Detecting Redundancies between User Stories with Graphs and Large Language Models

Lukas Sebastian Hofmann[1,2], Alexander Lauer[1], Jens Kosiol[1], Arno Kesper[1], Philipp Wieber[1], Amir Rabieyan[1], and Gabriele Taentzer[1]

[1] Philipps-Universität Marburg, Marburg, Germany
{lukas.hofmann, alexander.lauer, jens.kosiol, arno.kesper, taentzer}@uni-marburg.de, {rabieyan, wieberp}@mathematik.uni-marburg.de
[2] Universidad Complutense de Madrid, Madrid, Spain

Abstract. [**Context and motivation**] User stories (USs) are a widely used notation for requirements in agile software development. [**Question/problem**] In large software projects, redundancies between USs can easily occur, and unresolved redundancies can impact software quality. It is crucial for requirements engineers to know where redundancy occurs in their projects. However, some forms of redundancy may be acceptable. [**Principal ideas/results**] We present two automated approaches for detecting redundancy in a set of USs in order to prevent a decrease of software quality due to the realisation of redundant USs. The first approach is based on *annotation graphs*, containing the main actions and entities of a US. By design, this approach effectively identifies a *strict* form of redundancy. The second approach detects redundancies of a more semantic nature using large language models (LLMs). [**Contribution**] We present the concepts and tools of both approaches and evaluate their potential and limitations by applying them to a large corpus of USs. Our results show that the inherently fuzzy LLM-based approach is able to detect most of the strict redundancies and additionally finds many more non-strict *semantic* redundancies. Thus, this study contributes to the advancement of automated quality control of USs.

Keywords: User Story Quality · Redundancy · Graph · Large Language Model · Agile Development

1 Introduction

User stories (USs) are short requirements specifications from the user's point of view that are increasingly used in agile requirements engineering. In fact, they have become a common notation for requirements in agile projects [19]. As a key development artefact, USs with poor quality are detrimental to the quality of the resulting code [12]. Several sources, such as the IEEE Recommended Practice for Software Requirements Specifications [9], the INVEST framework [4], the Quality User Story Framework (AQUSA) [12], or Heck and Zaidman [8], present quality criteria for (sets of) USs, such as correctness, unambiguity, completeness, and uniqueness. However, despite the popularity of USs, the number of methods

for assessing and improving their quality is limited and existing methods focus on quality criteria that concern only a single US (such as atomicity or well-formedness).

In this work, we contribute to a human-in-the-loop approach to the quality assurance of *sets of USs* by presenting automated approaches to identify pairs of USs that show *redundancy*. Requirement engineers (REs) should be informed about pairs of redundant USs, as these may violate the uniqueness demand of USs and lead to redundant development steps later on, particularly in scenarios where multiple contributors are working in parallel or distributed across teams over an extended period. Redundancy can also indicate conflicts between USs in the sense that they are inconsistent. In the literature, pairs of USs are either checked for exact equality [12] or for similarity [7], where it remains unclear what 'similarity' precisely means. Dedicated automated analysis techniques for detecting redundancies in USs are missing.

To fill this gap, we present the following contributions: First, we define *redundancy* as a specific kind of similarity of USs, inspired by Lucassen et al. [11]. Intuitively, redundancy should capture that in two USs the same functionality is required or/and the same purpose is addressed. We define two forms of redundancy: *strict redundancy*, which compares key information of USs for equality, and *semantic redundancy*, which accounts for equivalence in meaning and thus captures the intuition more reliably. Our definitions assume that a US comes with an *annotation graph*, which provides a trace between different parts of the US and the conceptual roles these parts play in the story. We then present two approaches to automated redundancy detection; the first is based directly on annotation graphs, the second uses a large language model (LLM). We have evaluated both approaches on a large set of USs extracted from [1,5]. The evaluation shows that the graph-based approach quickly computes all strict redundancies. The LLM-based approach is still quite accurate and provides a broader, semantic analysis of USs. Thus, it goes beyond the intrinsic limitations of strict redundancy and also finds many more semantic redundancies.

