Table 1 below summarizes the main categories of commonly missed scenarios mentioned by the participants, along with the number of participants who brought up each category.

(Other categories like certain non-functional tests were mentioned by one or two participants but not commonly across the group; the focus here is on the major themes.)

Table 1: Missed or Difficult Test Scenario Categories (from interviews, $N = 8$ participants).

Scenario Category	**Description of Scenarios Commonly Missed**	**Participants (n=8)**
Negative paths (error/exception scenarios)	Tests for invalid inputs, error handling, or exception flows often omitted in the initial test design if not explicitly stated in the user story.	6 out of 8
Boundary values (extreme cases)	Tests using extreme or boundary input values (e.g., max/min limits, empty inputs) that are frequently overlooked or not thoroughly covered.	5 out of 8
Integration scenarios (cross-story or end-to-end)	Tests covering interactions between multiple user stories or components (end-to-end scenarios) which are not considered during story-level testing.	4 out of 8

7.1 Test Case Derivation Practices from User Stories

Participants described a largely experience-driven and criteria-focused approach to deriving test cases. Almost all interviewees stated that they start with the user story's content and acceptance criteria as the primary guide for creating test cases. Typically, a tester will read the user story description and any given acceptance criteria line by line to identify the "happy path" (primary flow) and ensure there is at least one test case covering each acceptance criterion. As one tester (P3) explained, "I usually begin by checking each acceptance criterion in the story and writing a test for it. That covers the basic functionality the story needs to deliver." This reliance on acceptance criteria means that if a scenario is not mentioned in the story or criteria, it is less likely to be tested initially. Another participant (P5) noted that time constraints and workload pressure reinforce this focus on core scenarios: "When we're in a rush, we validate the main use-case described and assume everything else will be fine. The weird cases don't always get attention unless someone remembers them." This suggests that under schedule pressure, participants tend to design the minimum set of tests that satisfy the explicit requirements, potentially overlooking implicit scenarios.

Personal experience and intuition play a significant role in test design. Several testers mentioned that beyond the written criteria, they try to think of "what could go wrong" based on their past experiences with similar features. For example, P7 (a senior tester) described using a mental checklist of common problem areas: "For a form input story, I always ask myself: what if the input is empty? What if it's way too long? What if the user cancels? Those aren't always in the story, but I've been burned by those cases before, so I add tests for them if I remember." This quote highlights that more experienced testers proactively consider additional scenarios such as empty or overlong inputs (boundary cases) and alternate flows. In contrast, less experienced participants admitted they sometimes struggle to go beyond the happy path unless prompted. One developer (P6) candidly stated, "As a developer, I tend to write tests for the scenarios I expect to code. I might miss some edge cases because I'm focused on making the main scenario work." This indicates that developers writing their own tests might only cover what the user story explicitly demands, expecting that specialized testers or later stages will catch anything else. Indeed, a theme that emerged is a division of mindset: developers often trust testers to handle unusual scenarios, whereas testers rely on their own heuristics or team discussions to surface those scenarios.

Additionally, collaboration and review practices were mentioned as a way tests are derived or refined. A few participants described that during backlog refinement or sprint planning meetings, testers and developers discuss each user story, and sometimes these conversations lead to identifying test scenarios. P2 (tester) gave an example: "When the team talks about a story, I'll bring up potential edge cases and see if the product owner thinks they're in scope. If they are, we add acceptance criteria or I note down to test them." However, this practice was not consistent across all participants—some said such discussions are brief and focused on functionality, not explicit testing scenarios. Formal peer review of test cases (e.g., one tester reviewing another's test design) was rare; only one participant mentioned occasionally reviewing test ideas with a colleague. In summary, deriving test cases is largely an individual effort guided by the story text and personal experience, with limited systematic process to ensure all scenario types are considered.

7.2 Commonly Missed or Difficult Scenarios

When asked about the kinds of test scenarios that are often missed or particularly hard to think of, participants consistently mentioned three main categories: negative paths, boundary values, and integration scenarios (see Table 1). These align with the problem context that certain "non-happy-path" tests tend to be overlooked. We describe each category below, with examples and quotes illustrating the challenges.

