CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	User stories as boundary objects in agile requirements engineering: A theoretical literature review
Authors	Sporsem, Tor Thorsrud;Dingsøyr, Torgeir;Stol, Klaas Jan
Publication date	2025
Original Citation	Sporsem, T., Dingsøyr, T. and Stol, K. J. (2025) 'User stories as boundary objects in agile requirements engineering: A theoretical literature review', Journal of Systems and Software, 233, 112693 (21pp). https://doi.org/10.1016/j.jss.2025.112693
Type of publication	Article (peer-reviewed)
Link to publisher's version	10.1016/j.jss.2025.112693
Rights	© 2025, The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/) - https://creativecommons.org/licenses/by/4.0/
Download date	2026-11-17 09:17:10
Item downloaded from	https://hdl.handle.net/10468/18378



University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

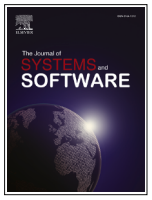
The Journal of Systems and Software 233 (2026) 112693



Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss





User stories as boundary objects in agile requirements engineering: A theoretical literature review

Tor Sporsem [a,b,*], Torgeir Dingsøyr [a,c], Klaas-Jan Stol [b,d,e]

[a] NTNU, Norway
[b] SINTEF, Norway
[c] SimulaMet, Norway
[d] University College Cork, Ireland
[e] Lero–the Research Ireland Centre for Software, Ireland

ARTICLE INFO

Editor: Christoph Treude

Keywords:
User stories
Theoretical literature review
Agile requirements engineering
Boundary objects theory
Knowledge boundaries
Software development

ABSTRACT

User stories have become the predominant method for managing requirements in software development, used by approximately half of all software developers. Despite this widespread adoption, there is limited theoretical understanding of how user stories are used in practice. Through a theoretical literature review of 14 industry studies, we develop five theoretical propositions: 1) user stories facilitate shared understanding between developers and users; 2) small user stories help developers cope with change; 3) clarifying the 'why' in user stories reinforces focus on user needs but adds complexity to the development process; 4) conversations triggered by user stories can hamper the sense of productivity; and 5) user stories as recorded in writing degrade over time. Using boundary object theory as an analytical lens, we explain how user stories facilitate knowledge transfer across syntactic, semantic, and pragmatic boundaries between developers and users. This theoretical lens offers new insights into why some user stories succeed while others fail to bridge boundaries between users and developers. The review highlights the sharp contrast between the widespread use of user stories among practitioners and the limited academic research on their practical application. We end with identifying opportunities for future research, particularly on how user stories can be used in the era of generative AI.

1. Introduction

A foundational activity in software development is to understand users and other stakeholders, so as to determine what they need the software to do. This is known as requirements engineering (RE) (Nuseibeh and Easterbrook, 2000; Cheng and Atlee, 2007). While there are different views on *how* to do requirements engineering, many agree that the fundamental purpose is to discover *what* users need and *why* (Nuseibeh and Easterbrook, 2000; Zave, 1997). In other words, it involves identifying and defining the *problems* the software is intended to solve (Cheng and Atlee, 2007). The goal of RE, then, is to enable knowledge integration: those who experience the problem—such as users, customers and product owners—contribute their knowledge so that developers can translate it into software requirements (Tiwana, 2004).

Traditionally, the RE field relied on extensive documentation to describe system requirements.[1] These documents were the primary way to transfer knowledge about the problem to developers (Sommerville and Sawyer, 1997). In the early 2000s, agile methods (or 'agile' for short in the remainder of this article) gained popularity for recognizing that identifying all requirements upfront, the standard approach in traditional plan-driven (waterfall) development, was practically impossible (Baham and Hirschheim, 2022). Instead, agile methods emphasize discovering requirements iteratively throughout the development process, allowing feedback from stakeholders during the whole development process. In particular, *user stories* have become the most commonly used artifact to capture requirements among software developers (approximately 50 % of all software developers (Kassab, 2015; Lucassen et al., 2016a) and 90 % of agile developers (Wang et al., 2014)).

User stories are brief, informal descriptions of a software feature, written from a user's perspective to clearly convey user needs (Cohn, 2004b). The level of detail in user stories is intentionally minimal, encouraging developers to interact with users to clarify specifics—or as

* Corresponding author.
E-mail address: tor.sporsem@sintef.no (T. Sporsem).

[1] It is worth noting that a detailed software requirements specification also served as a contract between the client and the supplier (Abbott and Moorhead, 1981)

https://doi.org/10.1016/j.jss.2025.112693
Received 2 May 2025; Received in revised form 22 October 2025; Accepted 29 October 2025
Available online 13 November 2025
0164-1212/© 2025 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Cockburn phrased it in 1998: "user stories are a promise of a conversation" (Shore and Warden, 2021). These conversations are discussions that help developers gain a deeper understanding of the requirements (Kassab, 2015), support them in ensuring that the requirements are *right* (Lucassen et al., 2016a), and to estimate the implementation effort (Mahnič and Hovelja, 2012). User stories, thus, seem to be effective in supporting knowledge transfer among different stakeholders. However, despite their widespread use, user stories have been critiqued on several points. Researchers have pointed out that they can be ambiguous, leading to misunderstandings between developers and users (Amna and Poels, 2022a). They are also seen as inadequate for capturing and documenting non-functional requirements (Behutiye et al., 2017), and for risking obscuring critical interdependencies between system components by ignoring holistic architectural design (Meyer, 2014, 2018). Thus, it appears that there are a number of tensions or conflicts surrounding the use of user stories in practice.

Now, since the publication of Cohn (2004b) seminal work on user stories more than 20 years ago, several studies of user stories in practice have been published. Yet, we observe a lack of synthesis that can help our understanding of how the tensions between benefits and shortcomings of user stories described above can be resolved. This absence of synthesis is particularly striking given the widespread adoption of user stories in industry. Thus, we set out to address the following research questions:

Research Question 1: Which research communities have generated empirical findings on the industry use of user stories?

Research Question 2: In what industry contexts have user stories been studied empirically?

Research Question 3: How do user stories facilitate knowledge integration between stakeholders in software development?

To answer these, we conducted a theoretical literature review (Paré et al., 2015) of user stories. The main goal of a theoretical review is to develop a conceptualization of prior literature, for example, the development of a set of research propositions (Paré et al., 2015). It brings together different streams of research, and helps discover patterns and commonalities (Paré et al., 2015). As a starting point, we adopt Tiwana (2004)'s view that software development is a knowledge intensive process that requires the coordination and integration of different types of specialized knowledge. Tiwana (2004) shows evidence that knowledge integration has a significant effect on software development performance. We draw on boundary object theory (Star and Griesemer, 1989) as a theoretical foundation, and posit that user stories are boundary objects that can help to address the knowledge integration problem.

This article is structured as follows. First, we provide a historical overview of traditional RE, the emergence of Agile RE, and boundary objects theory (Section 2). After laying out the review method (Section 3), we present the theoretical literature review that synthesizes the empirical findings from 14 industry studies of user stories, resulting in a set of theoretical propositions (Section 4). Finally, we view our findings through the lens of boundary object theory by considering three types of knowledge boundaries: syntactic, semantic, and pragmatic. Based on this analysis, we identify new research questions to guide future research (Section 5). We conclude this article in Section 6.

2. Background

User stories were first introduced in software engineering (SE) practice in the late 1990s and early 2000s. In this section, we explain what user stories are, outline their origins and history, and review how they have been studied in prior research. We then introduce boundary object theory, and explain how user stories are boundary objects that can help address the knowledge integration in software development.

2.1. User stories

User stories are short, informal written texts that represent user needs, and first appeared as part of Scrum (Cohn, 2004b). Practitioner literature explains what user stories are (Leffingwell, 2010; Cohn, 2004b): user stories are meant to be written from a user's perspective so that users themselves can understand the stories and participate in their prioritization. User stories are frequently written by a customer representative such as a product owner, who has the primary responsibility for prioritizing and refining them. Other stakeholders can also contribute to their creation, such as end users, developers, business analysts, and testers. Stories can be recorded in different ways, ranging from simple notes written on post-its and placed on a board, to entries in sophisticated project management tools such as Jira. A common format of a story is the 'Connextra' template, so named after the UK company where it was first proposed by Rachel Davis (Cohn, 2004b):

> "As a ⟨role⟩, we want ⟨goal⟩, so that ⟨benefit⟩"

which encourages developers to consider the user's role. The following is an example of a story for a photo sharing website:

> "As a user, I want to upload pictures, so that I can share them with my friends."

A story is then placed in a backlog, broken down into specific tasks such as coding, testing, and database development, and assigned to someone responsible for its implementation. This refinement may involve horizontal or vertical slicing depending on the team's approach. The idea is that stories do not contain all needed details; instead, it becomes a "promise for a future conversation" between developers and users (Shore and Warden, 2021). Through these conversations, the story evolves throughout development.

User stories are prioritized by a user representative so as to determine which stories are to be implemented in an upcoming iteration; within Scrum, this happens during sprint meetings (Hummel et al., 2015). This ensures that the features delivering the most value are developed first, and placed at the top of the backlog (Cohn, 2004b). User story mapping can help maintain an overview during this prioritization process, since it is easy to lose sight of the overall system when working with a backlog that is essentially a long list of user stories (Patton and Economy, 2014). As Patton and Economy (2014) points out, the real value of story mapping lies not in the artifact itself, but in the conversations it generates and the shared understanding it creates among stakeholders.

Initially, user stories are often incomplete and evolve as developers gain a deeper understanding of user needs through trial, error, and experimentation (Leffingwell, 2010). This process is termed user story refinement, and several tools exist to support it (Cohn, 2004b). For example, the use of acceptance criteria to explicitly define conditions under which the story is considered complete was proposed in one of the earliest publications that describe user stories (Jeffries, 2001). Another user story refinement method is the INVEST framework, which supports developers in checking if user stories are Independent, Negotiable, Valuable, Estimable, and Small. In a survey study, 23.5 % of developers reported using the INVEST framework, 33 % relied on self-defined guidelines, and 39.5 % did not validate their user stories using any guidelines (Lucassen et al., 2016a). Since user stories are often refined and change over time—and new ones emerge—product backlog prioritization should typically be revisited every 1–4 weeks during sprint meetings (Cohn, 2004b). At the end of each iteration, a demonstration is held to present the software increment developed during the sprint, allowing for user feedback.

User stories can appear at various levels of granularity. Epics, features, themes, and initiatives represent higher level abstractions that organize related stories. An epic is a large user story that can be broken down into smaller stories (Cohn, 2004b). Epics are usually too large

to estimate, prioritize, or implement within a single iteration. Initiatives exist above epics and refer to concrete programs of work that are strategically important and often time-bound (Leffingwell, 2010). A feature is a user story large enough to deliver standalone value to the user but typically smaller than an epic (Leffingwell, 2010). Themes are used to group related initiatives, epics, features, and user stories to create a higher-level overview and help organize the backlog (Cohn, 2004b). It is important to note that there appears to be no clear consensus on the definitions or use of these terms, and they tend to appear inconsistently across different sources. Practitioners likely adopt and adapt the terminology to their organizational context and preferences.

User stories express functional requirements, that is, requirements that are valuable and visible to stakeholders. Non-functional requirements, also referred to as quality requirements, describe how well the system must perform its functions (Glinz, 2007). For instance, they encompass aspects such as system response time. Unlike functional requirements, they are typically not deliverables that users directly perceive but are instead constraints that shape how the system operates (Cohn, 2004b). To address these requirements, Cohn recommends documenting them as "constraints," while Kniberg (2015) advocates for capturing them as "tech stories" and maintaining them on a separate list. Despite these strategies, agile software development has faced criticism for its inadequate handling of non-functional requirements, partly due to its strong emphasis on rapidly delivering functional features, and user stories have been reported to fall short in effectively capturing and specifying non-functional requirements (Behutiye et al., 2020).

User stories have also been criticized for not adequately specifying system requirements. Meyer (2018) argues that user stories provide an incomplete approach to system design; as he points out:

> "a user story describes one case; piling up case over case doesn't give you a specification. What it gives you is a system that handles the user stories—the exact planned scenarios—and possibly nothing else."

According to Meyer (2014), by emphasizing incremental feature implementation over holistic architectural design, user stories risk obscuring critical interdependencies between system components. While individual user stories may appear coherent in isolation, Meyer warns that subsequent stories can introduce unexpected conflicts with previously implemented features. However, this reflects a misunderstanding of the purpose of user stories, which were never intended to fully describe or specify an entire system. Instead, as Cockburn explained, they serve as "promises of conversations," designed to support shared understanding and provide contextual information. Furthermore, practices such as user story mapping and the use of higher-level abstractions like epics and themes address some of the concerns Meyer raises, but were not considered in his critique. The scaled agile frameworks also address this topic (Edison et al., 2022; Carroll et al., 2023). At the same time, Meyer's concern is related to a broader discussion about traditional software architecture approaches and the Agile Manifesto, which emphasizes emergent architecture (Nord and Tomayko, 2006; Abrahamsson et al., 2010).

Lastly, Amna and Poels (2022a) highlight that user stories can be ambiguous, drawing on findings from multiple studies in their literature review. In this context, ambiguity refers to unclear or imprecise descriptions of requirements that can be interpreted in multiple ways. Such problems often stem from inconsistent ways of articulating requirements or differences in domain knowledge. For example, they present two user stories that at first appear to be duplicates, but, upon closer inspection, may actually describe different system functions. The distinction, however, is not clearly stated, which leaves developers with the task of resolving the ambiguity. According to Amna and Poels (2022a), sources of ambiguity include varying levels of domain knowledge, inconsistent use of terminology, or simply being careless when writing user stories. We argue that these sources of ambiguity are all related to communication among stakeholders, which Tiwana (2004) refers to as knowledge integration.

2.2. The history of user stories

User stories evolved from use cases, which were first introduced by Jacobson in 1987 within the context of object-oriented programming (Spurrier and Topi, 2022). Use cases gained widespread attention in the early 1990s following the publication of Jacobson (1993)'s book. Their popularity was further reinforced when they became an essential part of the Unified Modeling Language (UML). Unlike user stories, use cases are typically associated with plan-driven software engineering and provide a formal, up-front detailed description of functional requirements (Spurrier and Topi, 2022). When Beck (1999) wrote his seminal article on Extreme Programming (XP) in 1999, he acknowledged Jacobson's influence, describing a story as "the amount of a use case that will fit on an index card" (Spurrier and Topi, 2022).

User stories originated from the Chrysler C3 project in Detroit, an attempt to replace several payroll systems with a single system. Beck (2000) described the XP development method used in the project, including "user story cards." Jeffries (2001), who was Beck's co-founder of the XP method, introduced the concept of "Card, Conversation, Confirmation," emphasizing that stories should be written on cards to *initiate* discussions between customers and developers rather than to document detailed requirements. Cohn (2004b) wrote the first practitioner book on user stories and published the first paper on user stories to appear in an academic publication venue at the 2004 Extreme Programming and Agile Methods—XP Conference (Cohn, 2004a). In the following years, the XP conference became the most important arena for academic publications on user stories (Śmiałek et al., 2005; Ludlow, 2007; Chubov and Droujkov, 2007; Kautz, 2009; Klärck et al., 2009; Gomez et al., 2010; Power, 2011).

User stories only began to appear in RE conferences in 2010 (El-Attar and Miller, 2010), suggesting a lag of 5 to 10 years behind the agile community. The RE field has seen significant growth in interest starting around 2015 (Harbers et al., 2015; Liskin, 2015; Lucassen et al., 2016b), and user stories are now a common topic at RE conferences. Still, there is ongoing debate among RE researchers about whether user stories should be considered requirements (Dalpiaz and Steghöfer, 2024). In contrast, practitioners often adopt them because they "just work" in practice (Dalpiaz and Steghöfer, 2024). Although user stories have attracted attention in both the Agile and RE communities, they remain a marginal topic in leading software engineering venues such as the International Conference on Software Engineering (ICSE), and top journals such as IEEE Transactions on Software Engineering (TSE). Spurrier and Topi (2022) offer a more detailed description of the history of use cases and user stories.

Today, user stories have become ubiquitous in the software industry. A survey by Wang et al. (2014) found that user stories were the most widely used technique for capturing requirements in software development projects. Another survey of 182 practitioners by Lucassen et al. (2016a) revealed that user stories are especially dominant in organizations adopting agile methodologies, with 99 % of Scrum teams, 79 % of Kanban teams, and 83 % of Extreme Programming (XP) teams employing user stories. According to the 17th State of Agile Report (2023), 71 % of respondents incorporate agile methods in their software development life-cycle. If we apply the estimate that 90 % of these agile developers implement user stories (Wang et al., 2014), approximately 63.9 % of global developers use user stories, roughly aligning with Kassab (2015)'s survey which found it to be 50 %. This highlights user stories' significant influence on software development practices.

