Requirements Eng (2016) 21:383–403
DOI 10.1007/s00766-016-0250-x

RE 2015

Improving agile requirements: the Quality User Story framework and tool

Garm Lucassen[1] · Fabiano Dalpiaz[1] · Jan Martijn E. M. van der Werf[1] · Sjaak Brinkkemper[1]

Received: 14 November 2015 / Accepted: 14 March 2016 / Published online: 1 April 2016

Abstract User stories are a widely adopted requirements notation in agile development. Yet, user stories are too often poorly written in practice and exhibit inherent quality defects. Triggered by this observation, we propose the Quality User Story (QUS) framework, a set of 13 quality criteria that user story writers should strive to conform to. Based on QUS, we present the Automatic Quality User Story Artisan (AQUSA) software tool. Relying on natural language processing (NLP) techniques, AQUSA detects quality defects and suggest possible remedies. We describe the architecture of AQUSA, its implementation, and we report on an evaluation that analyzes 1023 user stories obtained from 18 software companies. Our tool does not yet reach the ambitious 100 % recall that Daniel Berry and colleagues require NLP tools for RE to achieve. However, we obtain promising results and we identify some improvements that will substantially improve recall and precision.

Keywords User stories · Requirements quality · AQUSA · QUS framework · Natural language processing · Multi-case study

✉ Garm Lucassen
g.lucassen@uu.nl

Fabiano Dalpiaz
f.dalpiaz@uu.nl

Jan Martijn E. M. van der Werf
j.m.e.m.vanderwerf@uu.nl

Sjaak Brinkkemper
s.brinkkemper@uu.nl

[1] Department of Information and Computing Science, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands

1 Introduction

User stories are a concise notation for expressing requirements that is increasingly employed in agile requirements engineering [7] and in agile development. Indeed, they have become the most commonly used requirements notation in agile projects [29, 53], and their adoption has been fostered by their use in numerous books about agile development [2, 3, 10, 26]. Despite some differences, all authors acknowledge the same three basic components of a user story: (1) a short piece of text describing and representing the user story, (2) conversations between stakeholders to exchange perspectives on the user story, and (3) acceptance criteria.

The short piece of text representing the user story captures only the essential elements of a requirement: *who* it is for, *what* is expected from the system, and, optionally, *why* it is important. The most widespread format and de facto standard [36], popularized by Cohn [10] is: "As a ⟨type of user⟩, I want ⟨goal⟩, [so that ⟨some reason⟩]." For example: "As an Administrator, I want to receive an email when a contact form is submitted, so that I can respond to it."

Despite this popularity, the number of methods to assess and improve user story *quality* is limited. Existing approaches either employ highly qualitative metrics, such as the six mnemonic heuristics of the INVEST (Independent–Negotiable–Valuable–Estimatable–Scalable–Testable) framework [52], or generic guidelines for quality in agile RE [21]. We made a step forward by presenting the Quality User Story (QUS) framework (originally proposed in [35]), a collection of 13 criteria that determine the quality of user stories in terms of syntax, pragmatics, and semantics.

We build on the QUS framework and present a comprehensive, tool-supported approach to assessing and enhancing user story quality. To achieve this goal, we take advantage of the potential offered by natural language processing (NLP) techniques. However, we take into account the suggestions of Berry et al. [4] on the criticality of achieving 100 % recall of quality defects, sacrificing precision if necessary. We call this the *Perfect Recall Condition*. If the analyst is assured that the tool has not missed any defects, (s)he is no longer required to manually recheck the quality of *all* the requirements.

Existing state-of-the-art NLP tools for RE such as QuARS [6], Dowser [44], Poirot [9], and RAI [18] take the orthogonal approach of maximizing their accuracy. The ambitious objectives of these tools demand a deep understanding of the requirements' contents [4]. However, this is still practically unachievable unless a radical breakthrough in NLP occurs [47]. Nevertheless, these tools serve as an inspiration and some of their components are employed in our work.

Our previous paper [35] proposed the QUS framework for improving user story quality and introduced the concept of the Automated Quality User Story Artisan (AQUSA) tool. In this paper, we make three new, main contributions to the literature:

- We revise the QUS framework based on the lessons learned from its application to different case studies. QUS consists of 13 criteria that determine the quality of user stories in terms of syntax, semantics, and pragmatics.
- We describe the architecture and implementation of the AQUSA software tool, which uses NLP techniques to detect quality defects. We present AQUSA version 1 that focuses on syntax and pragmatics.
- We report on a large-scale evaluation of AQUSA on 1023 user stories, obtained from 18 different organizations. The primary goals are to determine AQUSA's capability of fulfilling the Perfect Recall Condition with high-enough precision, but also acts as a *formative evaluation* for us to improve AQUSA.

The remainder of this paper is structured as follows. In Sect. 2, we present the conceptual model of user stories that serves as the baseline for our work. In Sect. 3, we detail the QUS framework for assessing the quality of user stories. In Sect. 4, we describe the architecture of AQUSA and the implementation of its first version. In Sect. 5, we report on the evaluation of AQUSA on 18 case studies. In Sect. 6, we build on the lessons learned from the evaluation and propose improvements for AQUSA. Section 7 reviews related work. Section 8 presents conclusions and future work.

2 A conceptual model of user stories

There are over 80 syntactic variants of user stories [54]. Although originally proposed as unstructured text similar to use cases [2] but restricted in size [10], nowadays user stories follow a strict, compact template that captures *who* it is for, *what* it expects from the system, and (optionally) *why* it is important [54].

When used in Scrum, two other artifacts are relevant: *epics* and *themes*. An epic is a large user story that is broken down into smaller, implementable user stories. A theme is a set of user stories grouped according to a given criterion such as *analytics* or *user authorization* [10]. For simplicity, and due to their greater popularity, we include only epics in our conceptual model.

Our conceptual model for user stories is shown in Fig. 1 as a class diagram. A user story itself consists of four parts: one role, one means, zero or more ends, and a format. In the following subsections, we elaborate on how to decompose each of these. Note that we deviate from Cohn's terminology as presented in the introduction, using the well known means end [48] relationship instead of the ad hoc goal reason. Additionally, observe that this conceptual model includes only aggregation relationships. Arguably a composition relationship is more appropriate for a single user story. When a composite user story is destroyed, so are its role, means, and end(s) parts. However, each separate part might continue to exist in another user story in a set of user stories. Because of this difficulty in conceptualizing, we choose to use aggregation relationships because it implies a weaker ontological commitment.

2.1 Format

A user story should follow some pre-defined, agreed upon template chosen from the many existing ones [54]. The skeleton of the template is called *format* in the conceptual model, in between which the role, means, and optional end(s) are interspersed to form a user story. See the introduction for a concrete example.

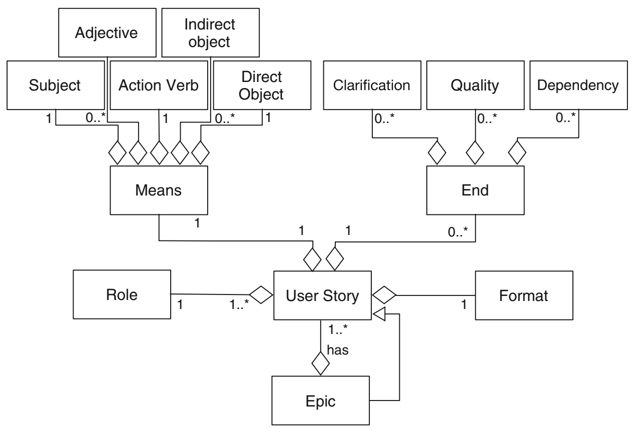


Fig. 1 Conceptual model of user stories

2.2 Role

A user story always includes one relevant role, defining what stakeholder or persona expresses the need. Typically, roles are taken from the software's application domain. Example stakeholders from the ERP domain are account manager, purchaser, and sales representative. An alternative approach is to use personas, which are named, fictional characters that represent an archetypal group of users [11]. Although imaginary, personas are defined with rigor and precision to clearly capture their goals. Examples are Joe the carpenter, Alice the mother, and Seth the young executive, who all have different goals, preferences and constraints. When used in a user story, the name of the persona acts as the role: "As a Joe" or "As an Alice."

2.3 Means

Means can have different structures, for they can be used to represent different types of requirements. From a grammatical standpoint, we support means that have three common elements:

1. a *subject* with an aim such as "want" or "am able,"
2. an *action verb*[1] that expresses the action related to the feature being requested, and
3. a *direct object* on which the subject executes the action.

For example: "I want to open the interactive map." Aside from this basic requirement, means are essentially free form text which allow for an unbounded number of constructions. Two common additions are an adjective or an indirect object, which is exemplified as follows: "I want to open a larger (adjective) view of the interactive map from the person's profile page (indirect object)." We included these interesting cases in the conceptual model, but left out all other variations, which we are currently examining in a different research project.

2.4 End

One or more end parts explain why the means [10] are requested. However, user stories often also include other types of information. Our analysis of the ends available in the data sets in our previous work [35] reveals at least three possible variants of a well-formed end:

1. *Clarification of means* The end explains the reason of the means. Example: "As a User, I want to edit a record, so that I can correct any mistakes."
2. *Dependency on another functionality* The end (implicitly) references a functionality which is required for the means to be realized. Although dependency is an indicator of a bad quality criteria, having no dependency at all between requirements is practically impossible [52]. There is no size limit to this dependency on the (hidden) functionality. Small example: "As a Visitor, I want to view the homepage, so that I can learn about the project." The end implies the homepage also has relevant content, which requires extra input. Larger example: "As a User, I want to open the interactive map, so that I can see the location of landmarks." The end implies the existence of a landmark database, a significant additional functionality to the core requirement.
3. *Quality requirement* The end communicates the intended qualitative effect of the means. For example: "As a User, I want to sort the results, so that I can more easily review the results" indicates that the means contributes maximizing easiness.

Note that these three types of end are not mutually exclusive, but can occur simultaneously such as in "As a User, I want to open the landmark, so that I can more easily view the landmark's location." The means only specifies that the user wishes to view a landmark's page. The end, however, contains elements of all three types: (1) a clarification the user wants to open the landmark to view its location, (2) implicit dependency on landmark functionality, and (3) the quality requirement that it should be easier than other alternatives.

3 User story quality

The IEEE Recommended Practice for Software Requirements Specifications defines requirements quality on the basis of eight characteristics [24]: correct, unambiguous, complete, consistent, ranked for importance/stability, verifiable, modifiable, and traceable. The standard, however, is generic and it is well known that specifications are hardly able to meet those criteria [19]. With agile requirements in mind, the Agile Requirements Verification Framework [21] defines three high-level verification criteria: completeness, uniformity, and consistency and correctness. The framework proposes specific criteria to be able to apply the quality framework to both feature requests and user stories. Many of these criteria, however, require supplementary, unstructured information that is not captured in the primary user story text.

