Towards the Art of Writing Agile Requirements with User Stories, Acceptance Criteria, and Related Constructs

António M. S. Ferreira[1], Alberto Rodrigues da Silva[1] and Ana C. R. Paiva[2]
[1]*INESC-ID - Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal*
[2]*INESC TEC, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal*

Keywords: Agile Specification of Requirements, User Stories, Epics, Acceptance Criteria, Writing Guidelines.

Abstract: Nowadays, more organizations adopt agile methodologies to guarantee short and frequent delivery times. A plethora of novel approaches and concepts regarding requirements engineering in this context are emerging. User stories are usually informally described as general explanations of software features, written from end-users perspective, while acceptance criteria are high-level conditions that enable their verification. This paper focuses on the art of writing user stories and acceptance criteria, but also on their relationships with other related concepts, such as quality requirements. In the pursuance of deriving guidelines and linguistic patterns to facilitate the writing of requirements specifications, a systematic literature review was conducted to provide a cohesive and comprehensive analysis of such concepts. Despite considerable research on the subject, no formalized model and systematic approach to assist this writing. We provide a coherent analysis of these concepts and related linguistic patterns supported by a running example of specifications built on top of ITLingo RSL, a publicly available tool to enforce the rigorous writing of specification artefacts. We consider that adopting and using the guidelines and patterns from the present discussion contribute to writing better and more consistent requirements.

1 INTRODUCTION

Agile methodologies have gained an increasing adoption across multiple organizations to address the rapidly changing nature of the markets in which they operate. The time constraints in these competitive contexts are such that software development teams often initiate the development before the conclusion of requirements analysis and design phases, resulting in requirements that tend to evolve across different project iteration cycles (Cao and Ramesh, 2008). Nevertheless, the crucial importance of having mature requirements to deliver the expected results successfully is well-identified (Shah and Jinwala, 2015). The achievement of the customer's needs depends on the common understanding of such requirements by all parties involved.

In agile contexts, user stories have emerged as an uncluttered artefact for expressing requirements while supporting this ever-changing nature, and their adoption has been widely accepted (Lucassen, Dalpiaz, Van der Werf and Brinkkemper, 2016). They are usually informally described as general textual explanations of software features, enforcing the end-user perspective (Lucassen, Dalpiaz, Van der Werf, and Brinkkemper, 2016). User stories are also usually strengthened through the specification of complementary artefacts, such as acceptance criteria and quality requirements, which provide further detail. While such adoption is widespread, the practices and patterns to improve and assess their written quality are limited and not generalized. Existing approaches resort to generic guidance (Heck and Zaidman, 2014) and mnemonic heuristics such as INVEST (Abdou, Kamthan & Pankaj, 2014).

This paper reviews notions and patterns commonly used in the agile requirements specification. It also proposes and discusses writing guidelines based on a running example. This example is specified with the ITLingo RSL Excel template, an easy-to-use tool that allows the specification of various requirements constructs and enforces their consistency.

This paper is structured in 7 sections. Section 2 introduces the relevant concepts and definitions under study: user stories, acceptance criteria, and quality requirements. Section 3 presents the state of the art of the mentioned concepts and discusses their

Ferreira, A., Rodrigues da Silva, A. and Paiva, A.
Towards the Art of Writing Agile Requirements with User Stories, Acceptance Criteria, and Related Constructs.
DOI: 10.5220/0011082000003176
In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022)*, pages 477-484
ISBN: 978-989-758-568-5; ISSN: 2184-4895

interdependent relationships. Section 4 enlists the proposed guidelines. Section 5 introduces the BillingSystem example and the ITlingo RSL-Excel template, enabling and supporting the discussion of the proposed guidelines in Section 6. Finally, Section 7 (Conclusion) summarizes the achievements and identifies the future work on the topic.

2 BACKGROUND

The agile specification of system requirements involves some key concepts and relationships, as shown in Figure 1: user stories, epics, themes, acceptance criteria, themes, and quality requirements.