Our paper is structured as follows: First, we discuss the related work on US quality assessment in Sec. 2. We then present our contributions: In Sec. 3, we define two forms of redundancy of USs based on a graph-like conceptual model (cf. [1]) and discuss them. We then introduce two approaches for redundancy detection between USs: a direct implementation of *strict redundancy* in Sec. 4 and an approach based on LLMs, which can identify *redundancies* on a more semantic level in Sec. 5. We evaluate the effectiveness and efficiency of our redundancy detection approaches in Sec. 6 using a large set of annotated USs from [1,5]. Finally, we summarise our findings and identify future research in Sec. 7.

2 Related Work

In this section, we discuss existing approaches to automated quality control and improvement of USs, with a particular focus on approaches that use LLMs.

Lucassen et al. [12,11] introduced the *Quality User Story Framework*, which consists of 14 quality criteria for USs. Their criterion of *uniqueness* corresponds closely to the notion of redundancy presented in this paper. Although different types of uniqueness are discussed, ranging from identical USs to only semantic similarity, based on a semi-formal conceptual model of USs, the underlying proof-of-concept tool AQUSA can only detect exact duplicates of USs. This means that the tool cannot distinguish between different spellings of the same word, e.g., when one US is written in British English and the other in American English. In their evaluations of AQUSA, only two uniqueness violations are reported. The evaluation data is not published.

Duszkiewicz et al. used the *Universal Sentence Encoder* (USE) to find similarities between USs [7]. With a threshold of 70% similarity, a pair of USs is considered similar. The implementation was tested on 13 USs (i.e., 169 pairs of USs) and compared with a human evaluation. Except for 4 false positives, all similar pairs were correctly identified. The authors note that false estimations occur particularly for values such as "add" and "remove", as these differences are not sufficiently penalised by the model. In addition, the use of synonyms can lead to a large change in the assessment of the similarity of USs.

Ronanki et al. [18] analysed the quality criteria *well-formedness*, *atomicity* and *minimality* from the *Quality User Story Framework* using *ChatGPT*. They have compared the results of *ChatGPT* with the results of the *Quality User Story Framework* by human evaluation. The results show that the human evaluators have a higher agreement rate with the results of *ChatGPT* than with those of AQUSA.

Zhang et al. [23] used LLMs to improve given USs in terms of the US quality criteria formulated in the INVEST framework [4]. In their evaluation, the USs are clearer and more understandable after improvement. However, human intelligence is still required to evaluate the modified USs to minimise hallucination and increase contextual accuracy. Redundancy of user stories is not checked in this approach.

In summary, beyond searching for exact duplicates or "similar" USs, where the notion of similarity used remains imprecise, automated approaches detecting well-defined and more focused similarities between USs are lacking.

3 User Stories and their Redundancy

In this section, we introduce USs and their annotation graphs. Based on these, we semi-formally define two kinds of *redundancy*.

User stories and their annotations USs, providing a concise and accessible overview of the required functionality of a system, are an important artefact in agile software development [1,11,15]. A US is a natural language text that expresses *who* wants to perform *what* action and, potentially, *why* in the system. While no uniform definition exists [13,20], the following template, which goes back to Cohn [4], is widely used in practice [13]: 'As a <role>, I want <goal>, [so that <benefit>]', where

1. <role> describes the person involved in the US;
2. <goal> describes the actions performed by the person on *entities*; and
3. an optional <benefit> describes what the person will receive after performing these actions.

Example 1. The following two sentences are USs, slightly adapted from the dataset *g14* of Dalpiaz et al. [5], that conform to the template from above. **US1**: 'As a <Publisher>, I want <to publish a dataset>, so that <I can view just the dataset with a few people>.'; **US2**: 'As a <publisher>, I want to <publish a dataset>, so that <I can inspect the dataset with a selected set of people>.'