With this in mind, we introduce the QUS framework (Fig. 2; Table 1). The QUS Framework focuses on the intrinsic quality of the user story text. Other approaches

[1] While other types of verbs are in principle admitted, in this paper we focus on action verbs, which are the most used in user stories requesting features.

complement QUS by focusing on different notions of quality in RE quality such as performance with user stories [33] or broader requirements management concerns such as effort estimation and additional information sources such as descriptions or comments [21]. Because user stories are a controlled language, the QUS framework's criteria are organized in Lindland's categories [31]:

Syntactic quality, concerning the textual structure of a user story without considering its meaning;
Semantic quality, concerning the relations and meaning of (parts of) the user story text;
Pragmatic quality, considers the audience's subjective interpretation of the user story text aside from syntax and semantics.

The last column of Table 1 classifies the criteria depending on whether they relate to an individual user story or to a set of user stories.

In the next subsections, we introduce each criterion by presenting an explanation of the criterion as well as an example user story that violates the specific criterion. We employ examples taken from two real-world user story databases of software companies in the Netherlands. One contains 98 stories concerning a tailor-made Web information system. The other consists of 26 user stories from an advanced healthcare software product for home care professionals. These databases are intentionally excluded from the evaluation of Sect. 5, for we used them extensively during the development of our framework and tool.

3.1 Quality of an individual user story

We first describe the quality criteria that can be evaluated against an individual user story.

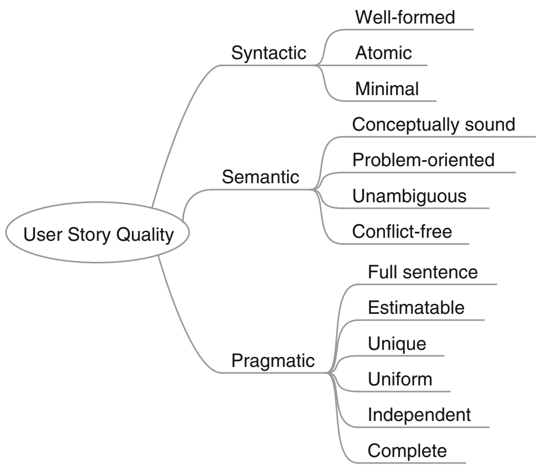


Fig. 2 Quality User Story framework that defines 13 criteria for user story quality: overview

3.1.1 Well-formed

Before it can be considered a user story, the core text of the requirement needs to include a role and the expected functionality: the *means*. US_1 does not adhere to this syntax, as it has no role. It is likely that the user story writer has forgotten to include the role. The story can be fixed by adding the role: "As a Member, I want to see an error when I cannot see recommendations after I upload an article."

3.1.2 Atomic

A user story should concern only one feature. Although common in practice, merging multiple user stories into a larger, generic one diminishes the accuracy of effort estimation [32]. The user story US_2 in Table 2 consists of two separate requirements: the act of clicking on a location and the display of associated landmarks. This user story should be split into two:

- US_{2A}: "As a User, I'm able to click a particular location from the map";
- US_{2B}: "As a User, I'm able to see landmarks associated with the latitude and longitude combination of a particular location."

3.1.3 Minimal

User stories should contain a role, a means, and (optimally) some ends. Any additional information such as comments, descriptions of the expected behavior, or testing hints should be left to additional notes. Consider US_3: Aside from a role and means, it includes a reference to an undefined mockup and a note on how to approach the implementation. The requirements engineer should move both to separate user story attributes like the description or comments, and retain only the basic text of the story: "As a care professional, I want to see the registered hours of this week."

3.1.4 Conceptually sound

The means and end parts of a user story play a specific role. The means should capture a concrete feature, while the end expresses the rationale for that feature. Consider US_4: The end is actually a dependency on another (hidden) functionality, which is required in order for the means to be realized, implying the existence of a landmark database which is not mentioned in any of the other stories. A significant additional feature that is erroneously represented as an end, but should be a means in a separate user story, for example:

Table 1 Quality User Story framework that defines 13 criteria for user story quality: details

Criteria	Description	Individual/set
Syntactic		
Well-formed	A user story includes at least a role and a means	Individual
Atomic	A user story expresses a requirement for exactly one feature	Individual
Minimal	A user story contains nothing more than role, means, and ends	Individual
Semantic		
Conceptually sound	The means expresses a feature and the ends expresses a rationale	Individual
Problem-oriented	A user story only specifies the problem, not the solution to it	Individual
Unambiguous	A user story avoids terms or abstractions that lead to multiple interpretations	Individual
Conflict-free	A user story should not be inconsistent with any other user story	Set
Pragmatic		
Full sentence	A user story is a well-formed full sentence	Individual
Estimatable	A story does not denote a coarse-grained requirement that is difficult to plan and prioritize	Individual
Unique	Every user story is unique, duplicates are avoided	Set
Uniform	All user stories in a specification employ the same template	Set
Independent	The user story is self-contained and has no inherent dependencies on other stories	Set
Complete	Implementing a set of user stories creates a feature-complete application, no steps are missing	Set

Table 2 Sample user stories that breach quality criteria from two real-world cases

ID	Description	Violated qualities
US_1	I want to see an error when I cannot see recommendations after I upload an article	*Well-formed* the role is missing
US_2	As a User, I am able to click a particular location from the map and thereby perform a search of landmarks associated with that latitude longitude combination	*Atomic* two stories in one
US_3	As a care professional, I want to see the registered hours of this week (split into products and activities). See: Mockup from Alice NOTE—first create the overview screen—then add validations	*Minimal* there is an additional note about the mockup
US_4	As a User, I want to open the interactive map, so that I can see the location of landmarks	*Conceptually sound* the end is a reference to another story
US_5	As a care professional I want to save a reimbursement—add save button on top right (never grayed out)	*Problem-oriented* Hints at the solution
US_6	As a User, I am able to edit the content that I added to a person's profile page	*Unambiguous* what is content?
US_7	As a User, I am able to edit any landmark	*Conflict-free* US_7 refers to any landmark, while US_8 only to those that user has added
US_8	As a User, I am able to delete only the landmarks that I added	
US_9	Server configuration	*Well-formed, full sentence*
US_{10}	As a care professional I want to see my route list for next/future days, so that I can prepare myself (for example I can see at what time I should start traveling)	*Estimatable* it is unclear what see my route list implies
EP_A	As a Visitor, I am able to see a list of news items, so that I stay up to date	*Unique* the same requirement is both in epic EP_A and in story
US_{11}	As a Visitor, I am able to see a list of news items, so that I stay up to date	US_{11}
US_{12}	As an Administrator, I receive an email notification when a new user is registered	*Uniform* deviates from the template, no "wish" in the means
US_{13}	As an Administrator, I am able to add a new person to the database	*Independent* viewing relies on first adding a person to the database
US_{14}	As a Visitor, I am able to view a person's profile	

- US$_{4A}$: “As a User, I want to open the interactive map”;
- US$_{4B}$: “As a User, I want to see the location of landmarks on the interactive map.”

3.1.5 Problem-oriented

In line with the problem specification principle for RE proposed by Zave and Jackson [57], a user story should specify only the problem. If absolutely necessary, implementation hints can be included as comments or descriptions. Aside from breaking the minimal quality criteria, US$_5$ includes implementation details (a solution) within the user story text. The story could be rewritten as follows: “As a care professional, I want to save a reimbursement.”

3.1.6 Unambiguous

Ambiguity is intrinsic to natural language requirements, but the requirements engineer writing user stories has to avoid it to the extent this is possible. Not only should a user story be internally unambiguous, but it should also be clear in relationship to all other user stories. The Taxonomy of Ambiguity Types [5] is a comprehensive overview of the kinds of ambiguity that can be encountered in a systematic requirements specification. In US$_6$, “content” is a superclass referring to audio, video, and textual media uploaded to the profile page as specified in three other, separate user stories in the real-world user story set. The requirements engineer should explicitly mention which media are editable; for example, the story can be modified as follows: “As a User, I am able to edit video, photo and audio content that I added to a person’s profile page.”

3.1.7 Full sentence

A user story should read like a full sentence, without typos or grammatical errors. For instance, US$_9$ is not expressed as a full sentence (in addition to not complying with syntactic quality). By reformulating the feature as a full sentence user story, it will automatically specify what exactly needs to be configured. For example, US$_9$ can be modified to “As an Administrator, I want to configure the server’s sudo-ers.”

3.1.8 Estimatable

As user stories grow in size and complexity, it becomes more difficult to accurately estimate the required effort. Therefore, each user story should not become so large that estimating and planning it with reasonable certainty becomes impossible [52]. For example, US$_{10}$ requests a route list so that care professionals can prepare themselves. While this might be just an unordered list of places to go to during a workday, it is just as likely that the feature includes ordering the routes algorithmically to minimize distance travelled and/or showing the route on a map. These many functionalities inhibit accurate estimation and call for splitting the user story into multiple user stories; for example,

- US$_{10A}$: “As a Care Professional, I want to see my route list for next/future days, so that I can prepare myself”;
- US$_{10B}$: “As a Manager, I want to upload a route list for care professionals.”

3.2 Quality of a set of user stories

We focus now on the quality of a set of user stories; these quality criteria help verify the quality of a complete project specification, rather than analyzing an individual story. To make our explanation more precise, we associate every criterion with first-order logic predicates that enable verifying if the criterion is violated.

Notation Lower-case identifiers refer to single elements (e.g., one user story), and upper-case identifiers denote sets (e.g., a set of user stories). A user story μ is a 4-tuple $\mu = \langle r, m, E, f \rangle$ where r is the role, m is the means, $E = \{e_1, e_2, \ldots\}$ is a set of ends, and f is the format. A means m is a 5-tuple $m = \langle s, av, do, io, adj \rangle$ where s is a subject, av is an action verb, do is a direct object, io is an indirect object, and adj is an adjective (io and adj may be null, see Fig. 1). The set of user stories in a project is denoted by $U = \{\mu_1, \mu_2, \ldots\}$.

Furthermore, we assume that the equality, intersection, etc. operators are semantic and look at the meaning of an entity (e.g., they account for synonyms). To denote that a syntactic operator, we add the subscript “syn”; for instance, $=_{syn}$ is syntactic equivalence. The function $depends(av, av')$ denotes that executing the action av on an object requires first executing av' on that very object (e.g., “delete” depends on “create”).

In the following subsections, let $\mu_1 = \langle r_1, m_1, E_1, f_1 \rangle$ and $\mu_2 = \langle r_2, m_2, E_2, f_2 \rangle$ be two user stories from the set U, where $m_1 = \langle s_1, av_1, do_1, io_1, adj_1 \rangle$ and $m_2 = \langle s_2, av_2, do_2, io_2, adj_2 \rangle$.