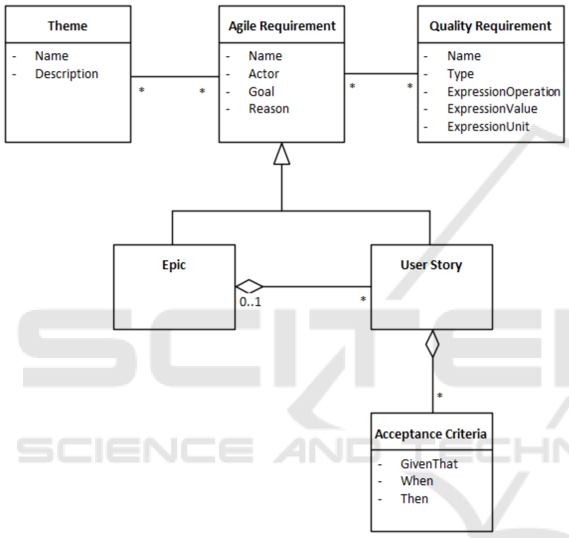

Figure 1: Conceptual model of agile requirements and related concepts (as discussed in this paper).

User Stories. Beck, Lucassen, and Brinkkemper (2017) introduced in 1999, for the first time, the term "user stories" as a short-loose description of a customer's needs. Agile methodologies practitioners handily adopted user stories as the basis for building features (Lucassen, Dalpiaz and Van der Wef, 2016) and for starting conversations between different stakeholders to mitigate incomplete requirements or communication flaws. Several proposals to redefine and further extend the concept of user stories have been attempted: Mike Cohn (Cohn, 2004) proposed them as a description of the required functionality for a system with value to and from the perspective of end-users or other stakeholders. It attempts to catch on paper the fundamental elements of a requirement: who is it for, what functionality is it to be developed, and why is it essential (Lucassen, Dalpiaz, Van der Werf and Brinkkemper, 2015). User stories have since evolved to resemble and follow standardized linguistic patterns or templates, such as the popular and widely adopted Connextra template: "As an <actor>, I want <goal> so that <reason>" (Dalpiaz and Brinkkemper, 2018). This enforces only the essential details to be considered. Other recommendations to assess and enhance the quality of user stories have been proposed and discussed, e.g., the INVEST and QUS (Lucassen, Dalpiaz, Van der Werf and Brinkkemper, 2015) frameworks.

Epics. User stories should be kept simple and straightforward to guarantee their deliverance within a single iteration of an agile sprint. So, an epic appears, by definition, as a larger user story, aggregating smaller ones. Epics represent larger or vague features and commonly follow the same patterns as user stories but are written in a more general or abstract way. They assist in guaranteeing that each user story is independently implementable and estimable (Pandit and Tahiliani, 2015).

Acceptance Criteria. The specification of user stories is usually attached to acceptance criteria constructs. Acceptance criteria are conditions of satisfaction that complement user stories by defining boundaries that validate their complete specification and implementation. They may encompass functional behaviour, business rules, and quality aspects to be tested. Martin Fowler proposed the popular Given-When-Then[1] template as: “Given that <aCondition> when <anAction> then <aDesiredConsequence>”. The purpose is to understand user stories better while also enabling acceptance tests generation for the code to be verified and validated at each sprint iteration.[1]

Themes. User stories and epics that share a standard classifier or characteristic can be grouped around a common theme by associating a label. Themes are tags that usually represent general categories within a system (e.g., "User Authentication" for user account management responsibilities) and help classify user stories under a given criterion.

Quality Requirements. Quality requirements represent cross-cutting concerns that a given component or system should be satisfied. In general, they establish multiple relationships among themselves and other requirements. A concrete quantitative expression ideally defines quality

[1] https://martinfowler.com/bliki/GivenWhenThen.html

requirements. E.g., a quality requirement for ensuring the complexity of a given password through multiple rules could be specified in a regex expression to avoid the potential ambiguity of rules written in a natural language.

3 RELATED WORK

Specifications of system requirements use different types of requirements, which have a variety of dependencies among them and related elements (Verelst et al., 2013). For instance, the ITLingo RSL language supports goals, constraints, user stories, use cases, functional and quality requirements (Silva, 2019).