In order to process USs automatically, it is advantageous to *annotate* them explicitly. An annotation is based on a *conceptual model* for USs and provides a trace between the different parts of a US and the concepts they belong to.

Figure 1 shows our conceptual model which can be represented as a graph; it is a slight adaptation of the conceptual model introduced by Arulmohan et al. [1]. This model captures instances of USs in a unified format, ensuring that the core information is consistently represented, while emphasizing the key actions that will be performed on each entity. A UserStory contains a Persona that triggers Actions which, in turn, target Entities. These might contain further Entities. In contrast to Arulmohan et al. [1], who use an additional attribute to distinguish between Actions and Entities which belong to the goal or the benefit of the US, we introduce dedicated types for each.

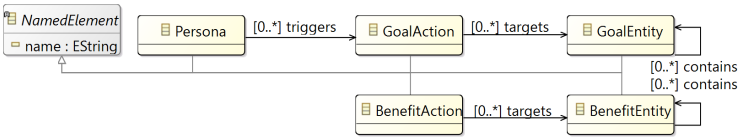

Fig. 1. The conceptual model for annotated User Stories

Definition 1 (Annotated user story). *An* annotated user story *is a user story (US) together with an instance c of the conceptual model from Fig. 1 such that each* name*-attribute of a node of c contains a part of the US text; the instance c is then called the* annotation graph *of US.*

We call a targets*-edge together with incident* GoalAction *and* GoalEntity *an* activity *of the annotated user story and a* targets*-edge together with incident* BenefitAction *and* BenefitEntity *a* reason.

Example 2. Figure 2 shows annotation graphs of the two USs from Ex. 1.

An annotated US needs to satisfy certain consistency criteria to be of value. Most importantly, the type of a node should align with the role its stored text plays in the US. For instance, the Persona-node should indicate the acting person of the US, or GoalAction should indicate the central verb, GoalEntity its object and a targets-edge should connect them. Such criteria are very hard to formally define in generality (and not just for USs following strictly a specific template);

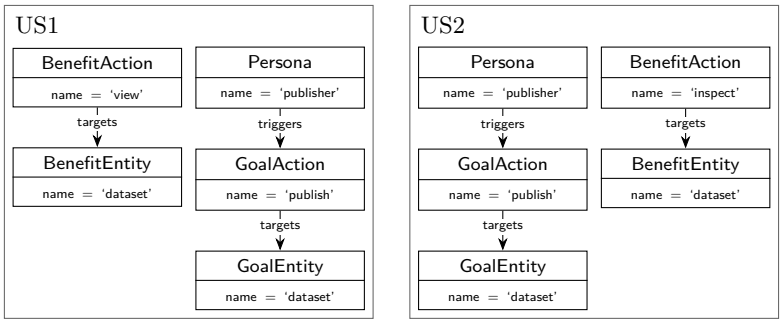

Fig. 2. Annotation graphs of the two USs from Ex. 1

the conceptual model we employ, however, is (to a certain degree) independent from such templates [1]. We therefore refrain from formally defining such criteria.

We define, however, *well-formedness* [11] as a quality criterion for annotated USs, upon which our notion of *redundancy* (and, hence, our approaches to *redundancy detection*) technically rest.

Definition 2 (Well-formedness [11]). *An annotated user story is* well-formed *if the annotation graph contains a* Persona*, an activity (i.e., a* targets*-edge that connects a* GoalAction *to a* GoalEntity*) and, if the benefit is not missing, a reason (i.e., a* targets*-edge that connects a* BenefitAction *to a* BenefitEntity*).*

In the following, we present two definitions of *redundancy* as a specific kind of similarity of a pair of (annotated) USs. Intuitively, we want to consider two USs as redundant if (i) the same actions are performed on same entities in the goal or (ii) they share the same benefit (this is inspired by similar definitions by Lucassen et al. [11]). Redundant USs might be duplicates (violating the quality criterion of uniqueness), might point to feature variation in the system or might contradict each other [11]. In any case, it is important that REs are aware of redundant USs so that they are able to make a qualified decision, like deleting a duplicated US, discussing conflicting USs with stakeholders or developing a system architecture that elegantly accounts for the feature variation.