3.2.1 Unique and conflict-free

We present these two criteria together because they rely on the same set of predicates that can be used to check whether quality defects exist.

A user story is unique when no other user story in the same project is (semantically) equal or too similar. We focus on similarity that is a potential indicator of duplicate user stories; see, for example, US$_{11}$ and epic EP$_A$ in

Table 2. This situation can be improved by providing more specific stories, for example:

- US_{11A} As a Visitor, I am able to see breaking news;
- US_{11B} As a Visitor, I am able to see sports news.

Additionally, a user story should not conflict with any of the other user stories in the database. A requirements conflict occurs when two or more requirements cause an inconsistency [42, 46]. Story US_7 contradicts the requirement that a user can edit any landmark (US_8), if we assume that editing is a general term that includes deletion too. A possible way to fix this is to change US_7 to: "As a User, I am able to edit the landmarks that I added."

To detect these types of relationships, each user story part needs to be compared with the parts of other user stories, using a combination of similarity measures that are either syntactic (e.g., Levenshtein's distance) or semantic (e.g., employing an ontology to determine synonyms). When similarity exceeds a certain threshold, a human analyst is required to examine the user stories for potential conflict and/or duplication.

Full duplicate A user story μ_1 is an exact duplicate of another user story μ_2 when the stories are identical. This impacts the *unique* quality criterion. Formally,

$$isFullDuplicate(\mu_1, \mu_2) \leftrightarrow \mu_1 =_{syn} \mu_2$$

Semantic duplicate A user story μ_1 that duplicates the request of μ_2, while using a different text; this has an impact on the *unique* quality criterion. Formally,

$$isSemDuplicate(\mu_1, \mu_2) \leftrightarrow \mu_1 = \mu_2 \wedge \mu_1 \neq_{syn} \mu_2$$

Different means, same end Two or more user stories that have the same end, but achieve this using different means. This relationship potentially impacts two quality criteria, as it may indicate: (1) a feature variation that should be explicitly noted in the user story to maintain an *unambiguous* set of user stories, or (2) a conflict in how to achieve this end, meaning one of the user stories should be dropped to ensure *conflict-free* user stories. Formally, for user stories μ_1 and μ_2:

$$diffMeansSameEnd(\mu_1, \mu_2) \leftrightarrow m_1 \neq m_2 \wedge E_1 \cap E_2 \neq \emptyset$$

Same means, different end Two or more user stories that use the same means to reach different ends. This relationship could affect the qualities of user stories to be *unique* or *independent* of each other. If the ends are not conflicting, they could be combined into a single larger user story; otherwise, they are multiple viewpoints that should be resolved. Formally,

$$sameMeansDiffEnd(\mu_1, \mu_2) \leftrightarrow m_1 = m_2 \wedge (E_1 \setminus E_2 \neq \emptyset \vee E_2 \setminus E_1 \neq \emptyset)$$

Different role, same means and/or same end Two or more user stories with different roles, but same means and/or ends indicates a strong relationship. Although this relationship has an impact on the *unique* and *independent* quality criteria, it is considered good practice to have separate user stories for the same functionality for different roles. As such, requirements engineers could choose to ignore this impact. Formally,

$$diffRoleSameStory(\mu_1, \mu_2) \leftrightarrow r_1 \neq r_2 \wedge (m_1 = m_2 \vee E_1 \cap E_2 \neq \emptyset)$$

End = means The end of one user story μ_1 appears as the means of another user story μ_2, thereby expressing both a wish and a reason for another wish. When there is this strong a semantic relationship between two user stories, it is important to add *explicit dependencies* to the user stories, although this breaks the *independent* criterion. Formally, $purposeIsMeans(\mu_1, \mu_2)$ is true if the means m_2 of μ_2 is an end in μ_1:

$$purposeIsMeans(\mu_1, \mu_2) \leftrightarrow E_1 = \{m_2\}$$

3.2.2 Uniform

Uniformity in the context of user stories means that a user story has a format that is consistent with that of the majority of user stories in the same set. To test this, the requirements engineer needs to determine the most frequently occurring format, typically agreed upon with the team. The format f_1 of an individual user story μ_1 is syntactically compared to the most common format f_{std} to determine whether it adheres with the *uniformity* quality criterion. US_{12} in Table 2 is an example of a nonuniform user story, which can be rewritten as follows: "As an Administrator, I want to receive an email notification when a new user is registered." Formally, predicate $isNotUniform(\mu_1, f_{std})$ is true if the format of μ_1 deviates from the standard:

$$isNotUniform(\mu_1, f_{std}) \leftrightarrow f_1 \neq_{syn} f_{std}$$

3.2.3 Independent

User stories should not overlap in concept and should be schedulable and implementable in any order [52]. For example, US_{14} is dependent on US_{13}, because it is impossible to view a person's profile without first laying the foundation for creating a person. Much like in programming loosely coupled systems, however, it is practically impossible to never breach this quality criterion; our recommendation is then to make the relationship visible through the establishment of an explicit dependency. How to make a dependency explicit is outside of the scope of the QUS Framework. Note that the dependency in US_{13} and US_{14} is one that cannot be resolved. Instead, the requirements engineer could add a note to the backside of their story cards or a hyperlink to their description fields in the issue tracker. Among the many different types of dependency, we present two illustrative cases.

Causality In some cases, it is necessary that one user story μ_1 is completed before the developer can start on another user story μ_2 (US_{13} and US_{14} in Table 2). Formally, the predicate $hasDep(\mu_1, \mu_2)$ holds when μ_1 causally depends on μ_2:

$$hasDep(\mu_1, \mu_2) \leftrightarrow depends(av_1, av_2) \ \wedge \ do_1 = do_2$$

Superclasses An object of one user story μ_1 can refer to multiple other objects of stories in U, indicating that the object of μ_1 is a parent or *superclass* of the other objects. "Content" for example can refer to different types of multimedia and be a superclass, as exemplified in US_6. Formally, predicate $hasIsaDep(\mu_1, \mu_2)$ is true when μ_1 has a direct object superclass dependency based on the subclass do_2 of do_1.

$$hasIsaDep(\mu_1, \mu_2) \leftrightarrow \exists \mu_2 \in \ U. \ is\text{-}a(do_2, do_1)$$

3.2.4 Complete

Implementing a set of user stories should lead to a feature-complete application. While user stories should not thrive to cover 100 % of the application's functionality preemptively, crucial user stories should not be missed, for this may cause a show stopping feature gap. An example: US_6 requires the existence of another story that talks of the creation of content. This scenario can be generalized to the case of user stories with action verbs that refer to a non-existent direct object: to read, update or delete an item one first needs to create it. We define a conceptual relationship that focuses on dependencies concerning the means' direct object. Note that we do not claim nor believe these relationships to be the only relevant one to ensure completeness. Formally, the predicate $voidDep(\mu_1)$ holds when there is no story μ_2 that satisfies a dependency for μ_1's direct object:

$$voidDep(\mu_1) \leftrightarrow depends(av_1, av_2) \wedge \ \nexists \mu_2 \in U. \ do_2 = do_1$$

4 The Automatic Quality User Story Artisan tool

The QUS framework provides guidelines for improving the quality of user stories. To support the framework, we propose the AQUSA tool, which exposes defects and deviations from good user story practice.

In line with Berry et al.'s [4] notion of a *dumb tool*, we require AQUSA to detect defects with close to 100 % *recall*[2], which is the number of true positives in proportion to the total number of relevant defects. We call this the *Perfect Recall Condition*. When this condition is not fulfilled, the requirements engineer needs to manually check the entire set of user stories for missed defects [51], which we want to avoid. On the other hand, *precision*, the number of false positives in proportion to the detected defects, should be high enough so the user perceives AQUSA to report useful errors.

AQUSA is designed as a tool that focuses on easily describable, algorithmically determinable defects: the *clerical* part of RE [51]. This also implies that the first version of AQUSA focuses on the QUS criteria for which the probability of fulfilling the Perfect Recall Condition is high; thus, we include the syntactic criteria and a few pragmatic criteria that can be algorithmically checked, but we exclude semantic criteria as they require deep understanding of requirements' content [47].

Next, we present AQUSA's architecture and discuss the selected quality criteria including their theoretical and technical implementation in AQUSA v1 as well as example input and output user stories.

4.1 Architecture and technology

AQUSA is designed as a simple, stand-alone, deployable as a service application that analyzes a set of user stories regardless of its source of origin. AQUSA exposes an API for importing user stories, meaning that AQUSA can easily integrate with any requirements management tool such as Jira, Pivotal Tracker or even MS Excel spreadsheets by developing adequate connectors. By retaining its independence from other tools, AQUSA is capable of easily adapting to future technology changes. Aside from importing user stories, AQUSA consists of five main architectural components (Fig. 3): linguistic parser, user story base, analyzer, enhancer, and report generator.

The first step for every user story is validating that it is well-formed. This takes place in the linguistic parser, which separates the user story in its role, means and end(s) parts. The user story base captures the parsed user story as an object according to the conceptual model, which acts as central storage. Next, the analyzer runs tailor-made method to verify specific syntactic and pragmatic quality criteria—where possible enhancers enrich the user story base, improving the recall and precision of the

[2] Unless mathematically proved, 100 % recall is valid until a counterexample is identified. Thus, we decide to relax the objective to "close to 100 % recall".

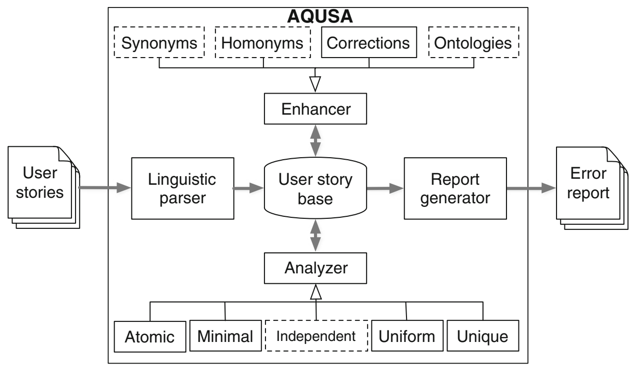


Fig. 3 Functional view on the architecture of AQUSA. *Dashed components* are not fully implemented yet

analyzers. Finally, AQUSA captures the results in a comprehensive report.

The development view of AQUSA v1 is shown in the component diagram of Fig. 4. Here we see that AQUSA v1 is built around the model–view–controller design pattern. When an outside requirements management tool sends a request to one of the interfaces, the relevant controller parses the request to figure out what method(s) to call from the Project Model or Story Model. When this is a story analysis, AQUSA v1 runs one or more story analyses by first calling the StoryChunker and then running the Unique-, Minimal-, WellFormed-, Uniform-, and AtomicAnalyzer. Whenever one of these encounters a quality criteria violation, it calls the DefectGenerator to record a defect in the database tables associated to the story. Optionally, the end user can call the AQUSA-GUI to view a listing of all his projects or a report of all the defects associated with a set of stories.