In the scope of agile approaches, user stories and related concepts are the most popular constructs used to write requirements. Product owners commonly write user stories in natural languages due to their ease of use. But requirements defined in natural languages are ambiguous, inconsistent and incomplete, as extensively discussed in the literature (Pohl, 2010; Silva, 2014; INCOSE, 2019).

The Connextra template helps writing user stories, but it does not enforce any reasoning on the semantics quality of the writing. For that reason, some frameworks have emerged to assess the quality of user stories, such as INVEST or QUS.

INVEST framework stands for Independent (self-contained), Negotiable (not an explicit contract), Valuable (that creates value for end-users), Estimable (its size can be predicted), Small (to fit in a time-box iteration), and Testable (ability to have attached acceptance criteria). INVEST has been widely adopted by the industry (Abdou, Kamthan and Pankaj, 2014). It facilitates the management of the product backlog by promoting the breakdown of large user stories and conveniently fits agile mindsets. However, INVEST does not concern the writing quality of user stories' attributes, like their consistent writing.

QUS (Quality User Story) framework (Lucassen et al., 2015) includes 14 criteria regarding syntactic, semantic, and pragmatic qualities as a comprehensive quality assessment enabler. Syntactically, user stories should be *atomic* (one requirement for one feature), *minimal* (solely contain a role, a goal, and an optional reason), and *well-formed*. Semantically, one should verify for *conflict-free* (inconsistency between user stories), *conceptually sound* (the feature and its goal must be both ambiguously expressed), *problem-oriented* (represents a problem and not a solution), and, finally, *unambiguous* (avoids terms or abstractions breeding different interpretations). Pragmatically, user stories should be *complete* (a given set of user stories creates a feature-complete application), *independent*, *scalable*, *uniform* (all following the same pattern), and *unique*. While comprehensive, this framework may appear too abstract, especially compared with INVEST one.

Acceptance criteria appeared as an extension to user stories, complementing them with the objectives within a given story. They concern functional behaviour, quality characteristics and business logic. The most popular linguistic pattern written in natural languages, both by the industry and the academy, is the "Given-When-Then" (GWT) pattern. This pattern structures the criteria into a test with three sections: (i) Given, which describes the pre-conditions, (ii) When, to specify the behaviour to be implemented, and (iii) Then, describing the changes/results (post-conditions) expected to be verified. Respecting this structure, all the stakeholders are accurately aware of what conditions must be met to finish the user story. The complete GWT sentence should be kept simple and small. If needed, it is recommendable to break down large acceptance criteria into multiple smaller ones, resulting in an assembled step sequence. Writing acceptance criteria is much easier when the user story is independent. Moreover, acceptance criteria are the fundamental building blocks for creating test cases.

Regarding the writing of use cases, Silva (2021) proposes several guidelines for adopting controlled natural languages (CNL) that help reduce ambiguity without compromising their inherent expressiveness. CNLs promote the shared understanding between all parts in a rigorous human-readable fashion. They can be regarded as subsets of natural languages, with restricted grammars (syntax) and narrowed set of terms (semantics). Although the original proposals are based upon multiple constructs – like business-level elements (Silva, 2017), data entities (Silva and Savic, 2021), or use cases and scenarios (Silva, 2021) – the rationale behind the proposed guidelines maintains its relevance when specifying requirements in agile contexts. Therefore, they will serve as a basis for the guidelines presented in this paper, which aim to enforce further the quality properties referred by QUS and INVEST frameworks.

4 WRITING GUIDELINES

This section summarizes guidelines for writing user stories, quality requirements, and acceptance criteria. These guidelines gather the hints and information that we analyzed in the related work and are also inspired by Silva's guidelines for writing use cases and scenarios (2021). These guidelines enforce both writing styles and semantic concerns.