Assuming that a US is correctly annotated, the targets-edges and their incident nodes capture exactly the central information about the US's goal and benefit. Therefore, we use this information to formally define two forms of redundancy. The *strict* form relies on the equality of strings, but thus only approximates what one would intuitively consider to constitute a redundancy. The second, *semantic* form of redundancy is related to the meaning of USs.

Definition 3 (Strict and semantic redundancy of USs). *Let two well-formed annotated user stories be given. They are* strictly redundant in their goal (benefit) *if they share an equal activity (reason), i.e., if amongst the activities (reasons) there is pair stemming from both USs such that their* GoalAction *(*BenefitAction*) and* GoalEntity *(*BenefitEntity*) have equal* name*-attributes.*

They are semantically redundant in their goal (benefit) *if they share a semantically equivalent activity (reason), i.e., if amongst the activities (reasons)*

there is a pair stemming from both USs such that their GoalAction *(*BenefitAction*) and* GoalEntity *(*BenefitEntity*) have semantically equivalent* name*-attributes.*

Note that we consider semantic equivalence to be reflexive; in particular, every strict redundancy is also a semantic one by definition.

Example 3. Consider the USs introduced in Ex. 1 and their annotation graphs (Fig. 2). Here, GoalAction, GoalEntity, and BenefitEntity are literally equal (the value of their name-attributes), whereas the BenefitActions are merely semantically equivalent ('view' and 'inspect'). Hence, this pair of USs is *semantically redundant* in their benefits and even *strictly redundant* in their goals.

To further clarify the concept of semantic redundancy, we provide another instance for US redundancy in the goal part. We classify the US pair 'As a data manager, I want to check the data model's completeness.' and 'As a data manager, I want to verify that the data schema contains all necessary attributes and relationships.' as redundant according to our definition because the actions and entities are semantically equivalent within their given context.

Discussion Our definitions of US redundancy have the following limitations: (i) both redundancy definitions depend on accurate annotation graphs and the well-formedness of USs; (ii) *strict redundancy* cannot account for misspellings and synonyms, and (iii) although *semantic redundancy* is more flexible by definition, it is difficult to analyse due to its inherent ambiguity.

However, relying on the annotation graph allows us to abstract from things like word order and to examine USs for redundancy that conform to different templates. Furthermore, the process of annotating a US can be automated [15]; and well-formedness is an established quality criterion that a US should meet anyway [11]. To guarantee well-formedness, tooling to fix ambiguous or poorly structured USs can be used, e.g., tooling by Lucassen et al. [11].

4 Strict Redundancy Analysis using Annotation Graphs

In this section, we present an approach for the detection of *strict redundancies* that is based on the annotation graphs of USs. By directly implementing Def. 3, the approach is guaranteed to be correct and complete. The pseudo-code is shown in Algorithm 1; our Python implementation is available on GitHub[3].

First, the annotated USs are translated into a JSON format in which each activity (reason), i.e., each targets edge, is represented as a pair [*src*, *dest*] where *src* is equal to the value of the name attribute of the GoalAction (BenefitAction) and *dest* is equal to the value of the name-attribute of the GoalEntity (BenefitEntity) of the respective activity (reason). Then, for each pair of USs, it is checked whether at least one *activity* (or *reason*) of the first US is contained in the second US of the pair. Using the optimised JSON format, we can check for strict