AQUSA v1 is built on the Flask microframework for Python. It relies on specific parts of both Stanford CoreNLP[3] and the Natural Language ToolKit[4] (NLTK) for the StoryChunker and AtomicAnalyzer. The majority of the functionality, however, is captured in tailor-made methods whose implementation is detailed in the next subsections.

4.2 Linguistic parser: well-formed

One of the essential aspects of verifying whether a string of text is a user story is splitting it into role, means, and end(s). This first step takes place in the linguistic parser (see the functional view) that is implemented by the component *StoryChunker*. First, it detects whether a known, common indicator text for role, means, and ends is present in the user story such as "As a," "I want to," "I am able to," and "so that." If successful, AQUSA categorizes the words in each chunk by using the Stanford NLP POS Tagger.[5] For each chunk, the linguistic parser validates the following rules:

- Role: Is the last word a noun depicting an actor? Do the words before the noun match a known role format, e.g., "as a"?
- Means: Is the first word "I"? Can we identify a known means format such as "want to"? Does the remaining text include at least a second verb and one noun such as "update event"?
- End: Is an end present? Does it start with a known end format such as "so that"?

Basically, the linguistic parser validates whether a user story complies with the conceptual model presented in Sect. 2. When the linguistic parser is unable to detect a known means format, it takes the full user story and strips away any role and ends parts. If the remaining text contains both a verb and a noun it is tagged as a "potential means" and all the other analyzers are run. Additionally, the linguistic parser checks whether the user story contains a comma after the role section. A pseudocode implementation is shown in Algorithm 1. Note that the Chunk method tries to detect the role, means, and ends by searching for the provided XXX_FORMATS. When detecting a means fails, it tests whether a potential means is available.

If the linguistic parser encounters a piece of text that is not a valid user story such as "Test 1," it reports that it is not well-formed because it does not contain a role and the remaining text does not include a verb and a noun. The story "Add static pages controller to application and define static pages" is not well-formed because it does not explicitly contain a role. The well-formed user story "As a Visitor, I want to register at the site, so that I can contribute," however, is verified and separated into the following chunks:

Role: As a Visitor
Means: I want to register at the site
End: so that I can contribute

Algorithm 1 Linguistic Parser

```
1: procedure StoryChunker
2:     role = Chunk(raw_text, role, ROLE_FORMATS)
3:     means = Chunk(raw_text, means, MEANS_FORMATS)
4:     ends = Chunk(raw_text, ends, ENDS_FORMATS)
5:     if means==null then
6:         potential_means= raw_text - [role, ends]
7:         if Tag(potential_means).include?('verb' and 'noun')
8:         then means = potential_means
```

[3] http://nlp.stanford.edu/software/corenlp.shtml.

[4] http://www.nltk.org/.

[5] http://nlp.stanford.edu/software/tagger.shtml.

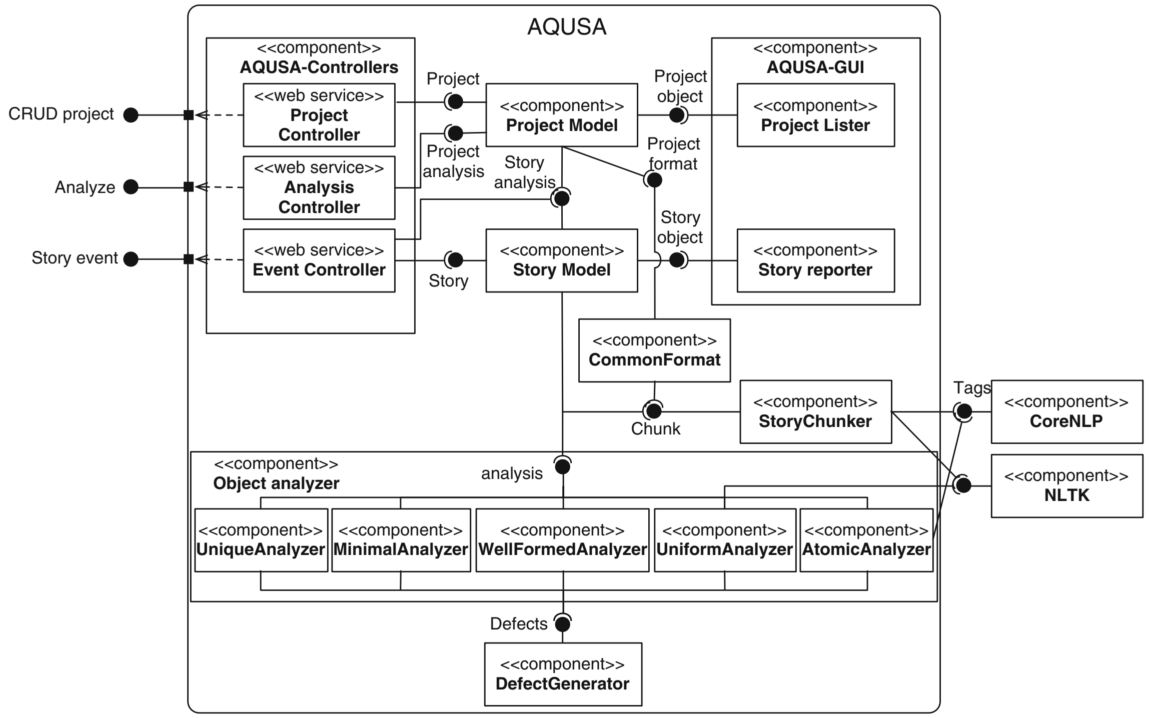

AQUSA
<<component>> AQUSA-Controllers
<<web service>> Project Controller
<<web service>> Analysis Controller
<<web service>> Event Controller
CRUD project
Analyze
Story event
Project
Project analysis
Story analysis
Story
<<component>> Project Model
<<component>> Story Model
Project object
Project format
Story object
<<component>> AQUSA-GUI
<<component>> Project Lister
<<component>> Story reporter
<<component>> CommonFormat
<<component>> StoryChunker
Chunk
Tags
<<component>> CoreNLP
<<component>> NLTK
<<component>> Object analyzer
analysis
<<component>> UniqueAnalyzer
<<component>> MinimalAnalyzer
<<component>> WellFormedAnalyzer
<<component>> UniformAnalyzer
<<component>> AtomicAnalyzer
Defects
<<component>> DefectGenerator

Fig. 4 Development view on the architecture of AQUSA

4.3 User story base and enhancer

A linguistically parsed user story is stored as an object with a role, means, and ends part—aligned with the first decomposition level in the conceptual model in Fig. 1—in the user story base, ready to be further processed. But first, AQUSA enhances user stories by adding possible synonyms, homonyms, and relevant semantic information—extracted from an ontology—to the relevant words in each chunk. Furthermore, the enhancer has a subpart *corrections* which automatically fixes any defects that it is able to correct with 100 % precision. For now, this is limited to the good practice of injecting comma's after the role section. AQUSA v1 does not include the other enhancer's subparts.

4.4 Analyzer: atomic

To audit that the means of the user story concerns only one feature, AQUSA parses the means for occurrences of the conjunctions "and, &, +, or" in order to include any double feature requests in its report. Additionally, AQUSA suggests the reader to split the user story into multiple user stories. The user story "As a User, I'm able to click a particular location from the map and thereby perform a search of landmarks associated with that latitude longitude combination" would generate a suggestion to be split into two user stories: (1) "As a User, I want to click a location from the map" and (2) "As a User, I want to search landmarks associated with the lat long combination of a location."

AQUSA v1 checks for the role and means chunks whether the text contains one of the conjunctions "and, &, +, or." When this is the case, it triggers the linguistic parser to validate that the text on both sides of the conjunction has the building blocks of a valid role or means as defined in Sect. 4.2. Only when this is the case, AQUSA v1 records the text after the conjunction as an atomicity violation.

4.5 Analyzer: minimal

To test this quality criterion, AQUSA relies on the results of chunking and verification of the *well-formedness* quality criterion to extract the role and means. When this process has been successfully completed, AQUSA reports any user story that contains additional text after a dot, hyphen, semicolon, or other separating punctuation marks. In "As a care professional I want to see the registered hours of this week (split into products and activities). See: Mockup from Alice NOTE: First create the overview screen—Then add validations" all the text after the first dot (".") AQUSA reports as not minimal. AQUSA also records the text between parentheses as not minimal.

AQUSA v1 runs two separate minimality checks on the entire user story using regular expressions in no particular order. The first searches for occurrences of special punctuation such as "-, ?, ., *." Any text that comes afterward is recorded as a minimality violation. The second minimality check searches for text that is in between brackets such as “(), [], {}, ⟨⟩” to record as a minimality violation.

4.6 Analyzer: explicit dependencies

Whenever a user story includes an explicit dependency on another user story, it should include a navigable link to the dependency. Because the popular issue trackers Jira and Pivotal Tracker use numbers for dependencies, AQUSA checks for numbers in user stories and checks whether the number is contained within a link. The example “As a care professional, I want to edit the planned task I selected—see 908” would prompt the user to change the isolated number to “See PID-908,” where PID stands for the project identifier. In the issue tracker, this should automatically change to “see PID-908 (http://company.issuetracker.org/PID-908).” This explicit dependency analyzer has not been implemented for AQUSA v1. Although it is straightforward to implement for a single issue tracker, we have not done this yet to ensure universal applicability of AQUSA v1.

Algorithm 2 Uniformity Analyzer

```
1: procedure GET_COMMON_FORMAT
2:     $format = [\,]$
3:     for $chunk$ in [‘role’, ‘means’, ‘ends’] do
4:         $chunks = [\,]$
5:         for $story$ in $stories$ do
6:             $chunks$ += extract_indicators($story$.chunk)
7:             $format$ += $Counter(chunks).most\_common(1)$
8:     $project.format = format$
```

4.7 Analyzer: uniform

Aside from chunking, AQUSA extracts the user story format parts out of each chunk and counts their occurrences throughout the set of user stories. The most commonly occurring format is used as the standard user story format. All other user stories are marked as non-compliant to the standard and included in the error report. For example, AQUSA reports that “As a User, I am able to delete a landmark” deviates from the standard ”I want to.“

When the linguistic parser completes its task for all the user stories within a set, AQUSA v1 first determines the most common user story format before running any other analysis. It counts the indicator phrase occurrences and saves the most common one. An overview of the underlying logic is available in Algorithm 2. Later on, the dedicated uniformity analyzer calculates the edit distance between the format of a single user story chunk and the most common format for that chunk. When this number is bigger than 3, AQUSA v1 records the entire story as violating uniformity. We have deliberately chosen 3 so that the difference between ”I am“ and ”I’m“ does not trigger a uniformity violation, while ”want“ versus ”can“ or ”need“ or ”able“ does.