4.1 User Stories

G.US.1. *Use popular linguistic patterns to guide the writing of user stories* (e.g., use the Connextra template). Linguistic patterns are the basis for writing coherent and rigorous user stories.
G.US.2. *Identify user stories by a unique Id* to allow unambiguous referencing.
G.US.3. *Define a short but suggestive name to the user story*. Choose simple but effective nouns, e.g., as Silva (2021) discussed.
G.US.4. If relevant, *classify a user story by one or more themes.* This classification makes sense to provide greater context and facilitate navigation in system specifications with many requirements. It may also link the functionalities to organizational aspects of the company.
G.US.5. Distinguish user stories from epics by adopting a prefix that recognizes their type, e.g., "us_" for user stories and "ep_" or "epic_" for epics.
G.US.6. *When defining the user story's actor, avoid using job titles and adopt user roles* instead. Notice that some functionalities described as user stories may involve *other stakeholders* (e.g., System Administrator, System Architect, Customer), *not necessarily user roles*.
G.US.7. Use a consistent and straightforward syntactical structure for writing user stories. Attempt to stick to the domain terms; for instance, use the "verb-noun" structure, with a predefined set of strong verbs and strong nouns to maintain the phrases unambiguous and straightforward. Use strong specific verbs (e.g., Create, Update) and specific nouns (e.g., Invoice, User, Password) to avoid ambiguity.
G.US.8. *Define the successful path for each user story*. This most common path allows focusing on the reasoning of the functionality to implement. If relevant, describe alternatives or exception situations only then.
G.US.9. Review the writing quality of the user stories by following a well-defined quality framework, such as the INVEST or QUS frameworks.
G.US.10. *Define libraries of reusable user stories,* e.g., for cross-cutting aspects such as authentication, authorization, logging, and auditing features. *Apply and reuse these reusable requirements in your system specification* if appropriate. This reuse mechanism would reduce writing costs and promote the sharing of good examples and writing practices.

4.2 Quality Requirements

G.QR.1. *Identify quality requirements by a unique Id with a specific prefix* (e.g., "qr_" or "quality_"), to allow unambiguous referencing.
G.QR.2. *Define a name to the quality requirement*, explicitly stating its scope.
G.QR.3. Break down *larger quality requirements into smaller ones*. Different user stories may have associated the same quality requirement. It is essential to keep them specific to a single concern.
G.QR.4. *Express quality requirements through an expression that can be implemented.* This step is positively helpful for later writing the acceptance criteria.
G.QR.5. *Describe what is expected* to facilitate the recognition. Particularly relevant when the quantitative expression is not in human-readable language.

4.3 Acceptance Criteria

G.AC.1. *Use popular linguistic patterns when writing the acceptance criteria* (e.g., use the "Given-When-Then" pattern). Linguistic patterns are the basis for achieving coherent and rigorous acceptance criteria.
G.AC.2. *Identify the artefact by a unique Id with a specific prefix* (e.g., "ac_", "test_") to allow their unambiguous referral.
G.AC.3. *Explicitly state the associated user story by its Id and Name* to facilitate recognition and navigability through the specifications.
G.AC.4. Use a consistent and straightforward syntax to write each fragment of the acceptance criteria. Attempt to stick to the domain terms. For example, in the GWT structure, the "GivenThat" fragment can start with an expression like "I <do something>", "I am <something>", or "I am in <some state>" (see Table 4).
G.AC.5. *Avoid writing general and vague acceptance criteria* that would be hard to be tested. Acceptance criteria should be kept small to be singularly testable.
G.AC.7. *Identify if the acceptance criteria concern a success or a failure case*. This information is relevant to distinguish and focus on its goal.
G.AC.8. *Enrich the acceptance criteria with failure cases*. These criteria shall provide further robustness by testing the implementation against bad inputs.

5 RUNNING EXAMPLE

Henceforward, this paper uses a running example to support the discussion.

5.1 Informal Description

This example is based on a simple version of the "BillingSystem", a fictitious "invoice management system" used for research and academic purposes. The original example discussed in (Silva and Savic, 2021; Silva, 2021) was extended to include usability concerns, as follows (see Spec.1):

```
BillingSystem is a software system that shall
allow end-users to manage data entities like
customers, products, and invoices. […]
Any end-user shall log in, log out, recover
password, and update his personal
information. […]
The system shall allow the user-operator to
create new invoices (with respective invoice
details). Still, before sending an invoice to
the customer, the invoice shall be formally
approved by the user-manager. After such
approval, the user-operator can issue and
send that invoice electronically by email[…].
Concerning users' data, their stored
passwords must be hashed and salted. Critical
information should be secured. […]
The system's user experience shall be
intuitive to learn and easy to use it. […]
```

Spec. 1: Description of the BillingSystem case study.