7.2.1 Negative paths (error and exception scenarios)

A majority of participants (6 out of 8) admitted that they often do not initially create test cases for error conditions, invalid inputs, or exception flows unless those are explicitly described. The normal tendency is to assume the system will handle errors, or to treat error handling as out of scope of the user story's acceptance criteria. One tester (P3) noted, "If the story doesn't mention how errors should be handled, we kind of ignore that at first. We usually test what's in the acceptance criteria, so things like 'what if the user inputs wrong data' can slip through." This reflects a common mindset that unmentioned error scenarios are implicitly handled or not critical, which can lead to missing tests. Indeed, several participants recalled bugs that occurred because an error case wasn't tested. For example, P8 (developer) remarked, "We had an issue where entering a really bad value caused an exception – none of our tests caught it because none of us thought to try a value that violates the format. We figured the UI would prevent it, but it didn't." This quote illustrates how an error scenario (invalid input format) was overlooked. Another participant summed it up by saying that negative testing requires a deliberate effort: "Happy path testing is almost automatic. Negative testing is something you have to consciously switch into, otherwise it gets forgotten" (P7). This suggests that testers recognize the importance of negative path testing, but it may not be integrated into their default workflow.

7.2.2 Boundary value scenarios (extreme or edge inputs)

Over half of the participants (5 out of 8) mentioned difficulty in anticipating and testing extreme boundary conditions. These include upper and lower limits of input ranges, maximum lengths, zero or empty inputs, and other edge values that are at the boundaries of acceptable input. Testers said that such cases are easy to miss during design because the user story usually provides typical examples, not extremes. P2 (tester) gave a telling example: "The story

described a field for entering a name, and of course I tested normal names. It never occurred to me to test 200 characters of gibberish. Later we found out it crashes if the input is super long." This incident shows a missed boundary test (an overly long string) resulting in a defect. Similarly, P5 noted that numeric boundaries can be overlooked: "If a story says 'user can enter their age', we test a normal age like 30. We might not test 0 or 150 unless we consciously think about limits." In this case, values at or beyond expected limits (e.g., zero or an extremely high age) might not be considered. Such edge cases often only come to light after production issues or explicit brainstorming. A couple of participants mentioned they sometimes use equivalence class and boundary analysis techniques in formal testing, but admitted that in agile practice, they do so inconsistently. Time pressure also plays a role here: "We prioritize a working basic scenario. Testing every tiny boundary is often left for later or never" (P6). Overall, boundary value tests are commonly recognized in theory but frequently missed in initial practice due to oversight or low perceived priority, unless the team has a checklist or prior bug to remind them.

7.2.3 Integration scenarios (cross-story interactions)

About half of the participants (4 out of 8) raised the issue of integration or end-to-end scenarios being missed when deriving tests for a single story. In an Agile setting, each user story is often tested in isolation for its specific functionality, but problems can arise when multiple stories or components interact. P1 (tester) explained this gap: "We test each story on its own, but we don't always think about how it works with other features that were already there. Sometimes a story might break something in a different area, or two new stories don't play well together." Such integration issues might involve data flowing between modules or a sequence of actions that spans multiple user stories. One developer (P6) provided an example of a missed integration test: "The payment confirmation story worked fine on its own and the order creation story worked on its own, but doing an end-to-end test from order to payment revealed an ID mismatch bug. No one wrote a test covering both because we treated them as separate tasks." This underscores how cross-story scenarios can slip through testing if each story's testing is done in isolation. Not every participant saw this as their responsibility—some assumed that separate system integration testing or user acceptance testing phases would catch such issues. However, in an Agile continuous delivery environment, dedicated integration testing may be minimal, so story-level testers ideally need to consider broader workflows. P4 (tester) noted, "Unless we explicitly think of a user journey that spans multiple stories, those combined scenarios won't be tested until perhaps an exploratory test or a customer does it." The difficulty is that integration scenarios are often not clearly described in any single user story, making them "invisible" unless one has a holistic view of the feature set.