4.8 Analyzer: unique

AQUSA could implement each of the similarity measures that we outlined in [35] using the WordNet lexical database [40] to detect semantic similarity. For each verb and object in a means or end, AQUSA runs a WordNet::Similarity calculation with the verbs or objects of all other means or ends. Combining the calculations results in one similarity degree for two user stories. When this metric is bigger than 90 %, AQUSA reports the user stories as potential duplicates.

AQUSA v1 implements only the most basic of uniqueness measures: exact duplication. For every single user story, AQUSA v1 checks whether an identical other story is present in the set. When this is the case, AQUSA v1 records both user stories as duplicates. The approach outlined above is part of future work, although it is unlikely to fulfill the Perfect Recall Condition unless a breakthrough in computer understanding of natural language occurs [47].

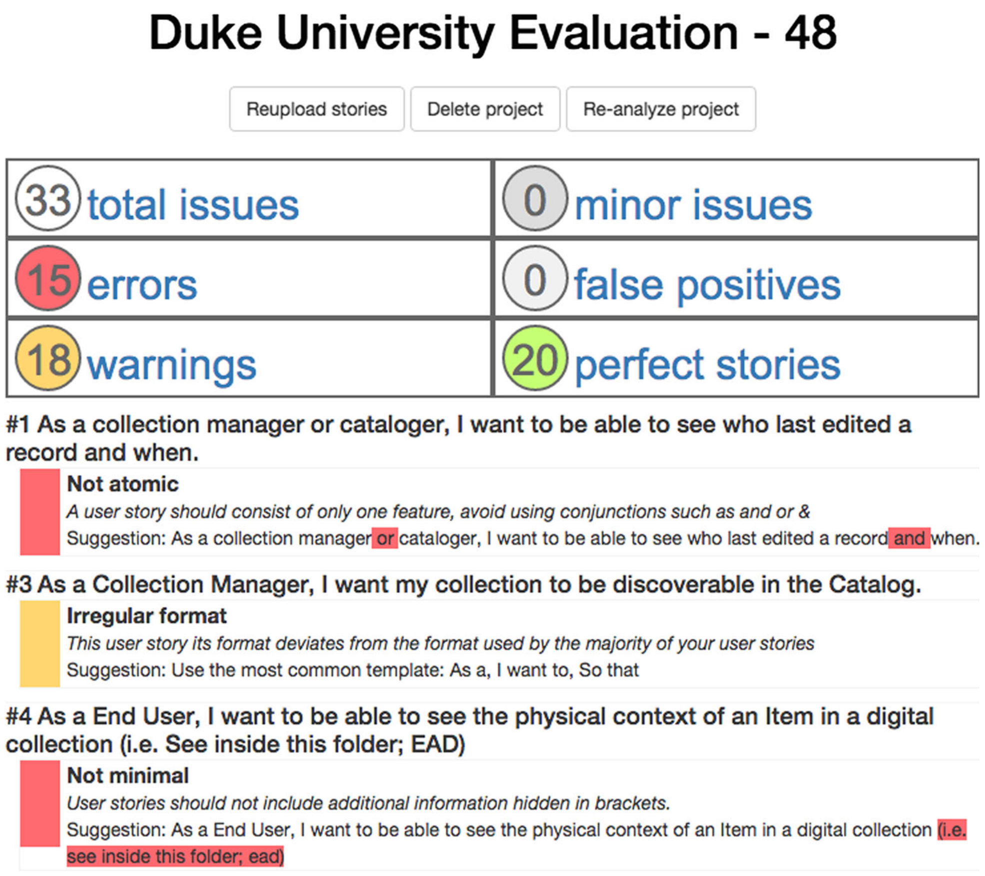
4.9 AQUSA-GUI: report generator

The AQUSA-GUI component of AQUSA v1 includes a report generation front-end that enables using AQUSA without implementing a specific connector. Whenever a violation is detected in the linguistic parser or one of the analyzers, a defect is immediately created in the database, recording the type of defect, a highlight of where the defect is within the user story and its severity. AQUSA uses this information to present a comprehensive report to the user. At the top, a dashboard is shown with a quick overview of the user story set’s quality showing the total number of issues, broken down into defects and warnings as well as the number of perfect stories. Below the dashboard, all user stories with issues are listed with their respective warnings and errors. See Fig. 5 for an example.

5 AQUSA evaluation

We present an evaluation of AQUSA v1 on 18 real-world user story sets. Our evaluation’s goals are as follows:

1. To validate to what extent the detected errors actually exist in practice;
2. To test whether AQUSA fulfills the Perfect Recall Condition;


Duke University Evaluation - 48

33 total issues	0 minor issues
15 errors	0 false positives
18 warnings	20 perfect stories

#1 As a collection manager or cataloger, I want to be able to see who last edited a record and when.

Not atomic
A user story should consist of only one feature, avoid using conjunctions such as and or &
Suggestion: As a collection manager or cataloger, I want to be able to see who last edited a record and when.

#3 As a Collection Manager, I want my collection to be discoverable in the Catalog.

Irregular format
This user story its format deviates from the format used by the majority of your user stories
Suggestion: Use the most common template: As a, I want to, So that

#4 As a End User, I want to be able to see the physical context of an Item in a digital collection (i.e. See inside this folder; EAD)

Not minimal
User stories should not include additional information hidden in brackets.
Suggestion: As a End User, I want to be able to see the physical context of an Item in a digital collection (i.e. see inside this folder; ead)

Fig. 5 Example report of a defect and warning for a story in AQUSA

3. To measure AQUSA's precision for the different quality criteria.

The 18 real-world user story sets have varying origins. Sixteen are from medium to large independent software vendors (ISVs) with their headquarters in the Netherlands. One ISV is headquartered in Brazil. Although all ISVs create different products focusing on different markets, a number of attributes are in common. For one, all 16 ISVs create and sell their software business to business. In terms of size, five ISVs have less than 50 employees, seven have between 100 and 200 employees, and five have between 500 and 10,000 employees. Unfortunately, we are unable to share these user story sets and their analyses due to confidentiality concerns. Because of this, we also analyzed a publicly available set of user stories created by a Duke University team for the Trident project.[6] This public dataset and its evaluation results are available online.[7] Note that due to its substantially different origin, this data set has not been incorporated in the overall statistics.

For each user story set a group of two graduate students from Utrecht University evaluated the quality of these user story sets by interpreting AQUSA's reports and applying the QUS Framework. As part of this research project, students investigated how ISVs work with user stories by following the research protocol accompanying the public dataset. Furthermore, the students assessed the quality of the company's user stories by applying the QUS Framework and AQUSA. They manually verified whether the results of AQUSA contained any false positives as well as false negatives and reported these in an exhaustive table as part of a consultancy report for the company. The first author of this paper reviewed a draft of this report to boost the quality of the reports. On top of this, we went even further to ensure the quality and uniformity of the results. An independent research assistant manually rechecked all the user stories in order to clean and correct the tables. He checked the correctness of the reported false positives and negatives by employing a strict protocol:

1. Record a false positive when AQUSA reports a defect, but it is not a defect according to the short description in the QUS Framework (Table 1).
2. Record a false negative when a user story contains a defect according to the short description in the QUS Framework (Table 1), but AQUSA misses it.
3. When a user story with a false negative contains another defect, manually fix that defect to verify that

[6] http://blogs.library.duke.edu/digital-collections/2009/02/13/on-the-trident-project-part-1-architecture/.

[7] http://staff.science.uu.nl/~lucas001/rej_user_story_data.zip.

AQUSA still does not report the false negative. If it does, remove the false negative. This is relevant in some cases: (1) When a user story is not well-formed, AQUSA does not trigger remaining analyzers; (2) When a minimality error precedes a false negative atomicity error, removing the minimal text changes the structure of the user story which may improve the linguistic parser's accuracy.

5.1 Results

The quantitative results of this analysis are available in Table 3. For each user story dataset, we include:

Def	The total number of defects as detected by AQUSA.
FP	The number of defects that were in the AQUSA report, but were not actually a true defect.
FN	The number of defects that should be in the AQUSA report, but were not.

From this source data, we can extract a number of interesting findings. At first glance, the results are promising, indicating high potential for successful further development. The average number of user stories with at least one defect as detected by AQUSA is 56 %.

The average recall and precision of AQUSA for all the company sets is given in Table 4. Note the differences between the average and weighted average (macro- vs. micro-) for recall and precision [50]. This highlights the impact of outliers like #13 SupplyComp, having only 2 violations, 0 false positives, and 1 false negative out of 50 user stories. For the micro-average, the number of violations of each set is taken into account, while the macro-average assigns the same weight to every set . This means that #13 SupplyComp its macro-average 67 % recall and 100 % precision weighs as much as all other results, while for the micro-average calculations its impact is negligible.

In total, AQUSA fulfills the desired Perfect Recall Condition for five cases, obtains between 90 and 100 % of defects for six sets and manages to get between 55 and 89 % for the remaining six. AQUSA's results for precision are not as strong, but this is expected because of our focus on the Perfect Recall Condition. For just two sets, AQUSA manages to get 100 % precision; for five sets, precision is between 90 and 100 %; three sets are only just below this number with 88–90 %. In seven cases, however, precision is rather low with a range of 50–73 %. While AQUSA is unable to achieve 100 % recall *and* precision for any of the sets, some do come close: for companies 7, 8, 9, 11, and 14, AQUSA v1 achieves recall and precision higher than 90 %. We discuss some improvements in Sect. 6.

Looking at the distribution of violations in Table 3 and the total number of violations, false positives, and false negatives in Table 5, a number of things stand out. With the exception of the quality criteria *unique*, the absolute number of false positives lies close to one another. Relatively speaking, however, *well-formed* and *atomic* stand out. Approximately 50–60 % of violations as detected by AQUSA are false positives. Similarly, the number of false negatives is particularly large for *atomic*, *minimal*, and *uniform*. In the remainder of this section, we investigate the causes for these errors.

Atomic Throughout the user story sets, the most frequently occurring false positive is caused by the symbol "&" within a role such as: "As an Product Owner W&O" and "As an R&D Manager" ($n = 38$). As we show in Sect. 6, this can be easily improved upon. The other two main types of false positives, however, are more difficult to resolve: nouns incorrectly tagged as nouns triggering the AtomicAnalyzer ($n = 18$) and multiple conditions with verbs interspersed ($n = 14$).

Tallying the number of false negatives, we find a diversity of causes. The biggest contributor is that forward or backward slashes are not recognized as a conjunction and thus do not trigger the atomic checker ($n = 5$). A more significant issue, however, is that our strategy of checking whether a verb is present on both sides of the conjunction backfired in two cases. Specifically, the words "select" and "support" were not recognized as a verb by the CoreNLP part-of-speech tagger, which employs a probabilistic maximum entropy algorithm that miscategorized these words as nouns.