2.3. From traditional requirements engineering to agile requirements engineering

To understand why user stories emerged, we need to look back at the evolution of RE, an area of interest that acquired momentum in the early 1990s, well before agile methods became popular. Traditionally, RE focused on gathering and documenting system requirements up

Table 1
Abstractionist vs. contextualist perspectives in requirements engineering.

	Abstractionist	Contextualist
Goal	Develop simplified, formal representations of systems and domains	Achieve a deep understanding of the real-world context in which the system operates
Focus of inquiry	Generalization, abstraction, and formal structure	Specificity, situatedness, and detailed contextual knowledge
Underlying assumption	Systems can be effectively specified and understood through abstract formal models	Requirements cannot be fully understood without preserving the specific context of use
Typical methods	Formal modeling (e.g., goal models, dependency graphs), structured specifications	Qualitative fieldwork (e.g., interviews, ethnography, observation), contextual inquiry
Strengths	Clarity, precision, analyzability, and potential for automation	Rich detail, stakeholder relevance, adaptability to real-world complexities
Drawbacks	Risk of oversimplification and detachment from stakeholder realities	Limited generalizability, harder to formalize, time-intensive data gathering
Output	Structured, formal artifacts that can be verified and validated systematically	Narrative accounts, design rationales, and rich descriptions that inform design decisions

front, aiming to create a solid knowledge base that would guide developers in building software to meet user needs (Sommerville and Sawyer, 1997). This approach was based on the assumption that requirements could be fully elicited and captured in documentation (Ralph, 2018), in full alignment with plan-driven software development approaches.

A key activity in RE is the modeling of requirements: the construction of abstract descriptions of the software to be made (Nuseibeh and Easterbrook, 2000). Models serve as key interfaces in all RE activities and provide the foundation for further RE work. Various approaches to modeling have been a major focus of research. Initially, models were purely technical, starting in the 1970s (Ross and Schoman, 1977), and was first used by MIT and the US Air Force (Ross, 1978). In the 1980s, goal modeling emerged, emphasizing the objectives the software must achieve (Van Lamsweerde, 2001; Yue, 1987). Later, in 1999, UML was introduced to bring together various modeling languages into one standardized framework (Van Lamsweerde, 2000). Despite these developments, practitioners criticized large and complex requirements models as impractical and hard to use, calling instead for simpler methods and tools (Ambler, 2003). In response, the 1990s saw the rise of scenario-based elicitation, which aimed to handle the difficulty of modeling complex realities (Van Lamsweerde, 2000). This approach focuses on describing hypothetical scenarios of how a future system should perform in different situations. The reasoning is that users often struggle to articulate their needs in abstract terms, but find it easier to understand and express them when framed in concrete, relevant scenarios.

A fundamental disagreement between two schools of thought in the field of RE emerged as a result of the discussion on the difficulties in modeling in the 1990s. On one side were the abstractionists (Potts and Hsi, 1997), who used models, that they claimed to be generalizable across development projects, to describe system requirements. Contextualists, on the other hand, criticized the abstractionists for overgeneralizing (Potts and Hsi, 1997; Hughes et al., 1995; Sommerville et al., 1993). They argued that the complexity of each project context was too intricate to be captured or generalized through models (Nuseibeh and Easterbrook, 2000). Contextualists relied on ethnomethodological approaches, such as participant observation, to uncover important contextual factors that supported the elicitation of requirements (Viller and Sommerville, 1999; Sommerville et al., 1993; Goguen and Linde, 1993). They saw the world as too complex, with too many details, concepts, and relationships to be adequately described explicitly. According to contextualists, this type of knowledge (such as 'unknown unknowns' Sporsem et al., 2023) could only be gained by physically experiencing the context or situation in question. Table 1 summarize the perspectives of the abstractionists and the contextualists.

Agile methods share much in common with the contextualist perspective. Their widespread adoption—from their emergence in the mid-1990s (cf. Schwaber, 1997) to their current dominance in software development—reflects the importance of adapting to each unique context rather than relying on generalized models. The agile movement also acknowledged that requirements cannot be fully specified in advance through documentation. The world contains too much complexity to be captured exhaustively in written requirements. In light of this, user stories—with their concise form—represent a practical response to traditional, document-heavy practices, recognizing that context can never be fully described in writing or models.

In the early 2000s, the agile movement really broke ground in academic communities, pushed by emerging practices of practitioners. Agile methods moved the industry from traditional plan-driven (waterfall) approaches towards iterative and incremental approaches (although these principles were already described decades ago (Larman and Basili, 2003)). In this new approach, requirements engineering no longer ended prior to coding began but continued throughout development, based on the assumption that requirements are never truly "done" (Ramesh et al., 2010). The widespread use of user stories suggests that practitioners find the contextualist perspective more useful than the modeling approaches proposed by the abstractionists.

Several systematic literature reviews (SLRs) have summarized the benefits of agile RE compared to traditional RE: (1) reduced process overhead, because developers spend less time on creating and revising detailed requirements documentation (Heikkilä et al., 2015); (2) better understanding of requirements, as prioritization and validation occur frequently with users or customers rather than just upfront (Inayat et al., 2015; Schön et al., 2017); and (3) greater adaptability to changes in users' and customers' understanding of the problem, the software, and priorities (Inayat et al., 2015).

While agile RE holds benefits over traditional RE, the same literature reviews highlight that it also introduces new challenges. A major issue is the reliance on tacit knowledge, which becomes problematic when team members leave, taking their undocumented knowledge with them (Inayat et al., 2015). In their absence, teams often need to rediscover missing information, leading to rework. Another challenge arises with geographically distributed teams, where coordination and communication are harder compared to co-located teams (Curcio et al., 2018). Since tacit knowledge is typically shared through direct collaboration and shared experiences, distributed teams face additional difficulties in working effectively (De Souza Santos et al., 2024). Furthermore, agile RE relies on regular access to users for validating requirements and code, but users are often unavailable or unable to commit sufficient time, leading to more rework (Inayat et al., 2015). Additionally, user stories and product backlogs, two primary forms of documentation in agile RE, are often found inadequate (Schön et al., 2017). Changes in requirements, tests, and code are frequently undocumented, complicating traceability and backtracking (Schön et al., 2017).

2.4. Previous literature reviews of user stories

A small number of reviews have summarized prior work on user stories (see Table 2), but none specifically address the use of user stories in practice. Raharjana et al. (2021)'s SLR on the use of natural language processing (NLP) techniques on user stories found that most studies used

Table 2
Previous reviews of user story research.

Study	Scope	Findings
Raharjana et al. (2021)	Systematic literature review of NLP research on user stories (n = 718) with a synthesis of n = 38 studies	Capturing contextual information surrounding user stories is challenging because user stories are typically articulated as one written sentence, leaving important context knowledge tacit in the minds of their authors and inaccessible to NLP technologies.
Amna and Poels (2022a)	Systematic literature review on ambiguity in users stories (n = 551) with a synthesis of n = 36 selected papers	Ambiguity in user stories come in four classes: vagueness, inconsistency, insufficiency, and duplication. Researchers have addressed ambiguity by proposing three solutions: conceptual modeling, algorithmic approaches, and better writing techniques, such as templates or frameworks
Amna and Poels (2022b)	Mapping study of n = 186 papers on user story research	80 % of papers focus on creating solutions for user story challenges, while only 20 % focus on problem investigation or explanation of observed phenomena
Kustiawan and Lim (2023)	Systematic literature review on user stories in requirements elicitation (2018–2023, n = 239), with n = 40 selected papers	75 % of publications on user stories were conference publications, 17.5 % journal papers, and 7.5 % workshop papers. A list of 24 problems surrounding user stories is presented and a list of nine solutions to these problems

NLP to extract the 'who,' 'what,' and 'why' aspects of user stories for purposes such as defect detection, software artifact generation, abstraction identification, and creating models out of user stories. A key finding was that capturing contextual information is challenging because user stories are typically brief, leaving important context knowledge tacit in the minds of their authors and inaccessible to NLP technologies. Indeed, NLP techniques cannot capture the conversations that are "promised" and yet to be had (see Section 2.1).

Through a mapping study of 186 papers, Amna and Poels (2022b) revealed that 80 % of the studies on user stories are solution-oriented, proposing tools, methods, or procedures to address challenges in user stories, while only 20 % focus on problem investigation (cf. Wieringa et al., 2006) or explanation of observed phenomena. They categorized user story research into three topics:

- Ambiguity: Issues in articulating requirements as user stories, leading to unclear or multiple interpretations. These arise from language differences, template limitations, and varying domain knowledge (2 % of studies)
- Collaboration: Problems in teamwork and stakeholder involvement, such as lack of validation, conflicts of interest, or stakeholder disengagement. These stem from how user stories are used, rather than the technique itself (26 % of studies)
- System Design: Challenges related to dependencies between user stories, requirement complexity, and architectural alignment. High-quality user stories are essential for ensuring the system meets requirements (52 % of studies)

Amna and Poels (2022b) provide little insight into the proportion of empirical studies in industry settings or the findings of the mapped literature. However, Amna and Poels (2022a)'s SLR of 36 studies focused on ambiguity in user stories. Their analysis categorized ambiguity into vagueness, inconsistency, insufficiency, and duplication. They identified three solution types proposed by researchers: conceptual modeling, algorithmic approaches, and improved writing techniques such as templates or frameworks. They conclude by emphasizing the need for further research on how human behavior contributes to ambiguity in user stories.

Lastly, Kustiawan and Lim (2023) conducted a study covering the period from 2018 to 2023, identifying publication channels, challenges in using user stories, and potential solutions. While described as an SLR, the study is better characterized as a mapping study, providing an overview of publication channels and summarizing findings rather than offering a detailed synthesis.

While our theoretical literature review aims to identify gaps and offer new perspectives, we wish to emphasize that the most effective way to study how user stories are applied in practice is through empirical research. This review differs in its methodology; rather than an SLR or a mapping study, ours is a theoretical review that seeks to integrate prior work into a conceptual framework or model. Reviewing and synthesizing previous empirical studies is essential—not only to guide future empirical work, but also to inform theory development. A solid understanding of what has already been found, and where the knowledge gaps lie, is necessary both for designing meaningful empirical studies and for advancing theory. This review seeks to support both aims: it contributes to theory building and provides a foundation for future empirical studies on user stories.

2.5. User stories as boundary objects

Software development can be understood as a knowledge integration challenge (Tiwana, 2004). On one side, there are domain experts who have domain knowledge about business needs, constraints, processes, and end user needs. On the other side, there are software developers with technical knowledge covering software development methodologies, testing, debugging, programming languages, design constraints, etc. (Tiwana, 2004). In order to succeed, developers must combine knowledge from both fields. This integration does not happen automatically, but requires specific facilitation and support (Carlile, 2002).

Consider the following example user story:

> "As a customer, I want to save items to a wishlist so I can purchase them later."

This story means different things to different stakeholders:

- Business stakeholders see potential revenue and customer retention
- Developers see database structures and API endpoints
- UX designers see interface elements and user flows
- Testers see validation scenarios and edge cases

Yet, despite these varying interpretations, the user story maintains its coherence as a shared reference point, allowing these stakeholders to coordinate their work effectively.

User stories capture requirements in a format that both domain experts and software engineers can understand and use, despite their different perspectives and areas of expertise. For example, in Fig. 1,[2] a product owner or business analyst may focus on the business value offered by the story, developers may concentrate on the technical implementation, and testers may look at validation scenarios—yet all are working with the same shared artifact. This mirrors (Star and Griesemer, 1989)'s concept of boundary objects, which they described as entities that maintain a common identity across boundaries of different

[2] This is a simplified illustration. The Domain Expert field may include roles such as users, business analysts, and managers, while the Software Engineering field may include roles such as testers and UX designers. In reality, both fields can be divided into several smaller subfields, each separated by its own knowledge boundaries. Thus, the figure presents a more aggregated view than what exists in practice.

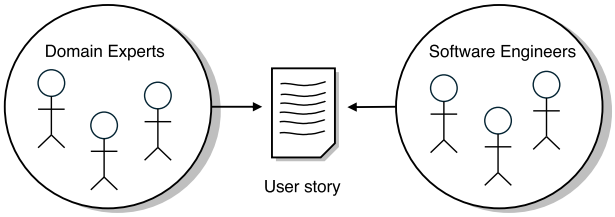


Fig. 1. How user stories function as boundary objects.

social worlds while being specific enough to be useful within each one. This adaptability to different contexts while maintaining a recognizable core identity is what Star (2010) identified as a key characteristic of boundary objects.

Boundary object theory has been applied in several studies within the SE and IS fields, extending beyond user stories. For example, O'hEocha and Conboy (2010) argued that user stories act as boundary objects that support innovation, while Sedano et al. (2019) interpreted the product backlog as a boundary object. Other studies have used boundary object theory to analyze knowledge sharing across disciplinary boundaries in cross-functional teams (Matthies et al., 2023; Barrett and Oborn, 2010). Wohlrab et al. (2019) conducted an extensive study of engineering artifacts in large-scale agile automotive projects, using boundary object theory to examine related challenges, guidelines, and practices. Similarly, Pareto et al. (2010) conceptualized architectural documentation as a boundary object between system developers and engineers in telecommunication projects. Through case study research, Matthies et al. (2023) identified four categories of boundary objects in DevOps projects: requirements, deployed system information, process checklists, and flexible-format artifacts. Finally, Miller et al. (2012) analyzed goal models as boundary objects, showing how they help bridge understanding between end users and developers.

2.6. Crossing knowledge boundaries

Viewing user stories as boundary objects implies that they help groups cross boundaries between them when collaborating. Carlile (2002) presents three types of knowledge boundaries that boundary objects support in crossing.

A *syntactic* knowledge boundary is the most basic type of knowledge boundary that exists between different groups or disciplines within an organization. It arises when different groups use varying vocabularies or terminologies to describe their work. To overcome this boundary, a shared language or terminology (i.e., a common syntax) must be established between groups. Carlile (2002) illustrates this concept with an example of engineers from different departments needing to communicate about parts and specifications. To transfer information effectively, they first require a shared technical vocabulary, including standardized part numbers, measurement units, and technical terms.

A *semantic* knowledge boundary arises when different groups have varying interpretations or understandings of the same things, even when they share a common terminology. Carlile (2002) exemplifies this with engineers who interpreted a design differently despite having established a common vocabulary. One engineer saw it as a successful design for highly complex functions, while another viewed it as problematic for mass manufacturing. Although the engineers could read the same design, they interpreted it differently.

A *pragmatic* knowledge boundary exists when different groups have different interests and consequences at stake in their work. At this boundary, stakeholders often hold conflicting interests, where changes proposed by one group create negative consequences for another. Recalling the example above, the design engineer had invested time in creating a working prototype that met functional requirements, while the manufacturing engineer recognized that the same design would cause major problems in production. To resolve this conflict, the design engineer needed to transform his knowledge of functional design into knowledge that also incorporated manufacturability constraints. Both perspectives were valid but competing, shaped by the specialized roles of each engineer.

The terms syntactic, semantic, and pragmatic are recurring in the RE literature. They have previously been applied to evaluate the quality of models (Lindland et al., 1994) and to develop quality assurance frameworks for writing user stories (Lucassen et al., 2016b). In this article, we adopt these terms in line with Carlile (2002), to study how user stories might help bridge different types of knowledge boundaries.

3. Method

We now outline the method used to identify and analyze existing literature on user stories in industry practice. Our aim was not only to review empirical findings but also to support theorizing about how user stories function in practice. The resulting theoretical propositions provides a coherent starting point for future empirical studies.

3.1. Search strategy and paper selection

Given that user stories have been widely used by developers in building our society's information systems for more than twenty years, one might expect a significant amount of research publications and theories that explain their use in depth. To address the research questions, we conducted a theoretical literature review (Paré et al., 2015). The primary aim is to develop a conceptual framework or model accompanied by research propositions or hypotheses. Theoretical reviews are valuable both for emerging issues, where new theoretical foundations are needed, and for mature topics, where existing research is abundant but theories are either absent or inadequate to address persistent problems (Paré et al., 2015). Their contribution lies in extending or generating new conceptualizations by identifying and emphasizing knowledge gaps between what is currently known and what still requires exploration (Paré et al., 2015).