In addition to these main categories, a few participants mentioned other scenarios that can be challenging, albeit less frequently. For instance, two participants brought up non-functional tests (like performance or security) as something not derived from user stories: "Performance tests are never in the user story, so if we don't specifically plan for them, they won't happen" (P8). Others noted that alternate flows (valid but less common ways to accomplish the same goal) might be missed—for example, a user editing something out of order—but often these overlap with what we have categorized as negative or exceptional flows. Overall, the themes from our interviews demonstrate that while participants are adept at creating tests for the primary specified behaviors, they acknowledge consistent blind spots in testing edge cases, error handling, and interactions. These findings provide a concrete baseline of human testing practices and gaps, as perceived by practitioners in the project.

8 Lessons Learned

From this study of RQ1, we derived several lessons learned regarding current test design practices and their shortcomings:

- **Test design is largely informal and experience-based:** There is no standardized or rigorous procedure in place for deriving test cases from user stories. Each tester or developer relies on personal experience and interpretation of the user story, which leads to variability in coverage. This ad-hoc approach means important scenarios can be overlooked if no one on the team happens to think of them.

- **"Happy-path" bias and omission of edge cases:** Even skilled practitioners tend to focus on the main success scenario (happy path) defined by the user story, often at the expense of negative tests and edge cases. We learned that without explicit prompts or prior incidents, critical scenarios like error conditions or boundary inputs are commonly omitted in initial testing. This bias toward expected behavior is a natural outcome of requirement-focused testing, but it leaves gaps in quality.

- **Time pressure exacerbates scenario gaps:** Under tight sprint deadlines, participants admitted to trimming the scope of testing to what is most obvious or what is explicitly required. Consequently, less obvious scenarios (which might be low-probability but high-impact) are deferred or ignored. The lesson here is that schedule pressure can directly impact test thoroughness, reinforcing the need for techniques to surface edge-case tests efficiently.

- **Collaboration and review can catch missing tests (when it happens):** On the occasions where testers and developers engaged in discussion (e.g., during refinement) or reviewed each other's test ideas, some of the overlooked scenarios were identified. This suggests that a more collaborative test design process could be beneficial. However, we learned that such collaboration is not routine; making it a standard practice (e.g., through pair testing or test case reviews) could serve as a safeguard against individual blind spots.

- **Certain scenario types consistently require support:** The recurring nature of missed negative, boundary, and integration tests indicates a systematic weakness that might be addressed through additional support. Participants implicitly expressed that they would welcome reminders or tools for these scenario types. One lesson is that tool support or guidelines (like checklists of common edge cases) could significantly improve test coverage. For example, a checklist that forces consideration of "error inputs" or "extreme values" for each story might have prompted participants to include those tests. This lesson directly points to the value of augmenting human test design with resources that ensure broader thinking.

These lessons highlight areas where the current practice could be improved and also provide insight into what kind of interventions (process changes or tools) might be most effective. In particular, the prevalence of the happy-path focus and the difficulty of thinking of all edge cases underscore the need for approaches that can systematically broaden test scenario coverage without imposing too much overhead on the team.

9 Discussion and Implications

The findings from our interview study offer a detailed snapshot of how testing is carried out from user stories in an Agile project, revealing strengths and weaknesses that carry broader implications for both research and practice. Firstly, our results confirm and contextualize known challenges in requirements-based testing. We observed that testers and developers naturally gravitate toward the specified functionality, often missing implicit scenarios such as error handling and edge cases. In fact, the specific categories of missed scenarios (negative paths, boundary values, integration flows) identified by our participants mirror those reported as problematic in earlier studies of requirements-driven test generation [7]. This convergence adds confidence that the issues are not unique to our case: manual test design inherently has blind spots which have been recognized by other researchers and practitioners.

However, our study contributes new insights by providing qualitative, human-centric explanations for why these blind spots occur. Unlike many existing works that evaluate test generation via technical metrics, we captured the thought processes and situational factors (like time pressure and reliance on acceptance criteria) that lead to omissions. This addresses a gap noted by Celik and Mahmoud [6], who pointed out that much of the work on automated testing focuses on metrics like coverage rather than on human workflows and acceptance.