Minimal The primary cause for minimality false positives is the idiosyncratic use of a symbol at the start of a user story such as the asterisk symbol ($n = 24$). Although a fairly easy false positive to prevent from occurring, the fix will introduce false negatives because in some cases a symbol at the start is an indication of a minimality error. Because our priority is to avoid false negatives, we have to accept these false positives as an unavoidable byproduct of the AQUSA tool. Another frequently occurring error is abbreviations or translations between brackets ($n = 14$). It might be possible to reduce this number with custom methods.

The seven false negatives for minimality primarily concern idiosyncratic, very specific textual constructs that are unsupported by AQUSA v1. For example, dataset 11 (AccountancyComp) delivered two user stories with superfluous examples preceded by the word "like." HealthComp (dataset 10) has three very large user stories with many different if clauses and additional roles included in the means and one user story with an unnecessary precondition interspersed between the role and means.

Well-formed The vast majority of false positives is due to unexpected, irregular text at the start of a user story which AQUSA v1 is unable to properly handle ($n = 32$). Examples are: "[Analytics] As a marketing analyst" and

Table 3 Detailed results split per data sets, showing number of defects correctly detected (Def), false positives (FP), and false negatives (FN)

	1: ResearchComp			2: ExpenseComp			3: EnterpriseComp			4: DataComp			5: RealProd		
	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN
Atomic	5	2	1	10	4	0	1	1	0	6	0	1	6	3	2
Minimal	6	3	0	3	1	0	25	5	0	4	2	0	16	6	0
Well-formed	6	4	0	1	1	0	33	21	0	2	0	0	0	0	0
Uniform	17	8	0	27	9	0	38	17	0	7	0	0	9	0	1
Unique	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0
SUM	36	17	1	41	15	0	97	45	0	19	2	1	33	9	3
N, precision, recall	50	53 %	95 %	50	63 %	100 %	50	55 %	100 %	23	89 %	94 %	51	73 %	89 %

	6: E-ComComp			7: EmailComp			8: ContentComp			9: CMSComp			10: HealthComp		
	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN
Atomic	7	5	0	12	6	0	9	2	3	1	0	1	8	1	2
Minimal	20	6	0	6	0	0	6	0	0	10	0	0	5	1	0
Well-formed	8	8	1	8	0	0	0	0	0	2	0	0	0	0	0
Uniform	33	4	1	36	0	0	34	0	0	35	0	0	11	0	7
Unique	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SUM	68	23	2	62	6	0	49	2	3	48	0	1	24	2	9
N, precision, recall	64	66 %	96 %	77	90 %	100 %	50	96 %	94 %	35	100 %	98 %	41	92 %	71 %

	11: AccountancyComp			12: PharmacyComp			13: SupplyComp			14: IntegrationComp			15: HRComp		
	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN
Atomic	12	2	0	10	3	1	4	2	1	3	2	0	21	7	6
Minimal	0	0	2	1	0	4	2	0	0	1	0	0	52	28	0
Well-formed	0	0	0	0	0	0	0	0	0	0	0	0	44	26	1
Uniform	11	0	0	14	0	9	0	0	0	46	0	0	41	17	0
Unique	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SUM	41	2	2	25	3	14	6	2	1	50	2	0	158	78	7
N, precision, recall	53	95 %	95 %	47	88 %	61 %	54	67 %	80 %	65	96 %	100 %	207	51 %	92 %

	16: FilterComp			17: FinanceComp			Public 1: Duke University		
	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN
Atomic	42	39	0	13	4	0	10	1	2
Minimal	5	0	0	25	5	0	4	3	0
Well-formed	0	0	0	0	0	0	0	0	0
Uniform	38	0	0	29	0	0	18	0	0
Unique	2	0	0	6	0	0	0	0	0
SUM	87	39	0	73	9	0	32	4	2
N, precision, recall	51	55 %	100 %	55	88 %	100 %	48	88 %	93 %

"DESIGN the following request: As a Job coach …" by company 3 and 15. Although these are not well-formed defects themselves, this text should not be included at all which means the violation itself is not without merit. Nevertheless, AQUSA could improve the way these violations are reported because these issues are also reported as a minimality violation. Similar to the minimality violations, a well-formedness error is also recorded when a symbol such as the asterisk starts the user story ($n = 24$) because AQUSA v1 is unable to detect a role.

Table 4 Overall recall and precision of AQUSA v1, computed using both the micro- and the macro-average of the data sets

	Recall (%)	Precision (%)
Macro	92.1	77.4
Micro	93.8	72.2

Table 5 Number of defects, false positives, false negatives, recall, and precision per quality criterion

$n = 1023$	Totals				
	# Def	# FP	# FN	Rec (%)	Prec (%)
Atomic	170	83	18	82.9	51.18
Minimal	187	57	6	95.5	69.52
Well-formed	104	60	2	95.7	42.31
Uniform	426	55	18	95.4	87.09
Unique	30	0	0	100	100
SUM	917	255	44	93.8	72.2

There are only two false negatives for the well-formedness criterion. Both of these user stories, however, include other defects that AQUSA v1 does report on. Fixing these will automatically remove the well-formedness error as well. Therefore the priority of resolving these false negatives is low.

Uniform The false positives are caused by a combination of the factors for minimality and well-formedness. Due to the text at the start, the remainder of the user story is incorrectly parsed, triggering a uniformity violation. Instead, these errors should be counted only as a minimal error and the remainder of the story re-analyzed as a regular user story.

The 22 uniformity false negatives are all similar: the user story expresses an ends using an unorthodox format. This can be either a repetition of ”I want to“ or a completely unknown like ”this way I.“ AQUSA v1 does not recognize these as ends, instead considering them as a valid part of the means—leading to a situation where AQUSA v1 never even tests whether this might be a deviation from the most common format.

Unique The recall and precision score for all unique measures is 100 %. This is because AQUSA v1 focuses only on exact duplicates, disregarding all semantic duplicates. One could argue that the data sets must thus contain a number of false negatives for unique. Unfortunately, we found in our analysis that this is very difficult to detect without intimate knowledge of the application and its business domain. Unsurprisingly, considering that the importance of domain knowledge for RE is well documented in the literature [57]. Exact duplicates do not occur in the data very often. Only company 11 has 18 violations in its set—the precise reason for why these duplicates are included is unclear.

5.2 Threats to validity

We discuss the most relevant threats to validity for our empirical study. For one, there is a *selection bias* in the data. All the analyzed user story sets are supplied by Independent Software Vendors (ISVs). Moreover, the majority of these ISVs originate from and have their headquarters in the Netherlands. This means that the evaluation results presented above might not be generalizable to all other situations and contexts. Indeed, the user stories from a tailor-made software company with origins in a native English speaking country could possess further edge cases which would impact the recall and precision of AQUSA.

Furthermore, the analysis suffers from *experimenter bias* because the quality criteria of the QUS Framework may have different interpretations. Thus, the independent researcher's understanding of the framework impacts the resulting analysis. To mitigate this, the independent researcher received one-on-one training from the first author, immediate feedback after his analysis of the first user story set and was encouraged to ask questions if something was unclear to him. In some cases a subjective decision had to be made, which the independent researcher did without interference from the authors. In general, he would opt for the most critical perspective of AQUSA as possible. Nevertheless, the data set includes some false negatives and false negatives the first author would not count as such himself.

6 Enhancements: toward AQUSA v2

To enhance AQUSA and enrich the community's understanding of user stories, we carefully examined each false positive and false negative. By analyzing each user stories in detail, we identified *seven* edge cases that can be addressed to achieve a substantial enhancement of AQUSA both in terms of precision and recall.

6.1 FN: unknown ends indicator

One of the most problematic type of false negatives is the failure to detect irregular formats because AQUSA is not familiar with a particular ends indicator (instead of the classic ”so that“). A simple first step is to add the unorthodox formats available in our data set. This tailored approach, however, is unsustainable. We should, thus, make AQUSA v2 customizable, so that different organizations can define their own vocabulary. Moreover, a crowdsourcing feature that invites users to report whenever one of their indicators is not detected by AQUSA should quickly eradicate this problem.

6.2 FN: indicator and chunk repetition

A particular elusive problem is the repetition of indicators and accompanying role, means, or ends chunks. When AQUSA v1 encounters an indicator text, all text afterward is a part of that chunk until it encounters an indicator text for the subsequent chunk. Consequentially, AQUSA does not raise any red flags when for example (1) a second role chunk is interspersed between the means and ends section like "As a pharmacist, I want to …, if …, if …, I as a wholesale employee will prevent …" or (2) a known means indicator format is used to express an ends as in "I want to change my profile picture because I want to express myself." To solve this problem, AQUSA v2 will scan for all occurrences of indicator texts and generate a warning whenever an indicator type occurs twice in the user story.

6.3 FN: add slash and greater than

One very simple improvement is to include the forward and backward slash symbols "/" and "\" in the list of conjunctions. In one user story, the greater than symbol ">" was used to denote procedural steps in the user story, prompting us to include this symbol and its opposite "<" in the list of conjunctions as well. Together, these simple improvements reduce the number of atomicity false negatives by one third.

6.4 FN: verb phrases after "and"

AQUSA v1 takes a "smart" approach to detecting atomicity errors. Whenever the atomic analyzer encounters a conjunction like "and," a POS tagger makes sure a verb is present on both sides of the conjunction. When this is not the case, it is likely that the user story does not include two separate actions. For 3 user stories in our data sets, however, the POS tagger incorrectly tags a verb as a noun introducing a false negative. This is not surprising. Because no available POS tagger is perfect, our approach is guaranteed to not to achieve the Perfect Recall Condition in all cases.

There are two options to resolve this issue. The simple method is to remove this smart approach and simply report all user stories that include a conjunction in the means as violating the atomic quality criteria. The problem is, however, that this introduces a substantial number of false positives. An alternative approach is to include exceptions in the POS tagger for a specific domain. In our dataset, we see that the four incorrectly tagged nouns are common actions in software development: select, support, import, and export. Compiling such an exception list does not guarantee the prevention of false negatives, but would improve the situation without re-introducing many false positives.

6.5 FP: symbols and starting text

Symbols cause the vast majority of false positives in our set of user stories. We want to resolve these without introducing new false negatives. To do this, we plan to enhance AQUSA in two ways.

Symbol in role Many user stories include a reference to a department as part of the role. When AQUSA v2 encounters an ampersand (&) or plus sign (+) in the role chunk, it takes the following approach:

1. Check whether there is a space before or after the ampersand/plus.
2. Count whether the number of characters before and after the ampersand/plus is bigger than a specific threshold such as three.
3. Run the POS tagger to check whether the phrases before and after the ampersand/plus are actual words. Exceptions are "I" and "A."