Unlike SLRs, theoretical reviews might lack a formal quality assessment protocol. Some scholars argue that a formal appraisal of quality is unnecessary, whereas others maintain that papers that are conceptually or methodologically flawed should be excluded altogether (Paré et al., 2015). In our study, given the absence of standardized quality assessments for theoretical reviews, we grounded our selection process in established, high-quality publication sources. We decided on using the Web of Science database, one of the leading databases for software engineering research. Additionally, we included the Association for Information Systems (AIS) library to cover Information Systems (IS) publications. The inclusion of the AIS library allows for the inclusion of different streams of research (Paré et al., 2015); while there is considerable overlap in topics between IS and SE research, these communities tend to employ different approaches, including different emphasis on theory development (Fitzgerald, 2024). A potential consequence of this is that we might miss some high quality papers that are published in lesser known journals and conferences. However, all leading SE journals, including the RE Journal, are included in Web of Science. The main agile and RE conferences are also included in Web of Science, such as the AGILE conference, the XP conference, the RE conference, and the REFSQ conference series.[3] This search strategy allowed us to claim that the findings in this review originate from rigorous research methods.

To further ensure that the selected papers were of sufficient rigor, we applied a set of evaluative criteria inspired by Greenhalgh and Dijkstra (2024). Each paper was reviewed to verify that it included an adequate description of its research methods and that the chosen analytical approaches were appropriate and grounded in the collected data. We also

[3] REFSQ: International Working Conference on Requirement Engineering: Foundation for Software Quality

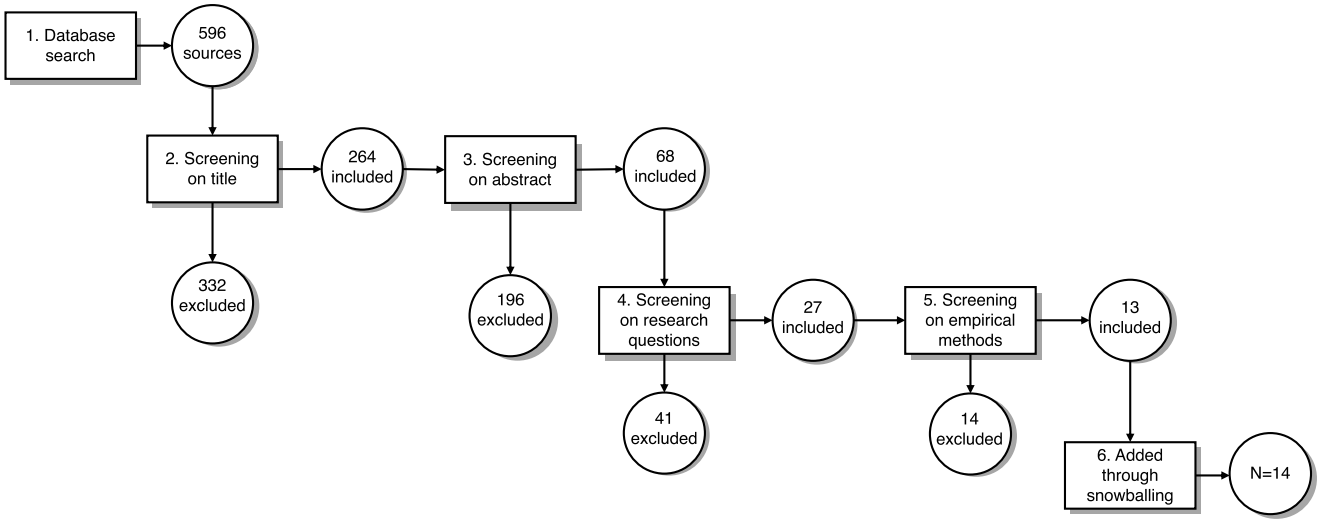


Fig. 2. Paper selection process.

assessed whether the studies employed suitable and well-described data collection methods, and whether their research designs aligned with the stated aims of the study. Furthermore, we examined whether the aims and objectives were clearly articulated, including the rationale for conducting the research, and whether the findings were presented in a clear and credible manner with justified conclusions. Together, these considerations provided confidence that the included studies met a reasonable standard of methodological and reporting quality.

We excluded gray literature in this review. While gray literature can offer valuable practitioner perspectives, it does not receive independent scrutiny in a peer-review process, which serves as an important safeguard for methodological transparency and evidentiary quality. Because its methodological foundations are often unclear, it is difficult to assess the reliability and replicability of findings. Many non-peer-reviewed sources provide minimal information on research design, data collection, or analysis, which limits their suitability for theorizing. Without a transparent methodological account, it becomes challenging to evaluate the strength of the evidence or place it in dialogue with other empirical studies. By relying solely on high-quality empirical studies that have passed this criterion, the propositions developed in this paper rest on a consistent and verifiable evidential basis.

We used the keywords "user stor*" and limited the search to "topic" (title, abstract, keywords), within the categories of "Software Engineering" and "Information Systems", which yielded 596 texts, of which 202 were journal publications.

To understand what research has revealed about user stories in industry settings, we excluded all papers that did not meet the following inclusion criteria:

- The study must be empirical;
- User stories must be the object of study, or there are significant findings on user stories;
- The study must be conducted in an industry setting or involve practitioners with industry experience;
- The study must examine developers that were already using user stories as part of their natural setting. We excluded studies in which researchers introduced an intervention, such as introducing new tools, training developers and observing their use over time, as these are closer to validation studies than studies of existing practice.

These specific criteria ensured we only included studies of user stories in their "natural setting" (Stol and Fitzgerald, 2018): contexts where researchers have not deliberately influenced the studied cases, such as by convincing a company to validate a particular method or tool. This is not to suggest that intervention, design science studies, or validation studies lack value. Rather, our objective is to develop an understanding of how user stories function in industry settings that remain unmodified by researcher intervention, providing insight into their authentic impact, rather than offering suggestions as to how to improve the use of user stories.

Fig. 2 shows the selection process. We screened the in total 596 titles,[4] excluding those that were clearly off-topic, reducing the total selection to 264 texts. We then reviewed each abstract, further narrowing the selection to 68 texts. These initial two screening steps (on title and abstract) were conducted independently by two investigators.

After analyzing the research questions and findings of each text, we reduced the pool to 27. Finally, we examined the research methods used, excluding any papers that lacked rigorous empirical methods, resulting in a selection of 13 texts. To address the concern that we may have missed relevant studies, we performed backward snowballing (Wohlin, 2014) on the selected texts with the result of adding one additional text, resulting in a total of 14 texts. Lastly, we thoroughly cross-checked all the identified secondary studies on user stories (see Sec.2) to ensure no rigorous empirical studies have been overlooked (Amna and Poels, 2022a,b; Kustiawan and Lim, 2023; Raharjana et al., 2021).

Some studies initially appeared to meet the inclusion criteria but were excluded upon closer examination. One such study, by Müter et al. (2019), analyzed tasks rather than user stories, leading to its exclusion. Similarly, Bik et al. (2017) described the development of a reference method for user stories but did not report findings on user stories themselves. And, Rocha Silva et al. (2020) investigated software users from an industry case, but the study was akin to an experimental simulation (Stol and Fitzgerald, 2018), where participants were asked to write user stories for a fictional system that was never developed, which makes it both contrived and intervention-based.

During the process of excluding papers, we observed several trends. Papers from the RE community tended to focus on two main areas: (1) developing models from user stories—often using AI technologies such as natural language processing—to support model-driven development (e.g., Robeer et al., 2016; Güneş and Aydemir, 2020; Umar and Lano, 2024); and (2) creating tools to improve the semantic quality of user stories, aiming to better document user needs, often also using AI technologies (e.g., Dalpiaz et al. (2018), Lucassen et al. (2015), Raharjana et al. (2021)). Many of the excluded studies were validation studies of

[4] For transparency, a stepwise list of the excluded papers is available in the OSF repository: https://osf.io/kbvcs/?view_only=b71bce502b4f4448bbfd7b3b5165c15e

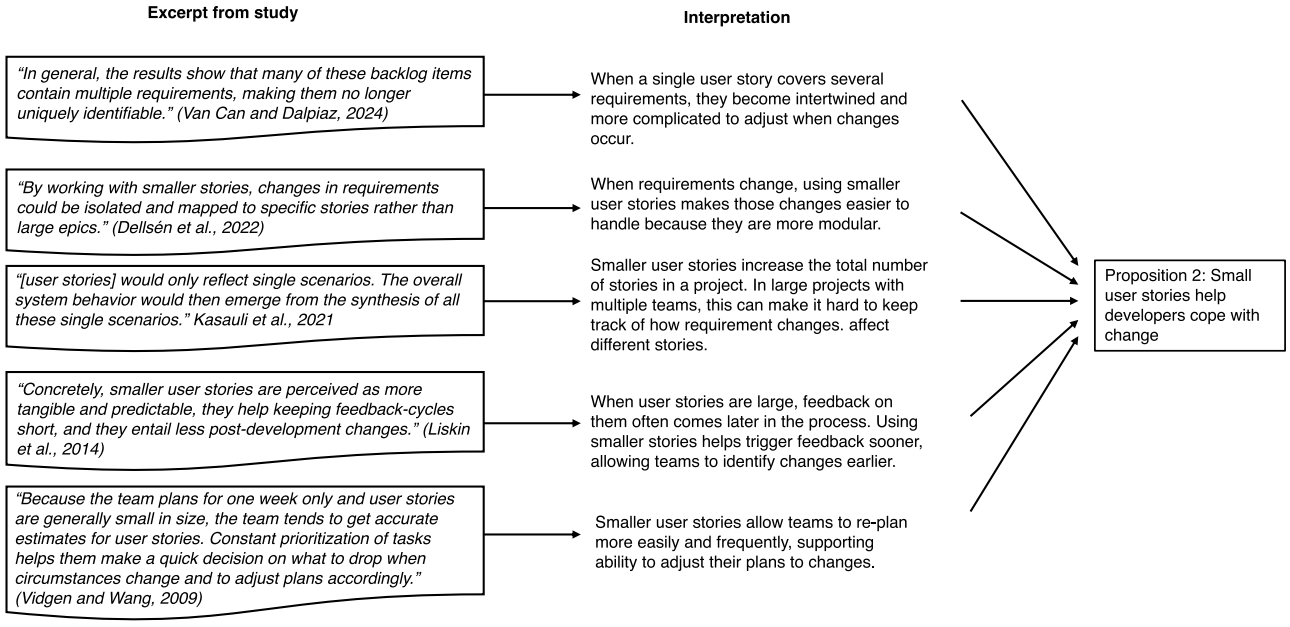


Fig. 3. Example of proposition development based on excerpts from selected studies.

tools, methods, or frameworks. These validations varied in approach, from involving students in quasi-experiments to encouraging companies to test the methods in simulations and workshops. A recurring sentiment in these studies was that user stories should be made more detailed, for example, by augmenting them with use cases or by using AI to generate "better user stories." While this is an interesting notion, it does seem to go against the initial concept of user stories as a "promise of a conversation" (see Section 2). Lastly, several laboratory experiments with students have been carried out on user stories (see, for example, Dalpiaz et al., 2019; Mahnič and Hovelja, 2012; Bragilovski et al., 2022); given the contrived research settings (Stol and Fitzgerald, 2018), we excluded those studies.

3.2. Developing theoretical propositions

Scholars have emphasized that developing and testing theories is essential for advancing scientific knowledge about software engineering processes (Conboy, 2009; Sjøberg et al., 2007). We share this view and aim to contribute by using this theoretical literature review to theorize about user stories. In theory-building, statements form the basic building blocks (Stol, 2024). Among these, propositions are a specific type of statement that assert a claim, express a truth, or make a prediction about a concept. In this review, we develop a set of propositions as an initial step toward constructing what Sandberg and Alvesson (2021) refer to as a comprehending theory. The goal of a comprehending theory is to develop a deeper understanding of an organizational phenomenon. For example, to assess what constitutes trustworthy behavior, one must first understand what trust is. Similarly, to lead effectively, one must understand what leadership entails. Researchers can build this understanding by studying organizations and providing descriptive accounts of their experiences. However, to go beyond mere description, researchers must offer deeper insights—such as interpreting the phenomenon through discourse, metaphors, or other rhetorical devices. The purpose of such theories is to introduce a new way of viewing a phenomenon, offering a fresh perspective that enhances a field's understanding.

To contribute to a deeper understanding of user stories, we go beyond merely describing and summarizing findings from the selected literature. We do so by developing a set of propositions that are grounded in the empirical findings from the selected papers, allowing for the identification of contrasts, commonalities, and potential relationships between concepts. By examining findings in relation to one another, meaningful theoretical insights can emerge. However, not all propositions are equally useful, and efforts were made to avoid trivial or redundant propositions that offer limited insight. Dubin (1978, p. 168) argued for parsimony by focusing on strategic propositions, and to ignore the trivial that do not add much insight. We focused on notable findings—those that were surprising, grounded in interesting situations, or hold practical relevance for software developers.

We adopted the constant comparison method to synthesize the propositions (Seaman, 1999; Miles and Huberman, 1994). First, we assigned labels to text segments that addressed specific themes within the topic of user stories. These labeled passages were then compared and grouped into broader patterns, which we analyzed to identify underlying themes and explanations. Fig. 3 presents an example of how excerpts from the selected studies were noted, interpreted, and how these led to the development of a proposition. During this process, we wrote memos to articulate preliminary propositions, which were iteratively discussed among the authors, refined, modified, and eventually evolved into Sections 4.1 to 4.5. Thus, the propositions are emergent and grounded in the literature rather than preconceived. Through this process, an initial set of nine propositions was distilled into five, emphasizing those with the strongest theoretical and practical significance. To ensure robustness, candidate propositions for which we did not find sufficient evidence were discarded; each of the remaining propositions was supported by at least three distinct sources spanning different contexts and research methods. While these propositions are grounded in empirical findings from the reviewed literature, we anticipate that future studies might extend, refine, or even challenge them.

4. A theoretical review of user story literature

We now present a synthesis of the findings from the selected 14 papers in the form of five theoretical propositions (see Fig. 4). Table 3 provides details on the selected papers, including their study design, the geographic regions where they were conducted, the industry domains involved, the size of the organizations, and the types of software systems studied. We used Meyer (2013)'s ABC characterization to distinguish software types: 'Acute', which includes life-critical and mission-critical software such as medical device software and aviation control systems; 'Business,' which are critical to keep organizations running, and

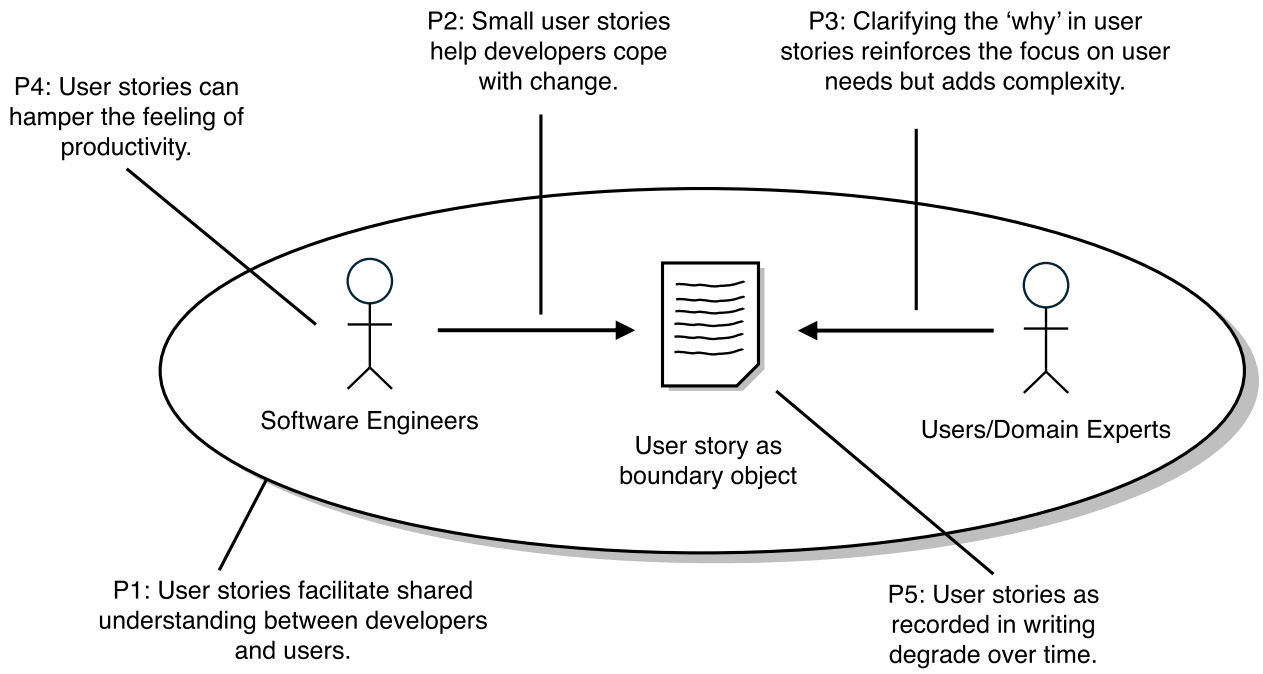


Fig. 4. A graphical summary of the propositions.