9.1 Implications for practice

The patterns we uncovered suggest several practical steps that Agile teams could take to improve test coverage from user stories. One implication is the value of introducing explicit checklists or guidelines for test design. Given that categories like negative and boundary scenarios are systematically missed, teams could adopt a simple checklist that prompts testers to consider these for every user story (e.g., "Have we thought about invalid inputs? Extreme values? Related features?"). This would operationalize the mental reminders that experienced testers like P7 already use, making them part of the team's routine. Another implication is to strengthen the collaborative aspect of testing our findings showed that when testers and developers did discuss scenarios together, more gaps were filled. Teams might formalize this by including "what could go wrong" brainstorming during refinement meetings or doing pairwise reviews of test cases. This could mitigate individual bias, as different perspectives often surface different scenarios.

9.2 Implications for tool support

Our study also has implications for tool support in the testing process. The fact that testers struggled to come up with certain scenarios indicates an opportunity for tools (such as AI-based test case generation assistants) to add value by suggesting those very scenarios. For instance, an LLM-based assistant could be tuned to propose test cases that involve boundary values or error conditions derived from the story context—areas our participants commonly missed. Importantly, the perceived gaps in human testing provide a roadmap for what an AI assistant should focus on: it should complement the human by covering the "hard-to-think-of" cases rather than the obvious happy paths. This aligns with the motivation behind RQ2 and RQ3 of the broader thesis.

9.3 Implications for research

Our findings provide a foundation for further research in several directions. First, they highlight the importance of studying human testing processes qualitatively. Future studies could build on our work by examining whether these patterns hold in other contexts (e.g., different domains, larger teams) or by quantifying the impact of missing tests on defect leakage. Additionally, our research can inform experimental evaluations of automated test generation. The effectiveness of an AI test assistant should be measured not just by code coverage, but by whether it successfully addresses the specific gaps we identified. Another research implication is exploring techniques to integrate systematic test design with agile workflows: how can we make consideration of edge cases a natural part of the story development cycle?

In summary, the discussion highlights that our RQ1 study serves as a reality check for both practitioners and tool-builders. It confirms that certain types of tests are consistently missed in practice (implicating a risk to quality), and it emphasizes that any interventions to address this should be aligned with human workflows and cognition. By understanding why testers miss certain scenarios, we can better tailor solutions—whether process changes, educational efforts, or AI-driven aids—to effectively support them.

10 Improvements for Future Execution

While the present study provided valuable insights on a single-project scale, there are several ways we could improve and scale up the execution of this research in the future:

- Include multiple projects and teams.
- Combine interviews with direct observation.
- Introduce a survey instrument.
- Refine the interview guide for depth.
- Leverage quantitative metrics alongside qualitative data.
- Ensure consistency and training for researchers.

By making these improvements, a scaled-up study could both validate the initial findings and expand upon them. Ultimately, scaling the execution would allow a transition from an in-depth single-case understanding to broader best practices and recommendations. It would also set the stage for evaluating interventions (like the introduction of an AI test assistant) in a controlled, measurable way across multiple settings.

11 Threats to Validity

In interpreting the results of this study, it is important to consider potential threats to validity and how we mitigated them:

- **Credibility (Internal Validity):** Participant bias and social desirability bias may influence interview responses; mitigation includes anonymity, non-judgmental framing, probing for concrete examples, and member checks.

- **Confirmability (Researcher Bias):** Researcher interpretation may influence themes; mitigation includes systematic coding, an audit trail, and peer debriefing.

- **Transferability (External Validity):** Single-project scope limits generalizability; mitigation includes rich context description and future replication across settings.

- **Reliability (Dependability):** Qualitative findings can vary by researcher and time; mitigation includes a defined interview guide, clear analysis procedure, and documented chain of evidence.

- **Limitations in data (Completeness):** Self-reporting and recall bias may omit details; mitigation includes prompting for recent examples and triangulation with artifacts when available.

In conclusion, while there are valid concerns regarding bias, generalizability, and data completeness, the study design aims to document and mitigate these threats where feasible. Future work involving broader samples or additional methods can further validate and refine these conclusions.