Only when the answer to two or more of these checks is no, AQUSA v2 records an atomicity violation.

Removing symbol (text) Whenever a user story has text with a symbol before the start of the role chunk, AQUSA v1 is unable to properly apply its analysis. To resolve this issue, AQUSA v2 will try to remove the symbol and any associated text preceding the first indicator text, check whether a valid user story remains and then rerun its analyses.

6.6 FP: abbreviations and translations

The majority of false positives for the minimality quality criterion are caused by an abbreviation or translation of a word in between brackets. To reduce this number, whenever the minimality analyzer detects a single phrase in between brackets it verifies whether the phrase could be an abbreviation of the word or word group immediately before the phrase.

6.7 Expected results

We expect that introducing these enhancements will generate a substantial improvement in terms of recall and precision. To foresee how substantial this improvement would be, we categorized all false positives and false negatives and removed those that the enhancements should be able to prevent from occurring by conducting a manual analysis of the data set. The results of our analysis are that the new micro-averaged recall and precision for this collection of user story sets would be 97.9 and 84.8 % (compare this to the values of AQUSA v1: recall 93.8 % and precision 72.2 %). With these changes, AQUSA would fulfill the Perfect Recall Condition for 9 of the 18 data sets.

7 Related work

We discuss relevant works about the syntax and use of user stories (Sect. 7.1), quality of requirements (Sect. 7.2), and applications of NLP to RE (Sect. 7.3).

7.1 User stories

Despite their popularity among practitioners [29, 53], research efforts concerning user stories are limited. While scenario-based requirements engineering has been studied since the 1990s [22, 34], the earliest research work on user stories proposes their use as the initial artifact in the design of human–computer interaction systems [25], and argues that user stories contain the intention and motives of a user. In later design stages, the authors propose to transform user stories into the more formal notation of use cases.

The majority of research in the field, however, attempts to create methods and tools that support or improve user story practice. Rees [45] proposes to replace the pen-and-card approach for writing user stories with the DotStories software tool to translate the index card metaphor to a digital environment. This tool relies on so called *teglets* which are “small rectangular areas on a webpage” … “users can drag teglets around the page in a similar manner to laying out cards on a table.” Today, most project management tools for agile software development with user stories are built around the same interface design, including Jira, PivotalTracker, Taiga, and Trello.

Observing that the simple comparison of a user story with a pair of other user stories is insufficient to accurately estimate user story complexity, Miranda et al. [41] propose a paired comparison estimation approach using incomplete cyclic designs. This work reduces the number of necessary comparisons while still producing reliable estimations. In industry, however, planning poker remains the de facto standard for estimating user story complexity. In a comparative study, Mahnič and Havelja found that the estimates from planning poker played by experts tend to be more accurate than the mean of all individual expert estimates [38].

Liskin et al. investigate the expected implementation duration of user story as a characteristic of granularity. They find that in practitioners’ experience combining the effort estimation of two small, clear-cut user stories produces more accurate results than when estimating a sinÆŠe, larger, more opaque user story [32]. Dimitrijevic et al. qualitatively compare five agile software tools in terms of their functionality, support for basic agile RE concepts and practices, and user satisfaction. They conclude that basic functionality is well supported by tools, but that user role modeling and personas are not supported at all [12].

In line with our conceptual model of Fig. 1, some authors have linked user stories with goals. Lin et al. [30] propose a mixed top-down and bottom-up method where an initial top-down analysis of the high-level goals is complemented by a bottom-up approach that derives more refined goals by analyzing user stories. A similar attempt has been implemented in the US2StarTool [39], which derives skeletons of *i** goal models starting from user stories. The key difference is that these models represent user stories as social dependencies from the role of the user stories to the system actor.

Other recent work is revisiting user stories from a conceptual perspective. Wautelet et al. [54] propose a unified model for user stories with associated semantics based on a review of 85 user story templates and accompanying example stories—20 from academic literature and 65 from practice. Gomez et al. [20] propose a conceptual method for identifying dependencies between user stories, relying on the data entities that stories refer to. Although related, these are different approaches from our conceptual model presented in Sect. 2, which does not aim at reconciling and supporting all possible dialects.

7.2 Quality of requirements

Multiple frameworks exist for characterizing the quality of (software) requirements. The *IEEE Standard Systems and software engineering—Life cycle processes—Requirements engineering* is the standard body of work on this subject, defining characteristics of a single requirement, sets of requirements as well as linguistic characteristics for individual requirements [23]. Unfortunately, most requirements specifications are unable to adhere to them in practice [19], although evidence shows a correlation between high-quality requirements and project success [28].

Heck and Zaidman created the *Agile Requirements Verification Framework*, which is a tailor-made quality framework for software development in an agile context. Some authors do mention a selection of quality recommendations [13, 37], but the majority of these are generic, organizational pieces of advice for high-level processes. One exception is the rigorous set of validation checks by Zowghi and Gervasi [58] to detect errors in requirements specifications. As part of their NORPLAN framework, Farid and Mitropoulos [15] propose requirements quality metrics for non-functional requirements. Aside from ambiguity, however, these metrics are based on the processes related to requirements instead of the actual requirements themselves. Patel and Ramachandran [43] propose the *Story Card Maturity Model*, a CMM-based process improvement model for story cards and their key process areas. They identify maturity levels that consist of six to seven key process areas with specific activities to obtain that maturity level. An example is the key process area of defining a standard story card structure which defines seven key attributes to include, or the risk assessment key process area which prescribes the user to (1) understand the problem, (2) assess story card risk, and (3) include the risk assessment on the story card. Unfortunately, however, their maturity model has not been validated yet.

7.3 Natural language processing for RE

Applying NLP to RE has historically been heralded as the final frontier of RE. Nowadays, the ambitious objective of automation is regarded as unattainable in the foreseeable future [4, 47].

Therefore, RE research has applied NLP to specific use cases. Trying to determine and/or improve the quality of requirements documents using NLP is a popular research domain for example. The DODT tool applies NLP and a domain ontology to semi-automatically transform NL requirements into higher quality semi-boilerplate requirements for embedded systems development [14]. The MaramaAI tool extracts semi-formal *Essential Use Cases* (EUC) models from natural language requirements and enables an end user to detect inconsistency, incompleteness, and incorrectness by visually highlighting the differences to a baseline EUC [27]. The EuRailCheck project uses rule-based techniques and ontologies to validate formalized requirement fragments and pinpoint flaws that are not easy to detect in an informal setting [8]. The HEJF tool bases its detection of "requirements smells" that are checked against the ISO 29148 [23] requirements quality standard. Their proposed light-weight approach uses POS-tagging, morphological analysis, and dictionaries to detect linguistic errors in requirements. In an evaluation with industry experts, they obtained positive feedback despite the inability to fulfill the Perfect Recall Condition [16].

Some tools look at very specific aspects of parsing natural language requirements. The SREE tool aims to detect a scoped set of ambiguity issues with 100 % recall and close to 100 % precision by using a lexical analyzer instead of a syntactic analyzer [51]. Although their precision is only 66 %, they argue that using their tool is still faster and more reliable than manually searching for all instances of ambiguity. Yang et al. combines lexical and syntactical analyzers with an advanced technique from the machine learning domain called conditional random fields (CRFs) to detect uncertainty in natural language requirements. They apply their tool to 11 full-text requirements documents and find that it performs reasonably well in identifying uncertainty cues with F-scores of 62 % for auxiliaries, verbs, nouns, and conjunctions. On the other hand, it under-performs in identifying the scope of detected uncertainty causing the overall F-score to drop to 52 % [56].

Another class of NLP for RE tools extract specific elements from natural language requirements. The NFR locator [49] uses a vector machine algorithm to extract non-functional requirements from install manuals, requests for proposals and requirements specifications. Their approach was twice as effective as a multinomial naïve Bayes classifier. The glossary tool suite by Arora et al. applies a collection of linguistic techniques to automatically extract relevant domain specific concepts to generate a glossary of terms. Their approach outperforms earlier tools in candidate term identification thanks to the application of tailor-made heuristics [1]. Finally, the Text2Policy tool attempts to extract access control policies (ACP) from natural language documents to reduce the manual effort for this tedious but important security task. Using both syntactic and semantic methods, this tool achieves accuracies ranging between 80 and 90 % for ACP sentence, rule, and action extraction [55]. The generation of models from natural language requirements has also been studied; for example, Friedrich et al. [17] combined and augmented several NLP tools to generate BPMN models, resulting in an accuracy of 77 % on a data set of text–model pairs from industry and textbooks.

Berry et al. argue that these tools suffer from lack of adoption because of their inaccuracy. If the tool provides less than 100 % recall, the analyst still has to repeat the entire task manually without any tool support. He proposes—and we support his position—that tools that want to harness NLP should focus on the *clerical* part of RE that software can perform with 100 % recall and high precision, leaving thinking-required work to human requirements engineers [4].

8 Conclusion and future research

In this paper, we presented a holistic approach for ensuring the quality of agile requirements expressed as user stories. Our approach consists of (1) the QUS framework, which is a collection of 13 criteria that one can apply to a set of user stories to assess the quality of individual stories and of the set and (2) the AQUSA software tool, that employs state-of-the-art NLP techniques to automatically detect violations of a selection of the quality criteria in the QUS framework.

In addition to laying foundations for quality in agile requirements, the implementation and evaluation of AQUSA on over 1000 user stories from 18 organizations provide evidence about the viability of the Perfect Recall Condition. According to this condition, NLP tools for RE should focus on the *clerical* activities that can be automated with 100 % recall and high-enough precision. Our results show that for some syntactic quality criteria of the QUS framework it is possible to achieve results that are close to the Perfect Recall Condition.

Based on our case studies, we have identified a number of easy-to-implement improvements that will be included in AQUSA v2. Although these improvements originate from 18 different cases, we will have to determine whether these changes lead to over-fitting for the datasets that we have studied so far, and if the actual benefit is as good as we expect.

Further research directions exist that future work should address. The effectiveness of the QUS framework as a quality framework for user stories should be studied in case studies and action research, which may lead to further improvements. Longitudinal studies should be conducted to determine the effectiveness of the AQUSA tool while the requirements database is being populated, as opposed to the current case studies where an existing requirements database was imported. To do this, an approach that explains how to embed AQUSA and the QUS Framework in a standard agile development environment is necessary. The challenge of reducing the number of false positives while staying (close to) 100 % recall will be a central direction to follow for AQUSA development. To determine whether our approximation of the Perfect Recall condition is sufficient, we will evaluate AQUSA's performance in comparison to human analysts. After all, humans are unable to achieve the Perfect Recall Condition themselves [4]. Finally, it is necessary to study whether and to what extent the pragmatic and semantic quality criteria can be included in AQUSA, at least to assist the engineering in specific sub-problems for which our recall/precision goals can be met.