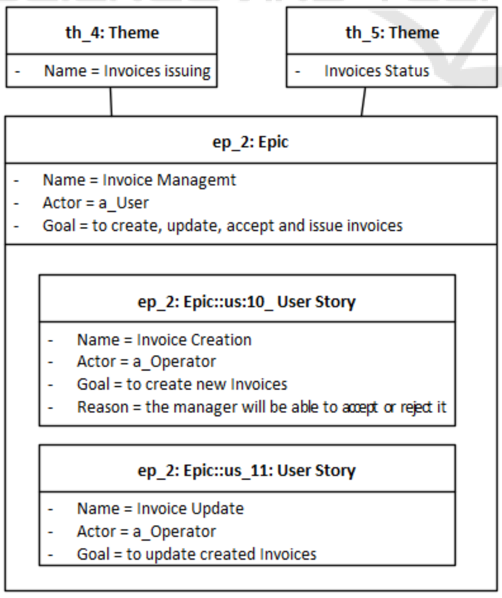
th_4: Theme
- Name = Invoices issuing
th_5: Theme
- Invoices Status
ep_2: Epic
- Name = Invoice Managemt
- Actor = a_User
- Goal = to create, update, accept and issue invoices
ep_2: Epic::us:10_ User Story
- Name = Invoice Creation
- Actor = a_Operator
- Goal = to create new Invoices
- Reason = the manager will be able to accept or reject it
ep_2: Epic::us_11: User Story
- Name = Invoice Update
- Actor = a_Operator
- Goal = to update created Invoices

Figure 2: An epic with user stories classified with themes.

5.2 Dataset of Specifications

To evaluate and enable the discussion upon the usage of the proposed guidelines, a set of agile constructs based upon the requirements of the BillingSystem was populated in the ITLingo RSL Excel template.

The ITLingo RSL Excel[2] template proposes to assist the systematization of producing rigorous requirements artefacts, mitigating the ambiguity of natural languages by instating a restricted syntax. It is assembled with constructs and predefined templates and attributes to enforce domain analysis. Its current implementation supports already the specification of user stories and quality requirements. It was extended to allow the specification of acceptance criteria and themes.

Tables 1-4 present "slices" of examples taken from the Excel dataset prepared throughout this research are presented in Tables 1 to 4. They were carefully specified from Spec.1 by applying the proposed guidelines in Section 5, and the given extracts were chosen as they refer to general features common across distinct projects.

For instance, Table 1 presents a set of themes specified with an identifier, name and brief description. Table 2 displays user stories and epics, written according the Connextra template and associated with the respective themes and quality requirements. The column *partOf* establishes the relationship between user stories and epics.

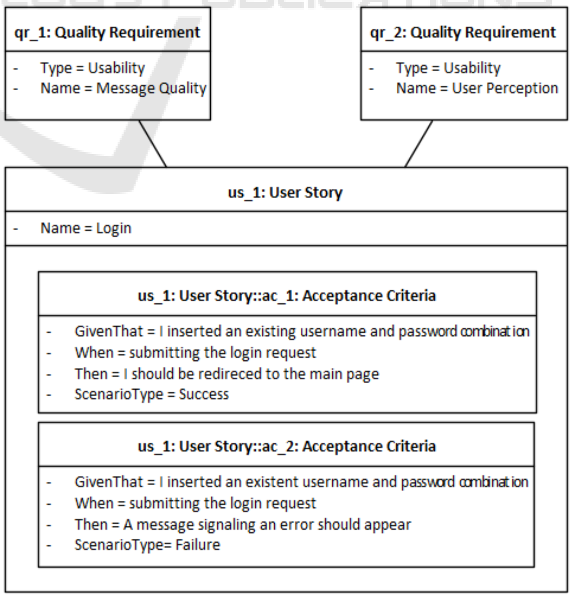
qr_1: Quality Requirement
- Type = Usability
- Name = Message Quality
qr_2: Quality Requirement
- Type = Usability
- Name = User Perception
us_1: User Story
- Name = Login
us_1: User Story::ac_1: Acceptance Criteria
- GivenThat = I inserted an existing username and password combination
- When = submitting the login request
- Then = I should be redireced to the main page
- ScenarioType = Success
us_1: User Story::ac_2: Acceptance Criteria
- GivenThat = I inserted an existent username and password combination
- When = submitting the login request
- Then = A message signaling an error should appear
- ScenarioType= Failure