'Casual,' which includes software that is not mission or business critical, such as games and other entertainment software. While this classification is very high-level, it does give an approximate indication of the type of software. Table 4 presents an overview of which primary studies support each proposition.

4.1. *Proposition 1: User stories facilitate shared understanding between developers and users*

User stories were not intended to provide a complete description of requirements. Instead, as Cockburn described, they act as a "promise of a conversation" to take place between developers and users—crossing their boundaries. The studies in this review highlight that these conversations help create a shared understanding of user needs. However, the studies also reveal that user stories alone are insufficient; creating a shared understanding also requires human skills in writing and telling the stories.

Using user stories involves developers engaging in discussions to interpret what should be built. This process builds a shared understanding between stakeholders. In their survey study, Lucassen et al. (2016a) found that creating user stories requires stakeholders to break down requirements into "small, comprehensible chunks." This decomposition forces developers and other stakeholders to think and talk about the specifics of a requirement, thereby building a shared understanding of what the user expects from the software (Lucassen et al., 2016a)—they are creating a shared semantic understanding of the articulated words in the user stories. These findings are echoed in Dellsén et al. (2022)'s interview study.

An example of how a user story facilitates shared understanding can be found in Børte et al. (2012)'s study. In their case, a scrum team in a governmental welfare organization was tasked with estimating the story of stopping pension payments when a citizen dies. The initial wording was ambiguous: participants debated what "stop" actually meant in this context, whether it referred to entering a stop code in the system or creating a new task in the case flow. As the discussion unfolded, the team drew simultaneously on the domain knowledge of how case workers handle pensions and the technical expertise of developers in designing system functionality. The user story thus acted as a trigger for conversation, allowing participants to surface different interpretations and negotiate a shared understanding that was sufficient to move forward.

Each successful conversation between users and developers strengthens their shared semantic understanding, and the story itself becomes a *reminder* of these conversations (Sedano et al., 2019). This enables developers and stakeholders to recall and build upon earlier insights, and keep evolving their shared understanding (Sedano et al., 2019). Sedano et al. (2019) describe how user stories are discussed during sprint planning meetings and prioritized as backlog items, adding that "a good specification *replaces* face-to-face communication while a good backlog *facilitates* face-to-face communication."

Additionally, creating shared understanding requires specific skills by those involved, such as strong storytelling abilities. But many organizations tend to value technical skills more highly than storytelling abilities (Conboy, 2021). Effective user stories rely not only on their content but significantly on the skills of the individual writing and conveying them. As one interviewee in Conboy (2021)'s study noted:

> "It is not the user story […] that's useful at all but the person writing and telling the story."

Some story creators are perceived as forward-thinking, "thinking about tomorrow" or "being one step ahead," while others are seen as maintaining "the status quo" or "living in the past" (Conboy, 2021). The level of creativity also varies among story creators—some are recognized as "highly creative," "visionary," or "outside the box" thinkers, whereas others are viewed as unable to envision alternatives to their established experiences (Conboy, 2021). Motivations differ as well, with some individuals writing stories for "the greater good," while others leverage story writing to fulfill their own needs at the cost of organizational needs (Conboy, 2021). Correspondingly, in Ananjeva et al. (2020)'s study, it was observed that when developers found a user story unclear or lacking in information, they often requested a more detailed written story. In response, user representatives (product owners) provided "a twenty-sentence specification" to address these concerns. This approach, however, hindered the intended encouragement of direct, face-to-face communication between developers and users, thereby undermining the fundamental purpose of user stories (Ananjeva et al., 2020).

Seeing a user story as a "reminder" of a conversation embodies an assumption that teams are long-lived; the team can only maintain a shared understanding if its composition remains stable. Without which it becomes challenging to maintain the shared understanding of user stories. In practice, however, team members frequently "come and go," moving

Table 3
Contexts of the selected papers.

Reference	Description	Region	Industry	Organization size	Software type
McAvoy and Butler (2005)	Longitudinal case study of one agile team with seven developers and a project manager. Five interviews and eight months of observations	Ireland	Governmental	NA	Business
Vidgen and Wang (2009)	Multi-case study of one team using XP, and one team using waterfall. Three group interviews, 11 individual interviews, and supplementary documents analysis	One Italian company & one multi-national company	IT consultancy	NA	Business & Casual
Børte et al. (2012)	Case study of two scrum teams. Video recordings of three meetings	Norway	Government	500	Business
Read et al. (2012)	Exploratory case study of one scrum team. Undisclosed number of participatory observations	NA	NA	NA	Business
Liskin et al. (2014)	Survey study of Github users with experience with user stories. Two rounds of questionnaires (n = 68, n = 72)	NA	NA	NA	NA
Lucassen et al. (2016b)	Survey and interview study of developers. Questionnaire (n = 182) and six interviews	Western Europe, South America & North America	Software product, consultancy, custom software companies	5–10,000	NA
Diebold et al. (2018)	Case study of one software team. Observation of one workshop, one demo, and two interviews	Germany	Software	8	Business
Melegati and Wang (2019)	Interview- and design science study. Five interviews with developers	Italy	Software consultancy	3–100	NA
Sedano et al. (2019)	Longitudinal case study of multiple teams using XP. 56 interviews, one questionnaire (n = 21), and document analysis	North America, Western Europe, Singapore	Software consultancy company	2900	Business
Ananjeva et al. (2020)	Action research of two development teams. Five months of participant observations and 32 interviews	Denmark & US	Software-as-a-service	28	Business
Conboy (2021)	Qualitative survey of practitioners with extensive experience with user stories. 29 interviews	All continents	Health, education, finance, electronics manufacturing	200–150,000	Business
Kasauli et al. (2021)	Multiple case study of developers and managers in seven companies doing large-scale agile development. 20 interviews, five focus groups, and eight cross-company workshops	Scandinavia	Telecom, automotive, medical manufacturing	50,000 +	Acute & Business
Dellsén et al. (2022)	Interview study of agile software roles in five companies. 12 interviews and observations of three meetings	Scandinavia, US & Europe	Tourism, financial, IT consultancy, logistics	10–5,000 +	NA
Van Can and Dalpiaz (2024)	Quantitative content analysis of 1636 product backlog items from 14 projects. Data from seven projects in a low-code platform provider and seven projects from two large Open Source issue trackers	US, Australia, Western Europe, East & South East Asia, India	Software-as-a-service, archival	5–19,000	Business & Casual

to other teams, projects, roles, or organizations. As a result, the context and shared understanding of user stories can be lost, particularly for those who were not part of the original discussions (Conboy, 2021). Because these conversations and shared understandings reside in the memory of individuals, they are no longer accessible to the team when those individuals depart. For new team members, a reminder of a conversation they did not participate in holds little value. Vidgen and Wang (2009)'s longitudinal case study highlights that understandings of user needs are often shared through informal practices, such as rotating pair programming. In this practice, developers working in rotating pairs demonstrate how knowledge of user needs extends beyond the written user stories, relying heavily on team interactions (Vidgen and Wang, 2009).

Overall, the findings suggest that user stories help create a shared understanding between users and developers by facilitating conversations across their boundaries. However, their effectiveness depends on the skills of those using them, particularly in writing and telling stories. User stories also seem to work best in stable teams with low turnover, where those who have built the shared understanding remain. This leads to the following proposition:

> Proposition 1: User stories facilitate shared understanding between developers and users.

4.2. Proposition 2: Small user stories help developers cope with change

When a user story encompasses too broad a scope,[5] it becomes challenging for a small team of developers to provide accurate estimates. Our selected literature shows that this is a common problem, as Van Can and Dalpiaz (2024) found that 45.5 % of backlog items in their study of fourteen projects contained two or more requirements. Moreover, our literature review also reveals that user stories with smaller scopes help developers feel more secure and in control when dealing with changes. There are several reasons for this finding.

First, small stories improve the accuracy of time estimations for developing user stories. User stories with finer granularity (i.e., smaller scope) enable developers to create more reliable short-term plans. In a survey study by Liskin et al. (2014), they observed that the clearer a developer's understanding of all necessary actions for implementing a user story, the more likely they are to identify dependencies and risks. On the contrary, user stories with a large scope pose a higher risk of unexpectedly growing into an even larger scope, often due to unantici-

[5] The term 'granularity' is often used instead of 'scope': a smaller scope indicates finer granularity.

Table 4
Support for propositions based on primary studies.

Study	P1	P2	P3	P4	P5
McAvoy and Butler (2005)	–	–	–	X	–
Vidgen and Wang (2009)	X	X	–	–	X
Børte et al. (2012)	X	–	–	–	–
Read et al. (2012)	–	–	–	–	X
Liskin et al. (2014)	–	X	–	–	–
Lucassen et al. (2016a)	X	–	X	X	–
Diebold et al. (2018)	–	–	X	X	–
Melegati and Wang (2019)	–	–	–	X	–
Sedano et al. (2019)	X	–	–	–	–
Ananjeva et al. (2020)	X	–	–	–	–
Kasauli et al. (2021)	–	X	–	–	X
Conboy (2021)	X	–	X	X	X
Dellsén et al. (2022)	X	X	–	–	–
Van Can and Dalpiaz (2024)	–	X	X	–	–

pated dependencies between stories. The study found that when developers estimated the implementation time for small user stories, these estimates were more often precise, allowing developers to correctly judge the required duration for implementation (Liskin et al., 2014). Similarly, Dellsén et al. (2022) found that splitting user stories into smaller ones helps clarify their actual scope and dependencies to other stories. They explain that this process requires software teams to carefully consider what should be included in a user story's scope, and thereby give them deeper understanding of the task at hand.

Second, larger user stories often create delayed feedback loops, posing challenges as developers must wait longer for validation from users. This introduces uncertainty about whether they are developing the correct feature. The survey by Liskin et al. (2014) found that large stories lead to delays in obtaining feedback, as one interviewee noted: "it takes longer until one can gather feedback about this story." This delay results in rework, as developers may produce more code and features than necessary (Liskin et al., 2014). The study emphasizes that the ability to estimate whether a story is large or small is critical, concluding that "short and effective feedback cycles with the customer are the basis of any serious agile software development." Without accurate estimates of implementation time, developers may misjudge the scope of user stories. This extends feedback cycles, making interactions that cross boundaries between users and developers less frequent and increasing the likelihood of change requests later (Liskin et al., 2014).

Third, smaller-scoped stories were found to promote agility, while a larger scope acted as a constraint. User stories with a small scope support a bottom-up planning approach, enabling development teams to focus on short-term, iterative planning rather than long-term, upfront planning (Vidgen and Wang, 2009). For example, in the study by Vidgen and Wang (2009), the team of developers engaged in weekly replanning (aligned with one-week iteration lengths), selecting new user stories from the backlog each week. This practice allowed the team to incorporate newly acquired knowledge into subsequent planning cycles, ensuring updates in understanding were reflected in the next iteration's plan. By breaking large stories into smaller ones, the team was able to plan their coming week in detail, thereby improving their sense of control and stability.

These findings suggest that small user story scopes support what Vidgen and Wang (2009) refer to as "artful planning," a concept introduced by Baskerville (2006): "planning for creativity and innovation, planning for serendipity, and planning not-to-plan." Artful planning involves navigating the paradox of planning and not planning, a practice required in environments characterized by high levels of uncertainty and ambiguity. Agile planning, underpinned by small user stories, allowed the team in Vidgen and Wang (2009)'s study to operate with stability by planning the coming week in detail, while still upholding the long term uncertainty of discovering new user needs, providing developers with a sense of security and control over their work. This short fixed-length iterations, in which user stories did not change, offered the team a steady pace, a clear focus, freedom from distractions, and short-term certainty to focus on work without anxiety. Small user stories can thus provide a relief against anxiety and a guard against overwork (Vidgen and Wang, 2009).

However, in large-scale agile development, where many teams work on the same system, some of the advantages of small user stories start to diminish. Each team maintains its own backlog, and the combined number of user stories across teams can quickly reach very high levels, making it necessary to synthesize them (e.g. into epics, features or themes) to keep an overall understanding of the system's functionality (Kasauli et al., 2021). As one respondent in Kasauli et al. (2021)'s survey study, working in an organization with 30 scrum teams, explained:

> "We must base the understanding on […] let's say […] 2000 user stories […] and try to find a good way of describing the complete system."

This number of stories can be assumed to fluctuate over time as some are completed, others are abandoned, new ones are added, and some are deleted. Keeping an overview under these conditions becomes a major challenge (Kasauli et al., 2021). Even if each team develops only a handful of user stories per sprint, 30 teams together may still be working on around 100 user stories simultaneously. Dependencies between stories across teams make this even more difficult: if one team prioritizes differently than another, conflicts arise where a team cannot implement its story until another team has finished the prerequisite story (Kasauli et al., 2021). Although such dependencies can be managed through coordination, prior research shows that coordination in large-scale agile development is a complex and non-trivial task (Dingsøyr et al., 2022).

This suggests that the benefit of small user stories for coping with change may only hold in the context originally envisioned for agile development—small, co-located teams (Dingsøyr et al., 2019). Still, practitioner literature meets this challenge by suggesting that user stories should be aggregated into epics to provide an overview of the system's purpose and overarching requirements (Leffingwell, 2010).

Our literature review shows that small user stories provide developers with a sense of certainty and stability during iterations (sprints), and shorter feedback loops. This helps them cope with the long term uncertainty that is inherent in agile projects (Vidgen and Wang, 2009). We suggest the following proposition:

> Proposition 2: Small user stories help developers cope with change.

4.3. Proposition 3: Clarifying the "why" in user stories reinforces the focus on user needs but adds complexity to the development process

The literature on user stories recognize the importance of understanding the 'why.' However, in a study of 14 projects by Van Can and Dalpiaz (2024) more than half of the backlog items did not include any justification for the requirements. Our review of the literature revealed three noteworthy findings related to this issue. First, including a 'why' section in user story templates can help developers focus more on user needs. Second, technical developers often struggle with applying user stories to technical requirements. Third, developers frequently modify user story templates to fit their specific context, even though they start with the original Connextra template (Section 2). In the following paragraphs, we discuss each of these findings in more detail.

Templates for user stories have been found to shift developers' mindset towards the needs of users, encouraging them to keep users in mind to a greater extent (Diebold et al., 2018). Lucassen et al. (2016a) suggest one explanation for this shift: including a 'why' component (e.g., "…so that [benefit]") in user stories has significant benefits. First, it reduces confusion among stakeholders; second, it minimizes the need for lengthy discussions; and third, it grants developers greater autonomy in

their work. However, identifying the 'why' can be challenging, as illustrated by one respondent in their study (Lucassen et al., 2016a):

> "Typically, the why question is correctly answered if after the initial answer, you ask 'why?' again for three more times."

This emphasis on the 'why' aligns with foundational principles in RE, which highlight the importance of understanding underlying motivations to foster shared understanding and guide appropriate software development (Lucassen et al., 2016a). Nevertheless, Diebold et al. (2018)'s study found that consistently incorporating the 'why' component is not always straightforward for developers. They found that the reasoning part ("…so that [benefit]") was often omitted because the same 'why' frequently applied to multiple user stories. Developers felt that repeatedly documenting an identical rationale for several stories added unnecessary effort.

Forcing a user perspective onto developers can present significant challenges. Developers with a more technical focus often feel ambivalent about user stories, as they perceive a user-centric perspective does not align well with the technical aspects of their work. Diebold et al. (2018)'s case study found that applying user stories to technical requirements was particularly problematic, as these requirements often lack a direct connection to a user perspective. This disconnect posed a major issue, as the majority of the product requirements were technical. As a result, adhering to user stories became more of a burden than a benefit for both those creating and those utilizing the stories (Diebold et al., 2018). Further, follow-up interviews from Lucassen et al. (2016a)'s survey study revealed mixed reactions among technical respondents regarding the impact of user stories on productivity and quality. Many reported that user stories did not necessarily make software development faster (Lucassen et al., 2016a). Similarly, Conboy (2021)'s study highlighted that approximately half of the respondents noted their organizations used user stories primarily "to show the customer we are thinking of them," rather than as a genuinely effective development tool.

Developers frequently adapt user story templates to align with their specific context. At the same time, 59 % reported still using the original Connextra template ("As a ⟨role⟩, we want ⟨goal⟩, so that ⟨benefit⟩"), and many developers indicated they "never encountered the need to change to another template" (Lucassen et al., 2016a). This aligns with findings from Conboy (2021)'s study, which revealed that while developers typically tailor software engineering methods to suit their context, user stories often remain an exception. Most developers reported continuing using the original user story template (Conboy, 2021). For instance, 13 out of 16 practitioners in Conboy (2021)'s study stated that the original Connextra template was the most commonly used format in their organizations, despite other parts of the software development process being frequently modified or tailored. Interestingly, Lucassen et al. (2016a) noted that the specific choice of template is less important than the act of using a template at all, as it provides structure and consistency to the process. The study indicated that having a single, agreed-upon template helps ensure consistency across the team, enhancing team alignment and contributing positively to both productivity and quality. One respondent summarized this impact:

> "It's not the template that improves quality, it's what we're doing—we're sharing requirements and a template makes that easy to do and more likely that we'll do it."