12 Conclusion

In this report, we presented a comprehensive examination of RQ1: How testers and developers derive test cases from user stories, and what scenarios they commonly miss. Through a qualitative interview study in an Agile project, we gained rich insights into the current manual test design process and its limitations. The Introduction and Problem Statement set the stage by highlighting that while user stories are central to Agile development, relying solely on them for test case design can leave important gaps (such as missed edge cases and negative paths). Our Results confirmed this: participants typically create tests focusing on the main acceptance criteria (the "happy path"), and they often overlook certain categories of scenarios like error conditions, boundary values, and cross-story interactions. We documented these categories in detail, providing evidence with direct quotes and examples of bugs that arose from missed tests.

The Aim and Objectives of the thesis included not only characterizing the human baseline (RQ1) but also evaluating an AI-based test generation assistant (RQ2, RQ3). While the scope of this report was limited to RQ1, our findings serve as that baseline description of human testing practices and gaps essentially fulfilling the first objective and the first expected contribution. We identified clear opportunities where an AI or any systematic approach could complement human testers, specifically by suggesting or prompting the kinds of test scenarios humans tend to forget.

In the Discussion, we connected our findings to existing literature and practical implications. The recurring omissions in human-generated test suites underscore why purely metric-driven assessments of testing (like code coverage percentages) can be misleading. A high coverage might still miss a critical scenario if that scenario was outside the testers' consideration. Therefore, one key conclusion is that improving testing in Agile requires addressing human cognitive limits whether through better processes (like checklist-driven test design) or tools (like intelligent assistants) or, ideally, both.

We also presented Lessons Learned and Threats to Validity to transparently reflect on what was discovered and how confident we can be in the interpretations. In conclusion, this study contributes a nuanced understanding of how Agile teams craft test cases from user stories and where they struggle. Recognizing these gaps is the first step toward bridging them. The findings provide a foundation for future research and practical interventions that can make "commonly missed" scenarios far less common.

References

[1] A. Chinnaswamy, B. A. Sabarish, and R. D. Menan, "User Story Based Automated Test Case Generation Using NLP," in *Computational Intelligence in Data Science (ICCIDS)*, 2024.

[2] Y. Albdeiwi, "Generating Test Cases Using Natural Language Processing," Master's thesis, Lund University, 2024.

[3] K. Li and Y. Yuan, "Large Language Models as Test Case Generators: Performance Evaluation and Enhancement," *arXiv preprint arXiv:2404.13340*, 2024.

[4] W. Wang et al., "TESTEVAL: Benchmarking Large Language Models for Test Case Generation," in *Findings of NAACL*, 2025.

[5] A. M. Dakhel et al., "Effective Test Generation Using Pre-trained Large Language Models," *Journal of Systems and Software*, 2024.

[6] A. Celik and Q. H. Mahmoud, "A Review of Large Language Models for Automated Test Case Generation," *Machine Learning and Knowledge Extraction*, 2025.

[7] B. R. Korraprolu et al., "Test Case Generation for Requirements in Natural Language," in *Proc. of an ACM Software Engineering venue*, 2025.

[8] Fraunhofer IESE, "Software Testing: Using Large Language Models to Save Test Design Time," Technical Blog, 2025.

[9] T. V. Hoang, "Can We Use Generative AI to Generate Test Cases from User Stories?," Thoughtworks Engineering Blog, 2025.

A Interview Guide (Semi-Structured Questions)

1. **Background** – Could you briefly describe your role in the project and your experience with designing or executing test cases? (This is to establish context: tester vs developer, years of experience, etc.)

2. **Test Case Derivation Process** – When you receive a new user story, how do you go about creating test cases for it? Please walk me through your process. (Possible follow-ups: What sources of information do you use? Do you rely on acceptance criteria or something else? At what point do you consider a test suite "good enough" for a story?)

3. **Scenario Coverage** – How do you ensure that different types of scenarios are covered by your tests? For example, do you think about edge cases, error cases, or alternative flows while designing tests? (Follow-up: Can you give an example of an edge case or negative case you considered? If they don't mention these, prompt: Do you explicitly test for things like invalid inputs or boundary values? Why/why not?)