Figure 3: A user story with acceptance criteria and quality requirements relationships.

Table 1: Selected examples of themes defined in the dataset.

Id	**Name**	**Description**
th_1	User Authentication	Users should log in, log out and recover their passwords.
th_4	Invoices Issuing	Operators and managers should be able to coordinate the issuance of invoices.
th_5	Invoices Status Monitoring	Operators and managers should be able to overview invoices and their status.

Table 2: Selected examples of user stories defined in the dataset.

			As a	**I want**	**So that**			
Id	**Name**	**Type**	**Actor**	**Goal**	**Reason**	**Theme**	**Part Of**	**QRs**
us_1	Login	User Story	a_User	to login	the system can authenticate, and trust me	th_1		qr_1, qr_2
ep_2	Invoice Management	Epic	a_User	to create, update, accept and issue invoices		th_4, th_5		
us_10	Invoice Creation	User Story	a_Operator	to create new invoices	the manager will be able to accept or reject it	th_4	ep_2	qr_2
us_11	Invoice Update	User Story	a_Operator	to update created invoices		th_4	ep_2	qr_2
us_12	Invoice Acceptance	User Story	a_Manager	To accept or reject invoices	they can be issued or discarded	th_4	ep_2	qr_1

Table 3: Selected examples of quality requirements defined in the dataset.

Id	**Name**	**Type**	**Op**	**Value**	**Metric**	**PartOf**	**Description**
qr_2	User perception	Usability					Forms filling shall be intuitive, simple, and quick.
qr_2_1	Forms auto-validator	Usability	<	2	Error PerTask	qr_2	When filling in a form, a user shall not err in filling the form with auto-validator due to wrong fields (e.g., error in entering data type or required field to fill) not more than 2 times.
qr_2_1_1	Password Validator	Usability	=	^(?=.*?[a-zA-Z]).{8,}$	Regular Expression	qr_2_1	Passwords shall contain 8 characters.
qr_4_1	Database Encryption	Security				qr_4	Critical data shall be encrypted with the Data Encryption Standard (DES).

Table 4: Selected examples of acceptance criteria defined in the dataset.

US Id	**User story**	**AC Id**	**Given That**	**When**	**Then**	**Scenario Type**
us_1	Login	ac_1	I inserted an existing username and password combination	submitting the login request	I should be redirected to the main page	Success
		ac_2	I inserted an inexistent username and password combination	submitting the login request	a message signalling an error should appear	Failure
us_3	Logout	ac_15	I'm authenticated	tapping the log out button	I should be redirected to the login page	Success
us_10	Invoice creation	ac_20	I'm a user-operator	submitting a valid invoice creation request	the invoice should be created with pending status	Success

Table 3 exhibits examples of quality requirements, specified through a quantitative expression. Table 4 presents acceptance criteria specified through both an identifier, a name, and the enforcement of the GTW pattern. The property Scenario Type was introduced to expand the acceptance criteria further to include the specification of bad input scenarios. Figures 2 and 3 show the relationships between these different artefacts. Fig.2 shows that larger features (e.g., invoice management), represented through epics, can be partitioned into other user stories. The association with multiple cross-cutting themes links the specification artefact to one or more organizational aspects of the project, which frames the goal of the user story. Fig.3 further schematizes the relationship of user stories with acceptance criteria and quality requirements. A single user story can be complemented with several acceptance criteria, which enforce the application of the quality requirements attached to a given story as conditions of success.