Drawing together the findings from these studies indicates that user story templates play a significant role in promoting a user focused perspective among developers. While the inclusion of a 'why' component can shift focus toward user needs, challenges arise in aligning technical requirements with user stories. Despite these challenges, many developers continue to rely on the original Connextra template. We suggest the following propostion:

> Proposition 3: Clarifying the "why" in user stories reinforces the focus on user needs but adds complexity to the development process.

4.4. Proposition 4: Conversations triggered by user stories can hamper the sense of productivity

While user stories have the potential to boost productivity by ensuring that the *right* software is developed (Lucassen et al., 2016a), our literature review reveals that they can also create an impression of reduced productivity.

One explanation for this perceived decrease in productivity is that user stories tend to stimulate numerous discussions. Developers have reported that ongoing discussions about user stories are time-consuming, and breaking stories into smaller, clearer parts often results in even more discussions (Lucassen et al., 2016a). In their survey study, Lucassen et al. (2016a) found that developers felt time spent discussing was time taken away from actual development work. Interestingly, this presents a paradox, as facilitating conversations is one of the key arguments in favor of employing user stories (Cohn, 2005; Leffingwell, 2010)—user stories are meant to generate discussions. This can be exemplified by findings from McAvoy and Butler (2005)'s longitudinal case study: they describe a project led by a project manager with a "get on with it" attitude, where user stories were perceived as cumbersome, even though individual developers found them valuable. This mindset led the team to view backlog prioritization and customer involvement as obstacles to progress because these activities required additional time. The project manager was particularly popular among developers because of his effort to minimize anything that slowed down development. As a result, activities like prioritizing user stories and exploring user needs were seen as hindrances to developer tasks. Influenced by the manager's popularity and a desire among developers to avoid conflict and maintain cohesion, the use of user stories eventually diminished.

Another explanation is that developers with a more technical perspective often feel ambivalent about the impact of user stories as they do not necessarily make development faster. As noted in follow-up interviews of Lucassen et al. (2016a)'s survey study, technical respondents expressed mixed feelings regarding the effect of user stories on their work productivity and quality. They did not necessarily find software development significantly faster. Diebold et al. (2018)'s case study showed similar findings, user stories were particularly challenging to apply for technical requirements, which often lack a clear user perspective. This made adhering to user stories more burdensome than beneficial for both those producing and consuming the stories.

Additionally, developers have a preference for coding and producing tangible results over the process of discovering user needs. Melegati and Wang (2019) argued that user stories are product assumptions that can be framed as hypotheses of users' needs and tested in experiments. However, developers expressed a preference for coding and creating tangible outcomes rather than following hypothesis-driven methods that do not directly involve coding. As one of Melegati and Wang (2019)'s interviewees said: "when you are a developer, you are just focused on actually developing." The developers emphasized the importance of feedback on what they were actively building (features and code), instead of feedback on constructed hypotheses. They even showed resistance to perform tasks like interviewing and surveying users, one developer considered such tasks as "time-consuming and difficult to manage," while another "had some doubts about who was going to perform these tasks," and he did not expect it, as a developer, to be him (Melegati and Wang, 2019).

In some cases, user stories were used mainly "to show the customer we are thinking of them" (Conboy, 2021). While developers noted that user stories were essential for customer relations, they were rarely integrated into core development activities, such as requirements specification, prioritization, testing, or retrospectives. As one developer put it (Conboy, 2021):

> "Stories are so exciting, rich, loaded with potential, and incredibly powerful communication tools, but people either just like them or they don't. So many people ignore them or don't really engage."

Paradoxically, while user stories contribute to creating the *right* software, they can create a sense of unproductiveness among developers. The time-consuming discussions associated with user stories often detract from the hands-on coding work that developers prefer, particularly for technical requirements. While the concept of encouraging discussions may seem beneficial in theory, in practice, it can become a case of negative discussions. As a result, user stories are sometimes perceived as burdensome, leading to their limited integration into core development processes despite their acknowledged value in customer communication and feedback. Thus, we propose the following proposition:

> Proposition 4: Conversations triggered by user stories can hamper the sense of productivity.

4.5. Proposition 5: User stories as recorded in writing degrade over time

User stories often degrade over time for three primary reasons. First, user stories typically go through an initial "glory period" marked by rich descriptions, discussions, and debates at the time of their creation (Conboy, 2021). However, as one interviewee in Conboy (2021)'s survey study noted, "the level of debate and discussion takes a dramatic drop soon after that creation." If a user story fails to spark ideas during this early stage, it is unlikely to do so later in the development process (Conboy, 2021).

Second, new insight and understanding about user needs are often not written down. As developers implement user stories, they engage in conversations that help create shared understanding and uncover additional stakeholder needs, as described in Proposition 1. However, these insights are often not captured as part of the user stories, leading to a disconnect between the written user story and the evolving user story held in the mind of developers. Consequently, the developers' mental model of the user story becomes the most accurate representation of stakeholder needs and context, making the written version effectively outdated. Vidgen and Wang (2009) observed how developers deliberately worked in rotating pairs to share such insights about user needs. This issue becomes particularly challenging in large projects, which often combine plan-driven approaches with agile practices (Kasauli et al., 2021). In such setups, requirements are allocated to teams, who then decompose them into user stories. Being able to trace implemented code to backlog items can be essential in these contexts but is hard to maintain if developers do not update the user stories. In their survey study, Kasauli et al. (2021) found that developers rarely perceive traceability as valuable, leading them to skip updating user stories. The absence of adequate traceability was found to hinder collaboration across teams, slowing down progress in large-scale projects. Developers acknowledged this issue and expressed a desire for incentives to update backlog items and thereby enhancing traceability (Kasauli et al., 2021).

Third, user stories tend to become obsolete or outdated over time. In Read et al. (2012)'s case study, it was observed that as each iteration or sprint introduced changes to the software, the product owner would reflect on these changes and spend time discussing them with designers and developers. This evolving understanding of user needs often made older user stories in the backlog obsolete (Read et al., 2012). Moreover, changes in context or the personas described in user stories can make them less relevant to the project's current state, leading to natural degradation as written user stories gradually fail to capture the evolving context, personas, or needs (Conboy, 2021). Building on the three findings outlined above, we suggest the following proposition:

> Proposition 5: User stories as recorded in writing degrade over time.

It is worth asking whether the degradation of written user stories actually poses a problem for developers. If user stories continue to fulfill their primary purpose—facilitating a shared understanding of user needs among developers—then this decline in written quality could be viewed as a rational and acceptable development. However, we also recognize that in contexts where traceability is important, such as in large-scale software projects (Kasauli et al., 2021), poorly documented or deteriorating user stories can present significant challenges.

5. Discussion

It has been more than twenty years since the first academic papers on user stories were published, and their widespread adoption in practice has been remarkable. This makes it timely to examine how empirical research has contributed to understanding their role and effectiveness. In the following, we return to the research questions introduced at the beginning of this article to discuss how empirical studies have deepened our understanding of how user stories are used in practice. A summary of the propostitions and their implications is presented in Table 5.

5.1. Empirical contributions from three research communities

Returning to RQ1: Which research communities have generated empirical findings on the industry use of user stories? Our selected studies mainly originate from three distinct research communities. All propositions were supported by studies from each of these three communities, with the exception of Proposition 5, which was supported by two of them.

The RE community: Out of the four published studies by the RE community, two take a broader perspective on user stories: how practitioners use them and whether they find them effective (Lucassen et al., 2016a), and what challenges arise when using user stories in large-scale development (Kasauli et al., 2021). The other two focus more narrowly: on the impact of splitting user stories (Dellsén et al., 2022), and on how user stories are labeled in backlogs (Van Can and Dalpiaz, 2024). Conboy (2021)'s critique of RE publications on user stories is particularly relevant, as it underscores the tendency to emphasize operational aspects such as testability, clarity, and independence. Conboy notes that "prior studies have concentrated solely on factors within the stories themselves, overlooking the development context factors that play a significant role." However, Kasauli et al. (2021) provide a detailed analysis of how the context of large-scale agile development affects the use of user stories. For example, they show that large scale development makes it harder for teams to demonstrate results to users and thus to obtain feedback; the sheer number of user stories makes it difficult to maintain an overview of the product; and prioritization becomes complicated because of dependencies between stories. RE researchers hold the opportunity to conduct more studies on contextual factors that shape the use of user stories in other settings, following the example of Kasauli et al. (2021).

The agile community: Surprisingly, with more than 8,500[6] published papers on agile software development, only five are empirical studies on user stories. Two of these studies (Liskin et al., 2014; Børte et al., 2012) investigated how practitioners estimate user stories, the knowledge required for estimation, and whether story scope can be managed by splitting them. One study investigated how to decide which agile practices should be implemented in different contexts, and how they should be implemented, with user stories being an essential practice (Diebold et al., 2018). Another study explored the nature and role of

[6] Number of papers in Web of Science with 'agile' in either the title, abstract or keywords within the categories of Software Engineering or Information Systems, as of 11th of April 2025

Table 5
Summary of proposition and implications for practice and research.

Propositions	Description	Implications
P1: User stories facilitate shared understanding between developers and users	Shared understanding emerges from conversations and is retained in the collective memory of the team. When team members are replaced, newcomers lack this conversational history, making it difficult to sustain shared understanding. For those who did not participate in earlier discussions, reminders of them have limited value.	When faced with turnover in teams, organizations should explicitly value and assess storytelling capabilities when replacing team members in roles that involve creating user stories (such as Product Owners, Business Analysts, or developers in some contexts) (Conboy, 2021). Researchers could study what storytelling skills are relevant in a software engineering context.
P2: Small user stories help developers cope with change	Small user stories may offer psychological benefits for developers navigating agile projects that constantly change. Smaller stories can create a "short-term certainty" within the broader context of project uncertainty. This allows developers to work with greater confidence within defined parameters (like a sprint), potentially providing a sense of stability when the larger landscape appears in flux.	Because agile development naturally involves frequent changes, smaller stories give developers more stability and a clearer sense of control within each sprint. This creates a counterbalance to the constant change and shifting goals that can cause anxiety and stress. With smaller stories, developers know that while change will happen, it will not disrupt the current sprint (Vidgen and Wang, 2009). Studies on developer experience (Fagerholm and Münch, 2012) in teams that use user stories could be a fruitful direction for future research.
P3: Clarifying the 'Why' in user stories reinforces the focus on user needs but adds complexity to the development process	Finding the true purpose behind a requirement often involves asking 'why?' multiple times to reach real understanding—a process that takes considerable time away from coding. Current literature highlights the benefits of this practice while overlooking the practical costs. This might explain why some developers see user stories primarily as a tool for customer relations rather than something central to their development work.	Organizations should recognize the extra work developers face when creating and exploring the 'why' component of user stories. It also requires time from users, who must answer these why-questions and step away from their day-to-day tasks. This user effort is a real contribution to the development process, yet it is often taken for granted and left unrecognized by companies.
P4: Conversations triggered by user stories can hamper the sense of productivity	Organizations often do not recognize that *technical requirements* fundamentally differ from *user requirements*. User stories are designed to bridge boundaries between developers and users, but technical stories primarily serve developers without crossing these boundaries. When technical requirements are forced into user story templates, it wastes time and undermines the true purpose of user stories as boundary-crossing objects.	By acknowledging this distinction, development teams can use more appropriate formats for technical requirements (like tech-stories) while reserving user stories for situations where they genuinely facilitate communication between technical and non-technical stakeholders.
P5: User stories as recorded in writing degrade over time	User stories become obsolete after they have served their purpose. This raises an important question: does this obsolescence represent a problem? Developers might view updating user stories as a waste of resources, especially when the story has already been implemented.	This has implications for ongoing research in RE that depends on traceability and on user stories being kept up to date. For example, recent work explores using LLMs to extract requirements from user stories (Wautelet et al., 2018; Kochbati et al., 2021; Robeer et al., 2016; Raharjana et al., 2021), improve their quality (Zhang et al., 2024), address ambiguity (Amna et al., 2025), and support their reuse in future projects (Resketi et al., 2020).

product backlogs and how they emerge (Sedano et al., 2019). A fifth paper examined how hypotheses in experiment-driven software development can be incorporated into user stories (Melegati and Wang, 2019). Lastly, a sixt paper, investigated how UX work was integrated with agile development through user stories (Ananjeva et al., 2020).

The agile community has long taken pride in its proximity and relevance to practitioners and its commitment to addressing developers' practical problems (Dybå and Dingsøyr, 2008). Yet, when it comes to user stories, this promise has not been fully realized. The community has a clear opportunity to strengthen this area by conducting field studies that generate robust empirical insights and provide practitioners with evidence-based guidance for working with user stories.

Information Systems Development (ISD): The publications from the ISD community differ from those in the agile and RE communities by placing greater emphasis on theoretical frameworks to interpret empirical findings. For example, one study applies Competing Commitment Theory to explain why developers' initial commitment to user stories shifted over time into skepticism and eventually abandonment (McAvoy and Butler, 2005). Two studies use Complex Adaptive Systems theory: one to examine how requirements and user stories change (Vidgen and Wang, 2009), and another to analyze how requirements are communicated (Read et al., 2012). The most recent IS publication, Conboy (2021), employs the myth of the Greek god Prometheus as "a metaphorical symbol of technology's radically innovative, transformative power" to explore the narrative potential of user stories. The ISD community shows maturity in using theoretical perspectives to conduct in-depth analyses of user stories, offering rich insights and new ways of understanding them. The agile and RE communities could gain substantially by adopting such a theory-oriented approach to research, particularly in the integration of theory into the study of software development practices such as user stories.

5.2. Industry contexts of user story studies

Returning to RQ2: In what industry contexts have user stories been studied empirically?

Table 3 shows that user stories have mainly been studied in the context of business software. Acute software and casual software appear only once and twice, respectively. This imbalance suggests that the propositions developed here may not apply to other types of systems, such as safety-critical or low-stakes domains, which limits how far current propositions can be generalized beyond business-oriented contexts. In addition, several important industries in which software plays a vital role are missing from the research, such as energy and utilities (for example, smart grids, oil and gas), retail and e-commerce, agriculture, the maritime industry, and entertainment. Each industry has its own specific context, and Proposition 3 has shown that user story templates are often adapted to fit the context. Conducting studies within these different industries could therefore provide developers with more relevant and practical insights into how user stories can be used effectively in their own domains, while also strengthening the empirical foundation for our propositions.

Most studies have been conducted in Europe and the United States, with only a few from South and Southeast Asia. Large parts of Asia, South America, and Africa are still underrepresented. As a result, the propositions are grounded in empirical studies from only a small portion of the world's cultures. Given that software development is often outsourced to lower-cost countries in Asia and Africa, understanding how user stories are used in these regions deserves more attention. It is also notable that South America, and particularly Brazil—with its strong software engineering research community—has little representation. Comparing how user stories are applied across different cultural, industrial,

and regional contexts could provide practitioners with valuable insights and help build more generalizable propositions and theories.

In the reviewed papers, user stories are examined in contexts where agile methods were not originally intended to be applied (Dingsøyr et al., 2019). Examples include large-scale development projects (Kasauli et al., 2021), geographically distributed teams (Ananjeva et al., 2020), and cases where the customer is not on-site (Sedano et al., 2019). Some studies also focus on organizations new to agile practices. In these cases, the results can be assumed to be influenced by inexperience and common beginner mistakes—for example, using user stories for technical tasks instead of applying *technical stories* as recommended by Kniberg (2015), or neglecting backlog refinement (Conboy, 2021). Such issues risk leading to criticism of user stories as an artifact, when in fact the challenges may stem from their misuse or from unsuitable settings. We argue that researchers should have prioritized studying cases that align well with agile principles—such as co-located teams and smaller project scopes—when assessing the claimed benefits of user stories.[7] Investigating how they perform in less favorable conditions is also important, but better suited for later stages of research. Nevertheless, the selected studies are methodologically sound and offer valuable insights that can guide future work.

5.3. User stories as boundary objects

Returning to RQ3: How do user stories facilitate knowledge integration between stakeholders in software development?