Acknowledgments This paper would not have been made possible without the contributions of many people. We would like to thank Floris Vlasveld, Erik Jagroep, Jozua Velle, and Frieda Naaijer for providing the initial real-world user story sets. We also thank the graduate students who participated in the course Software Product Management for their hard work in finding user stories to examine. Special thanks go to Marcel Robeer who did great work as independent research assistant. Petra Heck and Andy Zaidman's collection of agile requirements quality literature was of great help for writing Sect. 7.2. Finally, we thank the companies who generously contributed sets of user stories, time, and feedback.

References

1. Arora C, Sabetzadeh M, Briand L, Zimmer F (2014) Improving requirements glossary construction via clustering: approach and industrial case studies. In: Proceedings of the ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)
2. Beck K (1999) Extreme programming explained: embrace change. Addison-Wesley, Boston
3. Beck K, Fowler M (2000) Planning extreme programming, 1st edn. Addison-Wesley Longman, Boston
4. Berry D, Gacitua R, Sawyer P, Tjong S (2012) The case for dumb requirements engineering tools. In: Proceedings of international conference on requirements engineering: foundation for software quality (REFSQ), LNCS, vol 7195. Springer, pp 211–217
5. Berry DM, Kamsties E (2004) Ambiguity in requirements specification. In: Perspectives on software requirements, international series in engineering and computer science, vol 753. Springer, pp 7–44
6. Bucchiarone A, Gnesi S, Pierini P (2005) Quality analysis of NL requirements: an industrial case study. In: Proceedings of the IEEE international conference on requirements engineering (RE), pp 390–394
7. Cao L, Ramesh B (2008) Agile requirements engineering practices: an empirical study. Software 25(1):60–67
8. Cimatti A, Roveri M, Susi A, Tonetta S (2013) Validation of requirements for hybrid systems: a formal approach. ACM Trans Softw Eng Methodol 21(4):22:1–22:34
9. Cleland-Huang J, Berenbach B, Clark S, Settimi R, Romanova E (2007) Best practices for automated traceability. Computer 40(6):27–35
10. Cohn M (2004) User stories applied: for agile software development. Addison Wesley, Boston
11. Cooper A (1999) The inmates are running the asylum. Macmillan, Indianapolis
12. Dimitrijević S, Jovanović J, Devedžić V (2015) A comparative study of software tools for user story management. Inf Softw Technol 57:352–368
13. Dumas-Monette JF, Trudel S (2014) Requirements engineering quality revealed through functional size measurement: an empirical study in an agile context. In: Proceedings of the international workshop on software measurement (IWSM), pp 222–232
14. Farfeleder S, Moser T, Krall A, Stålhane T, Zojer H, Panis C (2011) DODT: increasing requirements formalism using domain ontologies for improved embedded systems development. In: Proceedings of the IEEE international symposium on design and diagnostics of electronic circuits systems (DDECS), pp 271–274
15. Farid W, Mitropoulos F (2013) Visualization and scheduling of non-functional requirements for agile processes. In: Proceedings of the IEEE Region 3 technical, professional, and student conference (Southeastcon), pp 1–8
16. Femmer H, Fernández DM, Juergens E, Klose M, Zimmer I, Zimmer J (2014) Rapid requirements checks with requirements smells: two case studies. In: Proceedings of the international workshop on rapid continuous software engineering (RCoSE), pp 10–19

17. Friedrich F, Mendling J, Puhlmann F (2011) Process model generation from natural language text. In: Proceedings of the international conference on advanced information systems engineering (CAiSE), LNCS, vol 6741. Springer, pp 482–496
18. Gacitua R, Sawyer P, Gervasi V (2010) On the effectiveness of abstraction identification in requirements engineering. In: Proceedings of the IEEE international requirements engineering conference (RE), pp 5–14
19. Glinz M (2000) Improving the quality of requirements with scenarios. In: Proceedings of the World Congress on Software Quality (WCSQ), pp 55–60
20. Gomez A, Rueda G, Alarcón P (2010) A systematic and lightweight method to identify dependencies between user stories. In: Proceedings of the international conference on agile software development (XP), LNBIP, vol 48. Springer, pp 190–195
21. Heck P, Zaidman A (2014) A quality framework for agile requirements: a practitioner's perspective. http://arxiv.org/abs/1406.4692
22. Holbrook H III (1990) A scenario-based methodology for conducting requirements elicitation. SIGSOFT Softw Eng Notes 15(1):95–104
23. IEEE: systems and software engineering—life cycle processes—requirements engineering. ISO/IEC/IEEE 29148:2011(E), pp 1–94, 2011
24. IEEE Computer Society (1994) IEEE recommended practice for software requirements specifications. IEEE Std 830-1993
25. Imaz M, Benyon C (1999) How stories capture interactions. In: Proceedings of the IFIP international conference on human–computer interaction (INTERACT), pp 321–328
26. Jeffries RE, Anderson A, Hendrickson C (2000) Extreme programming installed. Addison-Wesley Longman, Boston
27. Kamalrudin M, Hosking J, Grundy J (2011) Improving requirements quality using essential use case interaction patterns. In: Proceedings of the international conference on software engineering (ICSE), pp 531–540
28. Kamata MI, Tamai T (2007) How does requirements quality relate to project success or failure? In: Proceedings of the IEEE international requirements engineering conference (RE). IEEE, pp 69–78
29. Kassab M (2015) The changing landscape of requirements engineering practices over the past decade. In: Proceedings of the IEEE international workshop on empirical requirements engineering (EmpiRE). IEEE, pp 1–8
30. Lin J, Yu H, Shen Z, Miao C (2014) Using goal net to model user stories in agile software development. In: Proceedings of the IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD). IEEE, pp 1–6
31. Lindland OI, Sindre G, Sølvberg A (1994) Understanding quality in conceptual modeling. IEEE Softw 11(2):42–49
32. Liskin O, Pham R, Kiesling S, Schneider K (2014) Why We need a granularity concept for user stories. In: Proceedings of the international conference on agile software development (XP), LNBIP, vol 179. Springer, pp 110–125
33. Lombriser P, Dalpiaz F, Lucassen G, Brinkkemper S (2016) Gamified requirements engineering: model and experimentation. In: Proceedings of the 22nd international working conference on requirements engineering: foundation for software quality (REFSQ)
34. Loucopoulos P, Karakostas V (1995) System requirements engineering. McGraw-Hill, New York
35. Lucassen G, Dalpiaz F, van der Werf JM, Brinkkemper S (2015) Forging High-quality user stories: towards a discipline for agile requirements. In: Proceedings of the IEEE international conference on requirements engineering (RE). IEEE, pp 126–135
36. Lucassen G, Dalpiaz F, van der Werf JM, Brinkkemper S (2016) The use and effectiveness of user stories in practice. In: Proceedings of the international conference on requirements engineering: foundation for software quality (REFSQ), LNCS, vol 9619. Springer, pp 205–222
37. Lucia A, Qusef A (2010) Requirements engineering in agile software development. J Emerg Technol Web Intell 2(3):212–220
38. Mahnič V, Hovelja T (2012) On using planning poker for estimating user stories. J Syst Softw 85(9):2086–2095
39. Mesquita R, Jaqueira A, Agra C, Lucena M, Alencar F (2015) US2StarTool: generating i* models from user stories. In: Proceedings of the international i* workshop (iStar)
40. Miller GA (1995) WordNet: a lexical database for English. Commun ACM 38(11):39–41
41. Miranda E, Bourque P, Abran A (2009) Sizing user stories using paired comparisons. Inf Softw Technol 51(9):1327–1337
42. Paja E, Dalpiaz F, Giorgini P (2013) Managing security requirements conflicts in socio-technical systems. In: Proceedings of the international conference on conceptual modeling (ER), LNCS, vol 8217, pp 270–283
43. Patel C, Ramachandran M (2009) Story card maturity model (SMM): a process improvement framework for agile requirements engineering practices. J Softw 4(5):422–435
44. Popescu D, Rugaber S, Medvidovic N, Berry DM (2008) Reducing ambiguities in requirements specifications via automatically created object-oriented models. In: Innovations for requirement analysis. From Stakeholders' needs to formal designs, LNCS, vol 5320. Springer, pp 103–124
45. Rees M (2002) A feasible user story tool for agile software development? In: Proceedings of the Asia-Pacific software engineering conference (APSIC), pp 22–30
46. Robinson WN (1989) Integrating multiple specifications using domain goals. SIGSOFT Softw Eng Notes 14(3):219–226
47. Ryan K (1993) The role of natural language in requirements engineering. In: Proceedings of the IEEE international symposium on requirements engineering (ISRE), pp 240–242. IEEE
48. Simon H (1996) The sciences of the artificial, 3rd edn. MIT Press, Cambridge
49. Slankas J, Williams L (2013) Automated extraction of non-functional requirements in available documentation. In: Proceedings of the international workshop on natural language analysis in software engineering (NaturaLiSE), pp 9–16
50. Tague-Sutcliffe J (1992) The pragmatics of information retrieval experimentation. Revisited. Inf Process Manag 28(4):467–490
51. Tjong SF, Berry DM (2013) The design of SREE: a prototype potential ambiguity finder for requirements specifications and lessons learned. In: Proceedings of the international conference on requirements engineering: foundation for software quality (REFSQ), LNCS, vol 7830. Springer, pp 80–95
52. Wake B (2003) INVEST in good stories, and SMART tasks. http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Accessed 2015-02-18
53. Wang X, Zhao L, Wang Y, Sun J (2014) The role of requirements engineering practices in agile development: an empirical study. In: Proceedings of the Asia Pacific requirements engineering symposium (APRES), CCIS, vol 432, pp 195–209
54. Wautelet Y, Heng S, Kolp M, Mirbel I (2014) Unifying and extending user story models. In: Proceedings of the international conference on advanced information systems engineering (CAiSE), LNCS, vol 8484. Springer, pp 211–225
55. Xiao X, Paradkar A, Thummalapenta S, Xie T (2012) Automated extraction of security policies from natural-language software documents. In: Proceedings of the ACM SIGSOFT international symposium on the foundations of software engineering (FSE). ACM, pp 12:1–12:11
56. Yang H, De Roeck A, Gervasi V, Willis A, Nuseibeh B (2012) Speculative requirements: automatic detection of uncertainty in

natural language requirements. In: Proceedings of the IEEE international requirements engineering conference (RE), pp 11–20

57. Zave P, Jackson M (1997) Four dark corners of requirements engineering. ACM Trans Softw Eng Methodol 6(1):1–30
58. Zowghi D, Gervasi V (2003) On the interplay between consistency, completeness, and correctness in requirements evolution. Inf Softw Technol 45(14):993–1009