6 DISCUSSION

The guidelines proposed above exploit both the relationships between the different artefacts and the expressiveness of the RSL language.

The first step to achieve cohesive specifications (both intra- and inter-projects) concerns adopting linguistic patterns, as they enforce the same writing mindset and conscientiousness. Fig.2 schematizes the intrinsic relationships of user stories, epics, and themes in the BillingSystem project. The presented epic (ep_2 Invoice Management from Table 2) institutes the manipulation of invoices, which are the prime model in the BillingSystem and which encompasses creation (us_10), update (us_11), and issuance (us_12) capabilities. This large feature is broken into three individual user stories, independently implementable and testable. Nevertheless, the functionality is framed within its context by containing the stories explicitly within an epic. This empowers and facilitates agile activities such as story priority inference or accurate story points estimation for larger features.

Moreover, by relating user stories to themes, the organizational context within the requirement becomes explicit, too, driving a shared understanding among IT and non-IT team members. For consideration, the themes Invoices Issuing (th_4) and Invoices Status Monitoring (th_5) smooth the shared knowledge as developers are then aware of the need to program the domain objects in a prone to monitorization manner. At the same time, product owners can immediately recognize which parts of the system correspond to the different organization needs.

Tables 2, 3, and 4 show how the quality requirements specification drives the writing of acceptance criteria. Considering the user story us_1 Login, with associated quality requirements qr_1 Message Quality and qr_2 User Perception. It is immediately perceivable that the successful implementation of the story should encompass more than the feature itself. Given that the implementation of these additional concerns is to be verified in the acceptance testing phase, the acceptance criteria ac_1 and ac_2 (See Fig.3) both set up how the quality requirements are to be enforced, which forces their addressing and further mitigates story ambiguity. Table 4 shows how the proper specification of user stories and quality requirements allows the simplicity of the written acceptance criteria.

Given that the desired behaviour to be implemented is already specified in the quality requirement, the needed tests cases to concretize in the future can be unambiguously inferred by crossing these specifications artefacts instead of having multiple-step rules on acceptance criteria. For example, instead of writing multiple acceptance criteria to verify all the rules concerning the definition of a secure password, the regex rule expression in the quality requirement qr_2_1_1 (See Table 3) is enough to infer the complete test cases later. This contribution is crucial, as the importance of acceptance testing for achieving successful deliveries is well-attested in the community. Another promising factor in maintaining the acceptance criteria coherent and testable is the specification of small user stories and the breakdown of acceptance criteria in multiple scenarios (e.g., Success and Failure). This approach favours the quality of the written artefacts instead of the quantity to guarantee the complete specification.

The unique identifier assignment and referral of artefacts with prefixes are systematic practices to guarantee the completeness of the item's specifications without sacrificing its readability.

Finally, resorting only to project domain terms mitigates natural languages ambiguities. For example, the informal description of the BillingSystem (See Section 5.1) refers to the Creation, Update, and Approval terms within Invoice manipulation. Consequently, by following the guidelines, the user stories specifications (see Table 2) resort only to these project-specific terms, such as acceptance instead of approval.

7 CONCLUSION

This paper goes beyond reviewing the concepts used in agile specification by discussing linguistic patterns and practical guidelines to write them better. The analysis explores both the synergy between user stories and acceptance criteria and the relevance of following quality guidelines for writing user stories that can be of use. The consensus achieved enables taking a step forward by introducing new mechanisms in the requirements specification process, ensuring better specifications while respecting agile practices.

We plan to gather guidelines for writing specifications in agile contexts while also extending the ITLingo RSL-Excel template. We also plan to research transformation mechanisms to generate test cases by exploiting the written acceptance criteria and quality requirements. Indeed, prior experiences were already conducted with the ITLingo RSL language, namely on tests specification based on data entities, use cases and state machines (Silva et al., 2018), based use cases and scenarios (Gomes et al., 2021), or the broader approach from requirements to automated acceptance tests (Maciel et al., 2019; Paiva et al., 2019). We intend to explore a similar approach based on user stories and acceptance criteria.

ACKNOWLEDGMENTS

This research was partially funded by FCT 02/SAICT/2017/29360 and UIDB/50021/2020.