4. **Common Challenges** – What kinds of test scenarios are the most difficult for you to come up with when testing a user story? (Follow-up: Why do you think those are difficult? Is it due to the story not having details, time constraints, lack of tools, etc.?)

5. **Missed Scenarios and Examples** – Can you recall a situation where an important test case was missed initially and a bug was discovered later because of it? Please describe what happened and what kind of scenario was missed. (If they struggle to recall, broaden: "It could be you or someone on the team; any instance of a missed test.")

6. **Use of Tools/Checklists** – Do you use any specific techniques or tools to help generate or check test cases (for instance, a checklist of common test types, pair brainstorming, or an automated test generation tool)? If so, how do these help? If not, do you feel any such aid would be useful?

7. **Decision of Test Case Importance** – How do you decide if a test case is worth implementing? (E.g., if you think of a very unlikely scenario, what makes you include it or leave it out?)

8. **Integration and System Considerations** – When testing a user story, to what extent do you consider interactions with other user stories or components? (Follow-up: Do you ever design tests that span multiple user stories or features, or is it strictly one story at a time? Can you give an example?)

9. **Wrap-Up** – In your opinion, what could help you or your team derive better test cases from user stories? (This could be anything: more information in stories, more time, tools suggesting tests, training, etc.)

10. **Anything Else** – Is there anything else about your test design process or challenges that we haven't covered and you think is relevant?

Note: The interviewer used the above questions flexibly; not all were asked verbatim in every interview. The flow was adjusted based on each participant's responses. However, all major topics were covered by the end of each session.

B Coding Scheme (Extract from Codebook)

After transcribing the interviews, we developed a coding scheme to categorize the data. The table below lists the key codes and themes, along with their descriptions and example excerpts from the data. Codes were grouped into higher-level themes corresponding to our research focus on test derivation practices and missed scenarios.

Test Design Approach Codes

- **AcceptanceCriteria-Focused** – Participant derives tests directly from the user story's acceptance criteria or explicit requirements. Example: "I go through the acceptance criteria one by one and write tests for each" (P3).

- **HappyPath-First** – Tendency to design the main success scenario tests before anything else. Often implied by statements about verifying the story's primary function. Example: "First, ensure the basic story works as expected" (P1).

- **Experience-Heuristic** – Use of personal experience or intuition to add test cases beyond what is in the story. This includes mental checklists or recalling past bugs. Example: "I remember a bug from last time, so I test that scenario too" (P7).

- **Time-Constraint** – References to time pressure affecting test design decisions (e.g., skipping tests or deferring them). Example: "We were short on time, so I only did the basics" (P5).

- **Collaboration-Review** – Any mention of collaborating with others (developers, testers, product owners) to come up with test cases, or reviewing test ideas together. Example: "We discussed in refinement and identified a couple of edge cases to test" (P2).

Missed Scenario Codes

- **Negative-Missed** – An error condition or invalid input case that was not tested initially. Example: "Didn't test what happens if the file is corrupt – and that caused an issue" (P8).

- **Boundary-Missed** – A boundary value or extreme input that was overlooked. Example: "We didn't consider max characters for the text field" (P2).

- **Integration-Missed** – A scenario involving integration of multiple features or stories that was not tested. Example: "When Feature A and B were used together, there was a bug – we hadn't tested that combination" (P6).

- **AlternateFlow-Missed** – A valid but alternative flow or use-case not covered by tests (distinct from error cases). Example: "If the user skipped a step and went straight to checkout – we never tested that path" (P4).

- **NonFunctional-Missed** – References to performance, security, or other non-functional aspect not tested. Example: "We assumed performance was fine; didn't do any load testing for that story" (P8).

Other Contextual Codes

- **Story-Clarity** – If a participant mentioned the clarity (or lack thereof) of the user story impacting testing (e.g., ambiguous requirements leading to missed tests).
- **Tool-Usage** – Any mention of tools (or lack of tools) affecting their test case design (e.g., "We don't have automated test generation, so it's manual brainstorming").
- **Defect-Aftermath** – Participant describes a defect that occurred and what was learned or changed in test approach after that.