We have argued that user stories can be understood as boundary objects, and consequently the findings of this literature review align with boundary object theory. For example, boundary objects are subject to change, rather than being stable and static (Carlile, 2002; Doolin and McLeod, 2012), which mirrors how user stories change and degrade over time (Proposition 5). Boundary objects enable knowledge transfer and the translation of meaning and interests (Carlile, 2002; Star and Griesemer, 1989; Doolin and McLeod, 2012), which corresponds to how user stories help create shared understanding (Proposition 1) and how their 'why'-component helps convey user needs (Proposition 3). Further, boundary objects facilitate cooperation between intersecting communities by maintaining a common identity while satisfying the informational needs of each (Star and Griesemer, 1989; Star, 2010). This matches how user story templates provide common structure while allowing stakeholders to focus on their relevant aspects (Proposition 3).

By examining user stories through the lens of boundary objects theory, we can draw on boundary object literature to enhance our understanding of user stories. In the remainder of this discussion, we apply boundary object theory to explain how user stories effectively bridge knowledge boundaries between users (domain experts) and software developers.

5.4. Boundary objects-in-use

Although user stories are designed to act as boundary objects, they might not actually be *used* as such. By recognizing user stories as boundary objects, we can distinguish between those actively used to bridge boundaries between developers and users, and those which are only used separately by users and developers in their local fields. Levina and Vaast (2005) differentiate between *designated boundary objects*, which are objects intended to span boundaries but end up not being used as such, and *boundary objects-in-use*, which are actively used to bridge boundaries. Some user stories end up in the category of designated boundary objects because they are only used in one local field, not to span the boundaries between fields, as illustrated in Fig. 5. An example is tech-stories (see Proposition 4 in Section 4.4), which are written in

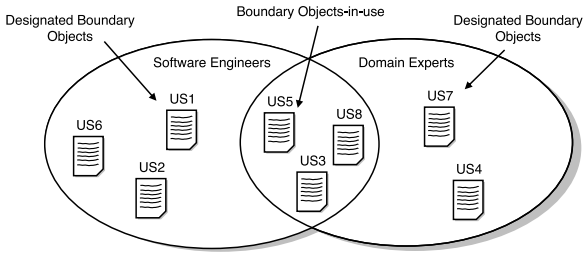


Fig. 5. Illustrates how some user stories act as boundary objects-in-use by facilitating crossing boundaries between developers and users (US3, US5, US8), while others remain designated boundary objects, only relevant within their local fields.

the user story format but are only useful for developers and not understandable to users. As a result, they do not facilitate boundary-spanning, as illustrated by US1, US2, and US6 in Fig. 5. Another example is user stories created solely to satisfy users but not utilized by developers (see Proposition 4 in Section 4.4). These stories serve as discussion points among users rather than bridging communication between users and developers, as shown by US4 and US7 in Fig. 5. In contrast, user stories that actively promote discussions, dialogue, and investigations unite developers and users, becoming boundary objects-in-use. This is illustrated by US3, US5, and US8 in Fig. 5. A true boundary object-in-use is characterized by participants reaching a shared understanding of its meaning, demonstrating its value through their actions, and using it to establish a common identity. Furthermore, it facilitates genuine negotiations among participants (Carlile, 2002; Levina and Vaast, 2005).

Boundary objects-in-use may begin as designated boundary objects that are adopted later, or they may emerge organically through interactions. The same goes for user stories. When user stories are small in size (Proposition 2), they become more concrete and accessible to all participants across boundaries. Similarly, when they clearly explain the 'why' component (Proposition 3), they help create shared understanding between domain experts and developers, bringing them closer to a common vision and enabling the story to serve as a boundary object-in-use rather than remaining a designated but unused artifact. Furthermore, even seemingly suitable boundary objects can end up as designated boundary objects. To establish effective boundary objects-in-use, organizations need people who understand which objects function well in local fields and can adapt them to create shared meaning in new contexts. These individuals, often called boundary spanners (Levina and Vaast, 2005), customize, demonstrate, and teach different groups how to use boundary objects both locally and across boundaries. They facilitate productive use of boundary objects and introduce them at appropriate times. This connects to Proposition 1, which emphasizes that storytelling skills are essential for creating shared understandings of user stories.

An important observation emerges when combining the distinction between designated boundary objects and boundary objects-in-use (Levina and Vaast, 2005) with the idea that an object's boundary-spanning effectiveness changes over time (Carlile, 2002, 2004; Star and Griesemer, 1989; Doolin and McLeod, 2012). This suggests that boundary objects are only in-use for specific periods. Proposition 5 supports this notion by highlighting that user stories eventually fall out of use (Conboy, 2021). Consequently, a user story that appears to be a designated boundary object at one point in time may, at another point, function as a boundary object-in-use. This highlights the importance of considering time and temporality when applying user stories.

5.5. Crossing knowledge boundaries

Suppose a user story has become a boundary object-in-use, it now supports crossing the knowledge boundary between fields. However, as described in Section 2, there are three types of boundaries (Carlile, 2002) to be crossed. In the following subsections, we explore how user

[7] We acknowledge that our search strategy of using Web of Science may have excluded studies that were set in these 'ideal' contexts.

stories can act as boundary objects-in-use to cross the syntactic, semantic, and pragmatic knowledge boundaries.

5.5.1. The syntactic knowledge boundary

The syntactic knowledge boundary arises when different groups use varying vocabularies or terminologies to describe their work. Syntactic boundaries become problematic when faced with novelty and change because new concepts and words emerge that are not part of the established vocabulary (Carlile, 2002). To overcome this boundary, a shared language or terminology must be established between the groups, creating a common syntax. The user story template, as described in Proposition 3, enforces a shared language that is understandable by non-technical stakeholders as well as developers. This ensures that user stories are accessible to all groups by introducing a shared and standardized vocabulary and terminology, thus bridging the syntactic boundary. However, none of the selected studies examine the actual words that are placed into the template–specifically, the terms used for the ⟨role⟩, ⟨goal⟩, and ⟨benefit⟩. It seems reasonable to assume that during backlog refinement, developers can ask clarifying questions like "what do you mean by that word?", which helps cross the syntactic boundary. In contrast, traditional requirements documents often include a glossary to define terminology in advance. This practice reduces the need for clarification during development because much of the dialogue has already taken place during the elicitation phase. However, when the requirements document is produced by someone other than the responsible developer, it can hinder dialogue and reinforce the boundary between those who define the requirements and those who implement them.

In contemporary times, software engineering for AI software is an example of such novelty where a new type of requirement is surfacing: explainability requirements (Chazette and Schneider, 2020; Köhl et al., 2019). When developing AI systems, users need explanations on how the system works and why they can trust them (Lebovitz et al., 2022; Miller, 2023; Balasubramaniam et al., 2023). Various practices and solutions for eliciting explainability requirements have already been developed (Balasubramaniam et al., 2024), including a user story template tailored for this type of requirement (Balasubramaniam et al., 2023):

> "As a ⟨type of addressee⟩, we want to get explanation(s) on an ⟨aspect⟩ of a ⟨system⟩ from an ⟨explainer⟩ in a ⟨context⟩."

This example illustrates how user story templates may evolve over time to adapt to new syntactic challenges and highlights untapped research opportunities on user stories. Different industries face unique syntactic challenges that have yet to be explored and described.

5.5.2. The semantic knowledge boundary

A semantic boundary emerges when distinct groups attribute different interpretations to identical terminology despite sharing a common language. To bridge semantic boundaries, groups must comprehend alternative perspectives, recognize the foundations of interpretive differences, and collaborate to establish mutual understanding. User stories help cross the semantic knowledge boundary in several ways. As demonstrated in Proposition 3, incorporating the 'why'-component support crossing this boundary by elucidating a user story's purpose, thereby reducing stakeholder confusion and minimizing lengthy discussions. Furthermore, as evidenced in Proposition 1, user stories support dialogues that foster shared understanding among participants.

Conversely, Proposition 5 indicates that user stories as recorded in writing degrade over time as project contexts evolve or new insights emerge. Since user stories are rarely updated, this can lead to a crevice between the written words and participants' mental models, potentially undermining the shared semantic understanding of the written words. This observation contradicts established research suggesting that boundary objects must be up-to-date (Carlile, 1997), yet outdated user stories continue to serve practical functions. This apparent contradiction thus prompts an important inquiry: as user stories degrade over time, do their usefulness as boundary objects diminish?

5.5.3. The pragmatic knowledge boundary

A pragmatic boundary exists when different groups have different interests and consequences at stake in their work. Despite having established a shared terminology and a shared semantic understanding of a user story, which has crossed the syntactic and semantic boundary, the user story can still be rejected if it threatens the interests of one or more of the involved groups. Each involved community has its own political agenda and sees its knowledge "at stake." This is the most complex knowledge boundary to bridge. Surrounding this boundary, we find groups with conflicting interests, where changes proposed by one group might create negative consequences for others, and people who are reluctant to change because their knowledge represents successful solutions that are costly to alter.

To cross pragmatic boundaries, groups need to negotiate changes that work for all parties, create new shared knowledge, and be willing to alter their existing practices, as suggested by Carlile (2002):

> "At a pragmatic boundary, an effective boundary object facilitates a process where individuals can jointly transform their knowledge. If there are negative consequences identified, then the individuals involved must be able to alter, negotiate, or change the object used."

As shown in Proposition 2, small user stories enable frequent feedback loops and negotiations between stakeholders. Small user stories create the opportunity to gain feedback from stakeholders quickly, frequently opening up the discussion of whether stories should be changed, removed, or added. Furthermore, the backlog represents the bigger picture in which stories are to be prioritized for development iterations (sprints), which is frequently reprioritized. In this way, user stories become a boundary object that supports crossing the pragmatic boundary.

To support crossing the pragmatic boundary, user stories need to hold *interpretive flexibility*. Boundary objects must be adaptable to different interpretations by various stakeholders while simultaneously allowing them to serve as a common reference point despite differing perspectives (Doolin and McLeod, 2012; Star, 2010; Abraham, 2013). This characteristic enables groups to engage with the object from their unique viewpoints, attributing distinct meanings while maintaining a shared reference frame. By being interpretively flexible, user stories can hold within them the interests of several groups at the same time and enable the groups to investigate each other's interests and conduct negotiations.

5.6. Generating new research questions with boundary objects as a lens

By viewing software development as a knowledge integration challenge, where domain experts bring business needs and user requirements, and software developers contribute technical expertise (Tiwana, 2004), it becomes key to study how knowledge is transferred between these groups. At the same time, the field of software engineering is undergoing a major shift (Stray et al., 2025). Practitioners and researchers are already exploring how software requirements can be used as prompts for large language models (LLMs) to automatically generate code (Robinson et al., 2025). This repositions part of the knowledge transfer process from human-to-human communication to human-to-machine, where requirements are translated directly into code. In this context, user stories are evolving into boundary objects, not just to bridge understanding between people, but also between humans and machines. This raises a timely question: are user stories well suited to carry knowledge about requirements into an LLM and Generative AI context?

When requirements are used as prompts for LLMs, the structure and phrasing of user stories become increasingly important, as they will most

Table 6
Future research questions in the age of Generative AI.

Boundary	Description	Future Research Questions
Syntactic	Focuses on the structure and format of user stories and how these affect LLM interpretation.	• How do cross-functional teams negotiate the syntax of user stories when writing them for dual use—as communication artifacts and LLM prompts? • How does the level of experience among team members affect the syntactic clarity of user stories used as LLM prompts? • How sensitive are different LLMs to ambiguity or redundancy in the syntactic construction of user stories? • To what extent can syntactic patterns be standardized to improve prompt consistency and code quality?
Semantic	Concerns the meaning embedded in user stories and how it is interpreted by both humans and machines.	• What types of semantic misunderstandings most commonly occur when LLMs interpret user stories? • How can teams detect and correct semantic inconsistencies in prompts that lead to faulty or unintended code? • How do LLM interpretations of user stories differ from those of human developers, and what are the implications? • What strategies can be used to align LLM interpretations with the intended meaning as understood by the team? • What mechanisms can be introduced to ensure that LLMs preserve the intended meaning of prompts in the generated code?
Pragmatic	Addresses how stakeholders negotiate intent, context, and purpose in user stories, particularly when used for LLM prompting.	• How do stakeholders negotiate what to include in user stories when preparing prompts for LLMs? • Who holds decision-making power over how user stories are phrased when the goal is LLM-based code generation? • How do power asymmetries between technical and non-technical stakeholders influence the final form of prompts given to LLMs? • How is collective intent negotiated and made explicit in user stories written for LLM interpretation? • What tools or practices support effective negotiation of user story content before it is submitted as a prompt to an LLM? • How do negotiation practices change over time as stakeholders gain experience with LLM-generated code?

likely shape how the model interprets the input and generates code. This shift introduces new syntactic considerations, where writing practices must account for both human understanding and machine processing. However, even when the syntax is well-formed, different interpretations of meaning can arise—what developers and stakeholders intend semantically may not align with what the LLM infers from the same text. This highlights the importance of examining how meaning is conveyed and preserved across human–LLM interactions. In addition, the process of negotiating requirements changes when the target audience is not just a developer but also an LLM. Traditionally, developers have interpreted and refined vague or incomplete requirements during implementation. With LLMs, that interpretive step is automated, making it essential to understand how stakeholders negotiate what to include, omit, or clarify in user stories that serve as prompts. Together, these considerations raise new research questions at the syntactic, semantic, and pragmatic knowledge boundaries (see Table 6).

5.7. Limitations

The findings presented in this paper should be seen in light of a number of limitations. First, the lack of a formal, documented quality assurance process for selecting papers that is commonly found in SLRs, makes it difficult for other researchers to assess the rigor behind our findings (Paré et al., 2015). We limited our search to Web of Science, which includes all leading SE and IS outlets, and the AIS Digital Library, relying on their quality control processes for curating publication channels (Clarivate, 2025). Additionally, we included a complete list of selected and rejected papers, along with a description of each exclusion step (see Fig. 2), to make our selection process as transparent as possible for interested readers.

Second, the limited number of studies raises concerns about the external validity of our findings. The five propositions presented would benefit from empirical validation through additional case studies and surveys before broader generalization. Since user stories are widely used in software development, researchers should have access to relevant cases, providing ample opportunity for empirical validation. However, our approach of theorizing and applying boundary objects as a theoretical lens, with propositions tested through detailed analysis, helps address some generalization limitations. In this way, we attempt to strengthen external validity by examining our findings in relation to established theory.

Third, although practitioner literature provides definitions of user stories (Cohn, 2004b; Leffingwell, 2010), their operationalization varies widely across companies. This variation creates a construct validity issue (Sjøberg and Bergersen, 2022): two companies may claim to use user stories but apply them so differently that they should be treated as distinct constructs. Unfortunately, only a few of the selected studies provide detailed descriptions of local practices, making it difficult to assess construct validity in the primary studies.

Fourth, researcher expectations can unintentionally lead to an overemphasis on studies that support boundary objects theory as a fitting explanation. To reduce this confirmation bias, we formulated our propositions before exploring theories to explain them. Since eliminating bias entirely is next to impossible, we held critical discussions to ensure that studies were not selectively chosen to align with preexisting views.

Finally, it is worth noting that a theoretical review such as this relies on the skills of investigators to perform a thorough synthesis of prior findings. Unlike statistical analyses, for example, such synthesis cannot be guided by standardized processes. Instead, the process of reading, interpreting, and synthesizing study results into theoretical propositions remains a subjective process for which there are no absolute evaluation criteria.

6. Conclusion

This article presents a theoretical review of empirical studies of user stories in practice. Our review identified 14 studies, exposing a significant gap between the widespread industry adoption of user stories and the limited academic research examining them in practice. Despite user stories being employed by approximately 50 % of all software developers and 90 % of agile developers, there are remarkably few rigorous empirical studies exploring their use in industry settings.

Based on a synthesis of the findings of these studies, we developed five theoretical propositions: (1) user stories facilitate shared understanding between developers and users; (2) small user stories help developers cope with change; (3) clarifying the ‘why’ in user stories reinforces the focus on user needs but adds complexity to the development

process; (4) conversations triggered by user stories can hamper the sense of productivity; and (5) user stories as recorded in writing degrade over time.

Analyzing these propositions through the lens of boundary object theory demonstrates how user stories function as boundary objects that can help bridge syntactic, semantic, and pragmatic knowledge boundaries between stakeholders. This theoretical perspective explains why certain user stories succeed while others fail to facilitate knowledge transfer across knowledge boundaries in software development.

The initial theoretical framework comprising the five propositions developed in this paper can offer a new theoretical foundation for future work, for example, by identifying new research questions, and by extending, refining, or possibly refuting the propositions.

CRediT authorship contribution statement

Tor Sporsem: Writing – original draft, Visualization, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Torgeir Dingsøyr:** Writing – review & editing, Supervision, Project administration, Methodology, Formal analysis, Conceptualization; **Klaas-Jan Stol:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Formal analysis, Conceptualization.