REFERENCES

Abdou, T., Kamthan, P., Pankaj, S. (2014). User Stories for Agile Business: INVEST, Carefully!. In AMECSE 2014.

Bick, N., Lucassen, G., Brinkkemper, S. (2017). A Reference Method for User Story Requirements in Agile Systems Development. In Proc. Workshops of REW'2017.

Cao, L., Ramesh, B. (2008). Agile Requirements Engineering Practices: An Empirical Study. In IEEE Software, vol. 25, no. 1, pp. 60-67.

Cohn, M. (2004). User Stories Applied for Agile Software Development. Addison Wesley, 1st edition.

Computer Applications.

Dalpiaz, F., Brinkkemper, S. (2018). Agile Requirements Engineering with User Stories. In Proc. RE'2018.

Gomes, A., Paiva, A.C.R., Silva, A.R. (2021). Generating Test Cases from Use Cases and Structured Scenarios: Experiences with the RSL Language. In Proc. ISD2021, AIS.

Heck, P., Zaidman, A. (2014). A Quality Framework for Agile Requirements: A Practioner's Perspective. In CoRR, vol. abs/1406.4692.

INCOSE (2019). Guide for Writing Requirements, v.3.

Lucassen, G., Dalpiaz, F., Van Der Werf, J., Brinkkemper, S. (2015). Forging High-Quality User Stories: Towards a Discipline for Agile Requirements. In Proc. RE'2015.

Lucassen, G., Dalpiaz, F., Van der Werf, J., Brinkkemper, S. (2016). Improving Agile Requirements: the Quality User Story Framework and Tool. In Requirements Engineering, volume 21, issue 3.

Lucassen, G., Dalpiaz, F., Van der Werf, J., Brinkkemper, S. (2016). The Use and Effectiveness of User Stories in Practice. In Proc. RE'2016.

Maciel, D., Paiva, A.C.R., Silva, A.R. (2019). From Requirements to Automated Acceptance Tests of Interactive Apps: An Integrated Model-based Testing Approach. In Proc. ENASE'2019, INSTICC.

Paiva, A.C.R., Maciel, D., Silva, A.R. (2019). From Requirements to Automated Acceptance Tests with the RSL Language, Communications in Computer and Information Science 1172, Springer.

Pandit, P., Tahiliani, S. (2015). A Framework for User Acceptance Testing based on User Stories and Acceptance Criteria. In International Journal of

Pohl, K. (2010). Requirements Engineering: Fundamentals, Principles, and Techniques, Springer.

Shah, U., Jinwala, D. (2015). Resolving Ambiguities in Natural Language Software Requirements: A comprehensive Survey. In ACM SIGSOFT Software Engineering Notes, vol.40, no.5, pp. 1-7.

Silva, A.R. (2014). Quality of Requirements Specifications: A Framework for Automatic Validation of Requirements. In Proc. ICEIS'2014, INSTICC.

Silva, A.R. (2017). Linguistic Patterns and Linguistic Styles for Requirements Specification (I): An Application Case with the Rigorous RSL/Business-level Language. In Proc. EuroPLOP'2017, ACM.

Silva, A.R. (2019). Rigorous Specification of Use Cases with the RSL Language. In Proceedings of the ISD'2019, AIS.

Silva, A.R. (2021). Linguistic Patterns, Styles and Guidelines for Writing Requirements Specifications: Focus on Use Cases and Scenarios. In IEEE Access, vol. 9, pp. 143506-143530.

Silva, A.R., Paiva, A.C.R., Silva. V. (2018). A Test Specification Language for Information Systems Based on Data Entities, Use Cases and State Machines. In Proc. MODELSWARD (Revised Selected Papers).

Silva, A.R., Savić, D. (2021). Linguistic Patterns and Linguistic Styles for Requirements Specification: Focus on Data Entities. Applied Sciences 11, no. 9.

Verelst, J., Silva, A.R., Mannaert, H., Ferreira, D.A., Huysmans, P. (2013). Identifying Combinatorial Effects in Requirements Engineering. In Proc. EEWC' 2013, Springer.