Themes and Grouping

From these codes, we formed broader themes. For example, **AcceptanceCriteria-Focused** and **HappyPath-First** codes were grouped under a theme called "Reliance on Story Criteria for Test Design", which speaks to the practice of starting with the happy path. Codes like **Negative-Missed**, **Boundary-Missed**, and **AlternateFlow-Missed** became part of the "Missed Scenario Types" theme, with sub-categories for each type. The code **Collaboration-Review** contributed to a theme on "Mitigating Factors" (things that help prevent missed cases, like collaboration).

This coding scheme was iteratively refined. Two transcripts were initially coded by two researchers independently using these codes, then the researchers compared and adjusted definitions for consistency. The final codebook (excerpted above) was applied using a qualitative analysis tool.

C Sample Transcript Excerpt (Interview Log)

Below is an excerpt from one of the interview transcripts (Participant 4, Tester), illustrating the dialogue format and the kind of information gathered.

> **Interviewer:** Can you give an example of a time when a test scenario was missed and a bug slipped through as a result?
>
> **Participant P4 (Tester):** Sure. One example was a story about uploading a profile picture. The acceptance criteria said it should accept JPG and PNG images. So, I tested with a normal JPG and a normal PNG, and it worked. But I didn't test what happens if the image file is not actually an image but has the wrong extension or is corrupted.
>
> **Interviewer:** Okay, and what happened?
>
> **Participant P4:** A few weeks later, a user tried to upload a file that wasn't a real image – I think they renamed a `.txt` file to `.jpg`. The app didn't handle it well; it crashed because we weren't validating the file content, only the extension. We got a bug report about that.
>
> **Interviewer:** Right. So the scenario was an invalid file masquerading as a valid format?
>
> **Participant P4:** Exactly. We missed that negative scenario. In hindsight, it's an obvious one – you should check the file actually is an image. But at the time, I was just going off the user story, which only talked about allowing JPG/PNG. It didn't mention what to do with bad files, so I didn't think to test it.

Interviewer: Did this experience change anything about how you test going forward?

Participant P4: It did, at least for similar stories. Now whenever we have an upload or any user input, I try to include one test for "what if this input is malformed or malicious". It's not always formally in the acceptance criteria, but I've learned my lesson that those cases can happen and cause errors.

(In this excerpt, we see the participant identify a missed negative path scenario—an invalid file upload—and reflect on why it was missed (not specified in story, thus not thought of). This kind of dialogue was common across interviews, helping pinpoint the gap between story content and thorough testing.)

D Detailed Results Table – Missed Scenario Categories with Examples

This table provides additional detail for the main categories of missed test scenarios identified in the study.

Table 2: Detailed Results Table – Missed Scenario Categories with Examples

Missed Scenario Category	Example Scenario (from participant data)	Mentioned by
Negative Path (error or exception cases)	Uploading a file with an unsupported format caused the application to crash because no error-handling test was implemented (P4). Another example involved entering invalid input, such as non-numeric values in numeric fields, which was not tested and resulted in runtime errors (P8).	6
Boundary Value (extreme input cases)	The maximum allowed number of characters in a text field was not initially tested, which led to a system crash when a very long string was eventually entered (P2). Similarly, boundary cases such as zero or extremely high numeric values were overlooked during testing (P5).	5
Integration Scenario (cross-feature interaction)	An end-to-end scenario—from placing an order to completing payment—was not tested, leading to a failure in order confirmation when the payment service returned a specific response (P6). In another case, a newly introduced feature failed to interact correctly with an existing module because both were tested in isolation (P1).	4

Notes: "Mentioned by" indicates how many out of the 8 total participants provided an example of this category of missed scenario. The examples are paraphrased from incidents described in the interviews, with participant codes in parentheses. Each category corresponds to a theme discussed in the Results section. The consistency in these reports across multiple team members

suggests that these are not isolated incidents but rather indicative of patterns in the test design process.