Data availability

No data was used for the research described in the article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We thank our colleagues at Aalto University for their valuable discussions and comments on an earlier draft of this article. We are also grateful to Daniel Mendez and Marcos Kalinowski for their engagement with the initial ideas that helped shape this work. In addition, we appreciate the insightful critique provided by our colleagues at NTNU in the reading circle “Forskerfabrikken” (“The Research Factory”) on an earlier draft of this article. We would like to express our gratitude to the anonymous reviewers and the editor for their constructive feedback that has led to a better article.

This work was supported by Norwegian Research Council, grant number 346563, Research Ireland grant 13/RC/2094-P2, and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero–the Research Ireland Centre for Software (www.lero.ie).

Appendix A. Author survey

The following email was sent to all authors of the reviewed papers in order to solicit further contextual information.

Dear <author>, We hope this message finds you well. We’re currently conducting a review of empirical studies on “user stories,” and have included your paper titled “Requirements engineering challenges and practices in large-scale agile system development,” which we found both valuable and insightful. As part of this review, we’re collecting some additional contextual information about the included studies, and we would be very grateful if you could help by briefly answering the following questions about your work: Region: From which geographical region(s) were the companies or participants in your study? For example, a specific country. Organization Size: What was the approximate size of the organization(s) of your participants? Industry domain: Which industry domain(s) did the studied organizations or participants represent? For example, companies may be operating in specific sectors like energy, automotive, or insurance (to name only a few). Organization Size: What was the approximate size of the organization(s) of your participants? Depending on what you may be able to report, this could be specified as the number of employees, or on a categorical scale such as startup, small and medium enterprise (SME), or large enterprise. System Type: Could you describe the type of system that was the focus of the study? If possible, we would greatly appreciate your response by August 20th. However, we completely understand if some of this information is difficult to retrieve or not readily available–please share what you can. Thank you very much for your time and support–we truly appreciate it.

Best regards, Tor Sporsem, Torgeir Dingsøyr, and Klaas-Jan Stol NTNU, SINTEF & UCC

References

Abbott, R.J., Moorhead, D.K., 1981. Software requirements and specifications: a survey of needs and languages. J. Syst. Softw. 2 (4), 297–316. https://doi.org/10.1016/0164-1212(81)90004-2

Abraham, R., 2013. Enterprise architecture artifacts as boundary objects-a framework of properties. In: Proceedings of the European Conference on Information Systems 2013. https://aisel.aisnet.org/ecis2013_cr/120.

Abrahamsson, P., Babar, M.A., Kruchten, P., 2010. Agility and architecture: can they coexist? IEEE Softw. 27 (2), 16–22. https://doi.org/10.1109/MS.2010.36

Ambler, S.W., 2003. Agile model driven development is good enough. IEEE Softw. 20 (5), 71–73. https://doi.org/10.1109/MS.2003.1231156

Amna, A.R., Poels, G., 2022a. Ambiguity in user stories: a systematic literature review. Inf. Softw. Technol. 145, 106824. https://doi.org/10.1016/j.infsof.2022.106824

Amna, A.R., Poels, G., 2022b. Systematic literature mapping of user story research. IEEE Access 10, 51723–51746. https://doi.org/10.1109/ACCESS.2022.3173745

Amna, A.R., Wautelet, Y., Poelmans, S., Heng, S., Poels, G., 2025. The AmbiTRUS framework for identifying potential ambiguity in user stories. J. Syst. Softw. , 112357. https://doi.org/10.1016/j.jss.2025.112357

Ananjeva, A., Persson, J.S., Bruun, A., 2020. Integrating UX work with agile development through user stories: an action research study in a small software company. J. Syst. Softw. 170, 110785. https://doi.org/10.1016/j.jss.2020.110785

Baham, C., Hirschheim, R., 2022. Issues, challenges, and a proposed theoretical core of agile software development research. Inf. Syst. J. 32 (1), 103–129. https://doi.org/10.1111/isj.12336

Balasubramaniam, N., Kauppinen, M., Rannisto, A., Hiekkanen, K., Kujala, S., et al., 2023. Transparency and explainability of AI systems: from ethical guidelines to requirements. Inf Softw. Technol. 159, 107197. https://doi.org/10.1016/j.infsof.2023.107197

Balasubramaniam, N., Kauppinen, M., Truong, H.-L., Kujala, S., et al., 2024. Candidate solutions for defining explainability requirements of AI systems. In: Requirements Engineering: Foundation for Software Quality. Springer Nature Switzerland, Cham, pp. 129–146. https://doi.org/10.1007/978-3-031-57327-9_8

Barrett, M., Oborn, E., 2010. Boundary object use in cross-cultural software development teams. Hum. Relat. 63 (8), 1199–1221. Sage Publications Ltd. https://doi.org/10.1177/0018726709355657

Baskerville, R.L., 2006. Artful planning. Eur. J. Inf. Syst. 15 (2), 113–115. https://doi.org/10.1057/palgrave.ejis.3000622

Beck, K., 1999. Embracing change with extreme programming. Computer 32 (10), 70–77. https://doi.org/10.1109/2.796139

Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley.

Behutiye, W., Karhapää, P., Costal, D., Oivo, M., Franch, X., 2017. Non-functional requirements documentation in agile software development: challenges and solution proposal. In: International Conference on Product-focused Software Process Improvement. Springer, pp. 515–522. https://doi.org/10.1007/978-3-319-69926-4_41

Behutiye, W., Karhapää, P., López, L., Burgués, X., Martínez-Fernández, S., Vollmer, A.M., Rodríguez, P., Franch, X., Oivo, M., et al., 2020. Management of quality requirements in agile and rapid software development: a systematic mapping study. Inf. Softw. Technol. 123, 106225. https://doi.org/10.1016/j.infsof.2019.106225

Bik, N., Lucassen, G., Brinkkemper, S., 2017. A reference method for user story requirements in agile systems development. In: 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW). IEEE, pp. 292–298. https://doi.org/10.1109/REW.2017.83

Børte, K., Ludvigsen, S.R., Mørch, A.I., 2012. The role of social interaction in software effort estimation: unpacking the “magic step” between reasoning and decision-making. Inf. Softw. Technol. 54 (9), 985–996. https://doi.org/10.1016/j.infsof.2012.03.002

Bragilovski, M., Dalpiaz, F., Sturm, A., 2022. Guided derivation of conceptual models from user stories: a controlled experiment. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 131–147. https://doi.org/10.1007/978-3-030-98464-9_11

Carlile, P.R., 1997. Understanding Knowledge Transformation in Product Development: Making Knowledge Manifest Through Boundary Objects. Ph.D. thesis. University of Michigan.
Carlile, P.R., 2002. A pragmatic view of knowledge and boundaries: boundary objects in new product development. Organ. Sci. 13 (4), 442–455. https://doi.org/10.1287/orsc.13.4.442.2953
Carlile, P.R., 2004. Transferring, translating, and transforming: an integrative framework for managing knowledge across boundaries. Organ. Sci. 15 (5), 555–568. https://doi.org/10.1287/orsc.1040.0094
Carroll, N., Conboy, K., Wang, X., et al., 2023. From transformation to normalisation: an exploratory study of a large-scale agile transformation. J. Inf. Technol. 38 (3), 267–303. SAGE Publications Ltd. https://doi.org/10.1177/02683962231164428
Chazette, L., Schneider, K., 2020. Explainability as a non-functional requirement: challenges and recommendations. Requir. Eng. 25 (4), 493–514. https://doi.org/10.1007/s00766-020-00333-1
Cheng, B. H.C., Atlee, J.M., 2007. Research directions in requirements engineering. Future of Software Engineering (FOSE'07) , 285–303. https://doi.org/10.1109/FOSE.2007.17
Chubov, I., Droujkov, D., 2007. User stories and acceptance tests as negotiation tools in offshore software development. In: Agile Processes in Software Engineering and Extreme Programming: 8th International Conference, XP 2007, Como, Italy, June 18–22, 2007. Proceedings 8. Springer, pp. 167–168. https://doi.org/10.1007/978-3-540-73101-6_27
Clarivate, Editorial selection process | Clarivate. https://clarivate.com/academia-government/scientific-and-academic-research/research-discovery-and-referencing/web-of-science/web-of-science-core-collection/editorial-selection-process/.
Cohn, M., 2004a. Effective user stories. In: Extreme Programming and Agile Methods - XP/Agile Universe 2004, Proceedings. Spinrger, p. 208. https://doi.org/10.1007/978-3-540-27777-4_35
Cohn, M., 2004b. User Stories Applied: For Agile Software Development. Addison-Wesley Professional.
Cohn, M., 2005. User stories for agile software requirements. In: Twenty-Third Annual Pacific Northwest Software Quality Conference, Proceedings. Pacific agenda, pp. 125–149. .
Conboy, K., 2009. Agility from first principles: reconstructing the concept of agility in information systems development. Inf. Syst. Res. 20 (3), 329–354. https://doi.org/10.1287/isre.1090.0236
Conboy, K., 2021. Enabling promethean leaps: an examination of storytelling techniques in information systems development. Commun. Assoc. Inf. Syst. 48 (1), 35. https://doi.org/10.17705/1cais.04835
Curcio, K., Navarro, T., Malucelli, A., Reinehr, S., et al., 2018. Requirements engineering: a systematic mapping study in agile software development. J. Syst. Softw. 139, 32–50. https://doi.org/10.1016/j.jss.2018.01.036
Dalpiaz, F., van der Schalk, I., Brinkkemper, S., Aydemir, F.B., Lucassen, G., 2019. Detecting terminological ambiguity in user stories: tool and experimentation. Inf. Softw. Technol. 110, 3–16. https://doi.org/10.1016/j.infsof.2018.12.007
Dalpiaz, F., Van der Schalk, I., Lucassen, G., 2018. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP. In: Requirements Engineering: Foundation for Software Quality: 24th International Working Conference, REFSQ 2018, Utrecht, The Netherlands, March 19–22, 2018, Proceedings 24. Springer, pp. 119–135. https://doi.org/10.1007/978-3-319-77243-1_8
Dalpiaz, F., Steghöfer, J.-P., 2024. Where requirements and agility meet: No-man's-land or a land of opportunity? IEEE Softw. 41 (3), 7–10. https://doi.org/10.1109/ms.2024.3363248
De Souza Santos, R., Ralph, P., Arham, A., Stol, K.-J., 2024. Distributed Scrum: a case meta-analysis. ACM Comput. Surv. 56 (4). https://doi.org/10.1145/3626519
Dellsén, E., Westgårdh, K., Horkoff, J., 2022. Invest in splitting: user story splitting within the software industry. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 115–130. https://doi.org/10.1007/978-3-030-98464-9_10
Diebold, P., Theobald, S., Wahl, J., Rausch, Y., 2018. An agile transition starting with user stories, DoD & DoR. In: Proceedings of the 2018 International Conference on Software and System Process, pp. 147–156. https://doi.org/10.1145/3202710.3203145
Dingsøyr, T., Falessi, D., Power, K., 2019. Agile development at scale: the next frontier. IEEE Softw. 36 (2), 30–38. https://doi.org/10.1109/MS.2018.2884884
Dingsøyr, T., Bjørnson, F.O., Schrof, J., Sporsem, T., et al., 2022. A longitudinal explanatory case study of coordination in a very large development programme: the impact of transitioning from a first- to a second-generation large-scale agile development method. Empir. Softw. Eng. 28 (1), 1. https://doi.org/10.1007/s10664-022-10230-6
Doolin, B., McLeod, L., 2012. Sociomateriality and boundary objects in information systems development. Eur. J. Inf. Syst. 21 (5), 570–586. https://doi.org/10.1057/ejis.2012.20
Dubin, R., 1978. Theory Building. The Free Press. revised edition edition.
Dybå, T., Dingsøyr, T., 2008. Empirical studies of agile software development: a systematic review. Inf. Softw. Technol. 50 (9-10), 833–859. https://doi.org/10.1016/j.infsof.2008.01.006
Edison, H., Wang, X., Conboy, K., et al., 2022. Comparing methods for large-scale agile software development: a systematic literature review. IEEE Trans. Softw. Eng. 48 (8), 2709–2731. https://doi.org/10.1109/TSE.2021.3069039
El-Attar, M., Miller, J., 2010. Developing comprehensive acceptance tests from use cases and robustness diagrams. Requir. Eng. 15, 285–306. https://doi.org/10.1007/s00766-009-0088-6
Fagerholm, F., Münch, J., 2012. Developer experience: concept and definition. In: 2012 International Conference on Software and System Process (ICSSP). IEEE, pp. 73–77. https://doi.org/10.1109/ICSSP.2012.6225984
Fitzgerald, B., 2024. Information systems and software engineering: the case for convergence. arXiv:2402.04200 Technical Report.
Glinz, M., 2007. On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007). IEEE, pp. 21–26. https://doi.org/10.1109/RE.2007.45
Goguen, J.A., Linde, C., 1993. Techniques for requirements elicitation. In: [1993] Proceedings of the IEEE International Symposium on Requirements Engineering. IEEE, pp. 152–164. https://doi.org/10.1109/ISRE.1993.324822
Gomez, A., Rueda, G., Alarcón, P.P., 2010. A systematic and lightweight method to identify dependencies between user stories. In: Agile Processes in Software Engineering and Extreme Programming: 11th International Conference, XP 2010. Proceedings 11. Springer, pp. 190–195. https://doi.org/10.1007/978-3-642-13054-0_17
Greenhalgh, T.M., Dijkstra, P., 2024. How to Read a Paper: The Basics of Evidence-based Healthcare. John Wiley & Sons.
Güneş, T., Aydemir, F.B., 2020. Automated goal model extraction from user stories using NLP. In: IEEE 28th International Requirements Engineering Conference (RE). IEEE, pp. 382–387. https://doi.org/10.1109/RE48521.2020.00052
Harbers, M., Detweiler, C., Neerincx, M.A., 2015. Embedding stakeholder values in the requirements engineering process. In: Requirements Engineering: Foundation for Software Quality: 21st International Working Conference, REFSQ 2015. Proceedings 21. Springer, pp. 318–332. https://doi.org/10.1007/978-3-319-16101-3_23
Heikkilä, V.T., Damian, D., Lassenius, C., Paasivaara, M., 2015. A mapping study on requirements engineering in agile software development. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, pp. 199–207. https://doi.org/10.1109/SEAA.2015.70
Hughes, J., O'Brien, J., Rodden, T., Rouncefield, M., Sommerville, I., 1995. Presenting ethnography in the requirements process. In: Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95). IEEE, pp. 27–34. https://doi.org/10.1109/ISRE.1995.512543
Hummel, M., Rosenkranz, C., Holten, R., et al., 2015. The role of social agile practices for direct and indirect communication in information systems development teams. Commun. Assoc. Inf. Syst. 36 (1). https://doi.org/10.17705/1CAIS.03615
Inayat, I., Salim, S.S., Marczak, S., Daneva, M., Shamshirband, S., et al., 2015. A systematic literature review on agile requirements engineering practices and challenges. Comput. Hum. Behav. 51, 915–929. https://doi.org/10.1016/j.chb.2014.10.046
Jacobson, I., 1993. Object-Oriented Software Engineering: A Use Case Driven Approach. Pearson Education India.
Jeffries, R., 2001. Essential XP: card, conversation, confirmation. https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/.
Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., de Oliveira Neto, F.G., 2021. Requirements engineering challenges and practices in large-scale agile system development. J. Syst. Softw. 172, 110851. https://doi.org/10.1016/j.jss.2020.110851
Kassab, M., 2015. The changing landscape of requirements engineering practices over the past decade. In: 2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE), pp. 1–8. ISSN: 2329–6356. https://doi.org/10.1109/EmpiRE.2015.7431299
Kautz, K., 2009. Customer and user involvement in agile software development. In: Agile Processes in Software Engineering and Extreme Programming: 10th International Conference, XP 2009. Proceedings 10. Springer, pp. 168–173. https://doi.org/10.1007/978-3-642-01853-4_22
Klärck, P., Rantanen, J., Härkönen, J., 2009. Executable requirements in practice. In: Agile Processes in Software Engineering and Extreme Programming: 10th International Conference, XP 2009. Proceedings 10. Springer, pp. 226–227. https://doi.org/10.1007/978-3-642-01853-4_42
Kniberg, H., 2015. Scrum and XP From The Trenches. Lulu. com.
Kochbati, T., Li, S., Gérard, S., Mraidha, C., 2021. From user stories to models: a machine learning empowered automation. In: 9th International Conference on Model-Driven Engineering and Software Development. Vol. 1. Science and Technology Publications, pp. 28–40. https://doi.org/10.5220/0010197800280040
Kustiawan, Y.A., Lim, T.Y., 2023. User stories in requirements elicitation: a systematic literature review. In: 2023 IEEE 8th International Conference on Software Engineering and Computer Systems (ICSECS), pp. 211–216. https://doi.org/10.1109/ICSECS58457.2023.10256364
Köhl, M.A., Baum, K., Langer, M., Oster, D., Speith, T., Bohlender, D., et al., 2019. Explainability as a non-functional requirement. In: 2019 IEEE 27th International Requirements Engineering Conference (RE), pp. 363–368. ISSN: 2332–6441. https://doi.org/10.1109/RE.2019.00046
Larman, C., Basili, V.R., 2003. Iterative and incremental developments. a brief history. Computer 36 (6), 47–56. https://doi.org/10.1109/MC.2003.1204375
Lebovitz, S., Lifshitz-Assaf, H., Levina, N., et al., 2022. To engage or not to engage with AI for critical judgments: how professionals deal with opacity when using AI for medical diagnosis. Organ. Sci. 33 (1), 126–148. https://doi.org/10.1287/orsc.2021.1549
Leffingwell, D., 2010. Agile Software Requirements: Lean Requirements Practices For Teams, Programs, and The Enterprise. Addison-Wesley Professional.
Levina, N., Vaast, E., 2005. The emergence of boundary spanning competence in practice: implications for implementation and use of information systems. MIS Q. 29 (2), 335–363. https://doi.org/10.2307/25148682
Lindland, O.I., Sindre, G., Solvberg, A., 1994. Understanding quality in conceptual modeling. IEEE Softw. 11 (2), 42–49. https://doi.org/10.1109/52.268955
Liskin, O., 2015. How artifacts support and impede requirements communication. In: Requirements Engineering: Foundation for Software Quality: 21st International Working Conference, REFSQ 2015, Essen, Germany, March 23–26, 2015. Proceedings 21. Springer, pp. 132–147. https://doi.org/10.1007/978-3-319-16101-3_9
Liskin, O., Pham, R., Kiesling, S., Schneider, K., 2014. Why we need a granularity concept for user stories. In: Agile Processes in Software Engineering and Extreme Programming:

15th International Conference, XP 2014, Rome, Italy, May 26–30, 2014. Proceedings 15, pp. 110–125. https://doi.org/10.1007/978-3-319-06862-6_8
Lucassen, G., Dalpiaz, F., van der, W. J. M. E.M., Brinkkemper, S., 2016a. The use and effectiveness of user stories in practice. In: Requirements Engineering: Foundation for Software Quality: 22nd International Working Conference, REFSQ, Proceedings 22. Springer, pp. 205–222. https://doi.org/10.1007/978-3-319-30282-9_14
Lucassen, G., Dalpiaz, F., Van Der Werf, J. M. E.M., Brinkkemper, S., 2015. Forging high-quality user stories: towards a discipline for agile requirements. In: IEEE 23rd International Requirements Engineering Conference (RE). IEEE, pp. 126–135. https://doi.org/10.1109/RE.2015.7320415
Lucassen, G., Dalpiaz, F., van der Werf, J. M. E.M., Brinkkemper, S., 2016b. Improving agile requirements: the quality user story framework and tool. Requir. Eng. 21, 383–403. https://doi.org/10.1007/s00766-016-0250-x
Ludlow, L., 2007. The application of user stories for strategic planning. In: International Conference on Extreme Programming and Agile Processes in Software Engineering. Springer, pp. 198–202. https://doi.org/10.1007/978-3-540-73101-6_36
Mahnič, V., Hovelja, T., 2012. On using planning poker for estimating user stories. J. Syst. Softw. 85 (9), 2086–2095. https://doi.org/10.1016/j.jss.2012.04.005
Matthies, C., Heinrich, R., Wohlrab, R., et al., 2023. Investigating software engineering artifacts in devops through the lens of boundary objects. In: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering. ACM, pp. 12–21. https://doi.org/10.1145/3593434.3593441
McAvoy, J., Butler, T., 2005. A paradox of the change to user stories: the application of the theory of competing commitments. ECIS 2005 Proceedings .
Melegati, J., Wang, X., 2019. Quest: new practices to represent hypotheses in experiment-driven software development. In: Proceedings of the 2nd ACM SIGSOFT International Workshop on Software-Intensive Business: Start-ups, Platforms, and Ecosystems, pp. 13–18. https://doi.org/10.1145/3340481.3342732
Meyer, B., 2013. The ABC of software engineering. https://bertrandmeyer.com/2013/03/25/the-abc-of-software-engineering/. Accessed: 2025-07-08.
Meyer, B., 2014. Agile!: The Good, The Hype and The Ugly. Springer Science & Business Media.
Meyer, B., 2018. Making sense of agile methods. IEEE Softw. 35 (2), 91–94. Conference Name: IEEE Software. https://doi.org/10.1109/MS.2018.1661325
Miles, M.B., Huberman, A.M., 1994. Qualitative Data Analysis: An Expanded Sourcebook. SAGE.
Miller, T., 2023. Explainable AI is dead, long live explainable AI! hypothesis-driven decision support using evaluative AI. In: Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency. Association for Computing Machinery, pp. 333–342. https://doi.org/10.1145/3593013.3594001
Miller, T., Pedell, S., Sterling, L., Vetere, F., Howard, S., 2012. Understanding socially oriented roles and goals through motivational modelling. J. Syst. Softw. 85 (9), 2160–2170. https://doi.org/10.1016/j.jss.2012.04.049
Müter, L., Deoskar, T., Mathijssen, M., Brinkkemper, S., Dalpiaz, F., 2019. Refinement of user stories into backlog items: linguistic structure and action verbs: research preview. In: Requirements Engineering: Foundation for Software Quality: 25th International Working Conference, REFSQ, Proceedings 25. Springer, pp. 109–116. https://doi.org/10.1007/978-3-030-15538-4_7
Nord, R.L., Tomayko, J.E., 2006. Software architecture-centric methods and agile development. IEEE Softw. 23 (2), 47–53. https://doi.org/10.1109/ms.2006.54
Nuseibeh, B., Easterbrook, S., 2000. Requirements engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. Association for Computing Machinery, pp. 35–46. https://doi.org/10.1145/336512.336523
O'hEocha, C., Conboy, K., 2010. The role of the user story agile practice in innovation. In: International Conference on Lean Enterprise Software and Systems. Springer, pp. 20–30. https://doi.org/10.1007/978-3-642-16416-3_3
Pareto, L., Eriksson, P., Ehnebom, S., 2010. Architectural descriptions as boundary objects in system and design work. In: Model Driven Engineering Languages and Systems: 13th International Conference, MODELS 2010, Proceedings, Part II 13, pp. 406–419. https://doi.org/10.1007/978-3-642-16129-2_29
Paré, G., Trudel, M.-C., Jaana, M., Kitsiou, S., 2015. Synthesizing information systems knowledge: a typology of literature reviews. Inf. Manage. 52 (2), 183–199. https://doi.org/10.1016/j.im.2014.08.008
Patton, J., Economy, P., 2014. User Story Mapping: Discover The Whole Story, Build The Right Product. O'Reilly Media, Inc.
Potts, C., Hsi, I., 1997. Abstraction and context in requirements engineering: toward a synthesis. Ann. Softw. Eng. 3 (1), 23–61. https://doi.org/10.1023/A:1018994106507
Power, K., 2011. Using silent grouping to size user stories. In: Agile Processes in Software Engineering and Extreme Programming: 12th International Conference, XP. Proceedings 12. Springer, pp. 60–72. https://doi.org/10.1007/978-3-642-20677-1_5
Raharjana, I.K., Siahaan, D., Fatichah, C., 2021. User stories and natural language processing: a systematic literature review. IEEE Access 9, 53811–53826. https://doi.org/10.1109/ACCESS.2021.3070606
Ralph, P., 2018. The two paradigms of software development research. Sci. Comput. Program. 156, 68–89. https://doi.org/10.1016/j.scico.2018.01.002
Ramesh, B., Cao, L., Baskerville, R., 2010. Agile requirements engineering practices and challenges: an empirical study. Inf. Syst. J. 20 (5), 449–480. https://doi.org/10.1111/j.1365-2575.2007.00259.x
Read, A., De Vreede, G.-J., Briggs, R.O., et al., 2012. Describing agile requirements development and communication using complex adaptive systems theory. In: AMCIS 2012, Proceedings. Assoc Information Systems. .
Resketi, M.R., Motameni, H., Nematzadeh, H., Akbari, E., 2020. Automatic summarising of user stories in order to be reused in future similar projects. IET Softw. 14 (6), 711–723. https://doi.org/10.1049/iet-sen.2019.0182
Robeer, M., Lucassen, G., Van Der Werf, J. M. E.M., Dalpiaz, F., Brinkkemper, S., 2016. Automated extraction of conceptual models from user stories via NLP. In: 2016 IEEE 24th International Requirements Engineering Conference (RE). IEEE, pp. 196–205. https://doi.org/10.1109/RE.2016.40
Robinson, D., Cabrera, C., Gordon, A.D., Lawrence, N.D., Mennen, L., 2025. Requirements are all you need: the final frontier for end-user software engineering. ACM Trans. Softw. Eng. Method. 34 (5), 1–22. https://doi.org/10.1145/3708524
Rocha Silva, T., Winckler, M., Bach, C., 2020. Evaluating the usage of predefined interactive behaviors for writing user stories: an empirical study with potential product owners. Cognit. Technol. Work 22 (3), 437–457. https://doi.org/10.1007/s10111-019-00566-3
Ross, D.T., 1978. Origins of the APT language for automatically programmed tools. In: History of Programming Languages. ACM, pp. 279–338. https://doi.org/10.1145/800025.1198374
Ross, D.T., Schoman, K.E., 1977. Structured analysis for requirements definition. IEEE Trans. Softw. Eng. (1), 6–15. https://doi.org/10.1109/TSE.1977.229899
Sandberg, J., Alvesson, M., 2021. Meanings of theory: clarifying theory through typification. J. Manage. Stud. 58 (2), 487–516. https://doi.org/10.1111/joms.12587
Schwaber, K., 1997. Scrum development process. In: Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings 16 October 1995, Austin, Texas. Springer, pp. 117–134. https://doi.org/10.1007/978-1-4471-0947-1_11
Schön, E.-M., Thomaschewski, J., Escalona, M.J., et al., 2017. Agile requirements engineering: a systematic literature review. Comput. Stand. Interfaces 49, 79–91. https://doi.org/10.1016/j.csi.2016.08.011
Seaman, C.B., 1999. Qualitative methods in empirical studies of software engineering. IEEE Trans. Softw. Eng. 25 (4), 557–572. https://doi.org/10.1109/32.799955
Sedano, T., Ralph, P., Péraire, C., 2019. The product backlog. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, pp. 200–211. https://doi.org/10.1109/icse.2019.00036
Shore, J., Warden, S., 2021. The Art of Agile Development. O'Reilly Media, Inc.
Sjøberg, D. I.K., Bergersen, G.R., 2022. Construct validity in software engineering. IEEE Trans. Softw. Eng. 49 (3), 1374–1396. https://doi.org/10.1109/TSE.2022.3176725
Sjøberg, D. I.K., Dybå, T., Jørgensen, M., 2007. The future of empirical methods in software engineering research. Future of Software Engineering (FOSE'07) , 358–378. https://doi.org/10.1109/FOSE.2007.30
Śmiałek, M., Bojarski, J., Nowakowski, W., Straszak, T., 2005. Writing coherent user stories with tool support. In: Extreme Programming and Agile Processes in Software Engineering: 6th International Conference, XP 2005, Sheffield, UK, June 18–23, 2005. Proceedings 6. Springer, pp. 247–250. https://doi.org/10.1007/11499053_38
Sommerville, I., Rodden, T., Sawyer, P., Bentley, R., Twidale, M., 1993. Integrating ethnography into the requirements engineering process. In: Proceedings of the IEEE International Symposium on Requirements Engineering. IEEE Comput. Soc. Press, p. 165–173. https://doi.org/10.1109/isre.1993.324821
Sommerville, I., Sawyer, P., 1997. Requirements Engineering: A Good Practice Guide. John Wiley & Sons, Inc.
Sporsem, T., Hatling, M., Tkalich, A., Stol, K.-J., 2023. Knowns and unknowns: an experience report on discovering tacit knowledge of maritime surveyors. In: International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ). Springer. https://doi.org/10.1007/978-3-031-29786-1_22
Spurrier, G., Topi, H., 2022. Synergistically employing user stories and use cases in the practice and teaching of systems analysis and design. Commun. Assoc. Inf. Syst. 51 (1), 140–178. https://doi.org/10.17705/1CAIS.05108
Star, S.L., 2010. This is not a boundary object: reflections on the origin of a concept. Sci. Technol. Hum. Values 35 (5), 601–617. Sage Publications Inc. https://doi.org/10.1177/0162243910377624
Star, S.L., Griesemer, J.R., 1989. Institutional ecology, 'translations' and boundary objects: amateurs and professionals in Berkeley's museum of vertebrate zoology, 1907-39. Soc. Stud. Sci. 19 (3), 387–420. https://doi.org/10.1177/030631289019003001
Stol, K.-J., 2024. Teaching theorizing in software engineering research. In: Handbook on Teaching Empirical Software Engineering. Springer Nature, Cham, pp. 31–69. https://doi.org/10.1007/978-3-031-71769-7_3
Stol, K.-J., Fitzgerald, B., 2018. The ABC of software engineering research. ACM Trans. Softw. Eng. Methodol. 27 (3), 1–51. https://doi.org/10.1145/3241743
Stray, V., Hanssen, G.K., Barbala, A., Smite, D., Stol, K.-J., 2025. What is generative AI good for? introduction to the special issue on generative AI in software engineering. Inf. Softw. Technol. 187. https://doi.org/10.1016/j.infsof.2025.107857
Tiwana, A., 2004. An empirical study of the effect of knowledge integration on software development performance. Inf. Softw. Technol. 46 (13), 899–906. https://doi.org/10.1016/j.infsof.2004.03.006
Umar, M.A., Lano, K., 2024. Advances in automated support for requirements engineering: a systematic literature review. Requir. Eng. 29 (2), 177–207. https://doi.org/10.1007/s00766-023-00411-0
Van Can, A.T., Dalpiaz, F., 2024. Requirements information in backlog items: content analysis. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 305–321. https://doi.org/10.1007/978-3-031-57327-9_19
Van Lamsweerde, A., 2000. Requirements engineering in the year 00: a research perspective. In: Proceedings of the 22nd International Conference on Software Engineering, pp. 5–19. https://doi.org/10.1145/337180.337184
Van Lamsweerde, A., 2001. Goal-oriented requirements engineering: a guided tour. In: Proceedings Fifth IEEE International Symposium on Requirements Engineering. IEEE Comput. Soc, p. 249–262. https://doi.org/10.1109/isre.2001.948567
Vidgen, R., Wang, X., 2009. Coevolving systems and the organization of agile software development. Inf. Syst. Res. 20 (3), 355–376. https://doi.org/10.1287/isre.1090.0237
Viller, S., Sommerville, I., 1999. Social analysis in the requirements engineering process: from ethnography to method. In: Proceedings IEEE International Symposium on Requirements Engineering, pp. 6–13. https://doi.org/10.1109/ISRE.1999.777980

Wang, X., Zhao, L., Wang, Y., Sun, J., et al., 2014. The role of requirements engineering practices in agile development: an empirical study. In: Zowghi, D., Jin, Z. (Eds.), Requirements Engineering. Springer, pp. 195–209. https://doi.org/10.1007/978-3-662-43610-3_15

Wautelet, Y., Velghe, M., Heng, S., Poelmans, S., Kolp, M., 2018. On modelers ability to build a visual diagram from a user story set: a goal-oriented approach. In: Requirements Engineering: Foundation for Software Quality: 24th International Working Conference, REFSQ, Proceedings 24. Springer, pp. 209–226. https://doi.org/10.1007/978-3-319-77243-1_13

Wieringa, R., Maiden, N., Mead, N., Rolland, C., 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir. Eng. 11 (1), 102–107. https://doi.org/10.1007/s00766-005-0021-6

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. ACM, London England United Kingdom, pp. 1–10. https://doi.org/10.1145/2601248.2601268

Wohlrab, R., Pelliccione, P., Knauss, E., Larsson, M., et al., 2019. Boundary objects and their use in agile systems engineering. J. Softw. Evol. Process 31 (5), e2166. https://doi.org/10.1002/smr.2166

Yue, K., 1987. What does it mean to say that a specification is complete? In: Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design.

Zave, P., 1997. Classification of research efforts in requirements engineering. ACM Comput. Surv. 29 (4), 315–321. https://doi.org/10.1145/267580.267581

Zhang, Z., Rayhan, M., Herda, T., Goisauf, M., Abrahamsson, P., 2024. LLM-Based agents for automating the enhancement of user story quality: an early report. In: International Conference on Agile Software Development. Springer Nature, pp. 117–126. https://doi.org/10.1007/978-3-031-61154-4_8