[3] https://github.com/Hofmannl/GraphAndLLMbasedRedundancyAnalysis

Algorithm 1: Graph-based analysis of *strict redundancies*

```
Data: A set of annotated USs
Result: A list containing pairs of USs that are redundant in the goal or
        benefit, respectively.
Translate annotations into JSON scheme;
pairs ← compute all pairs of USs;
goalRedundancies ← ∅ ;
benefitRedundancies ← ∅ ;
foreach (us_1, us_2) ∈ pairs do
    foreach activity ∈ us_1 do
        if activity ∈ us_2 then
            goalRedundancies.add((us_1, us_2));
        end
    end
    foreach reason ∈ us_1 do
        if reason ∈ us_2 then
            benefitRedundancies.add((us_1, us_2));
        end
    end
end
return (goalRedundancies, benefitRedundancy);
```

redundancy by checking whether both stories contain a tuple with identical *src* and *dest* values.

For example, US1 (shown in Fig. 2) contains the pairs [*publish*, *dataset*] in the goal and [*view*, *dataset*] in the benefit and US2 (also shown in Fig. 2) contains the pairs [*publish*, *dataset*] in the goal and [*inspect*, *dataset*] in the benefit. The graph-based approach detects the strict redundancy in the goal but cannot detect the semantic redundancy in the benefit.

Our approach and implementation could easily be extended to provide at least basic support for finding more general forms of redundancies by using a tool like WordNet[4] to analyse individual words for semantic equivalence. This is also suggested (but not implemented) for AQUSA in [12].

5 Redundancy Analysis with Large Language Models

We now present our LLM-based approach to detecting US *redundancies* according to Def. 3. For example, Arulmohan et al. [1], Devlin et al. [6] and Miller et al. [14] have shown that LLMs can easily work with text, e.g., summarising or detecting similarities. Furthermore, the versatility of LLMs to adapt to different tasks through prompt engineering [3,21,22] makes them powerful tools. This motivated us to take advantage of the natural language capabilities of LLMs

[4] https://wordnet.princeton.edu/

to analyse redundancy in USs; we have realised a proof-of-concept implementation[3]. Our implementation relies on the *GPT* models from OpenAI[5], which are prominent state-of-the-art examples of LLMs which are based on the pre-trained transformer model architecture [**?**]. However, our proof-of-concept has been designed to be easily adaptable to other (open) LLMs, such as Mistral[6], Qwen2[7], and LLaMA[8], which offer greater control over data flow, rather than relying solely on OpenAI's proprietary *GPT* models. This flexibility is realised by a modular architecture based on the Strategy Pattern, which allows us to define adapters for each LLM agent that correspond to a common interface.

The workflow for our LLM-based approach consists of the following steps:

1. `Select & Pre-process US data:` The user selects the set of USs to be analysed. Either the textual descriptions or the annotation graphs can be used as an input to the LLM. In addition, all USs that do not conform to the conceptual model introduced in Sec. 3 are reported and ignored.
2. `Build LLM prompts:` This step creates the required prompt message chain (PMC) containing the general context for the analysis and a specific US pair to be analysed. We also provide examples of *redundant* US pairs and a JSON schema to be returned by the LLM; it contains a boolean field to indicate whether a *redundancy* was detected or not, a string field for a *redundancy description* and a tuple for redundant text parts (analogous to Sec. 4).
3. `Handling US pairs with LLMs:` We send the PMC for each US pair to the LLM. This includes processing our prompt sequences and validating the JSON response. For schema violations, a repaired JSON is requested.

An important aspect of using LLMs is the prompt engineering; we follow the research in [3,2,10,17,21,22] and use *Role-Based Prompting* where we set the context and role for the model, *Chain-of-Thoughts* to structure our input from problem to solution to ensure a logical flow, and *Few-Shot Prompting* to feed the agent with input-output examples to guide it to produce a valid output. As an additional strategy, we use *Feedback-Loop-Prompting* to request a new JSON response from the LLM if the JSON schema for the response has been violated. We also rely on *system simulations*. A system simulation is a prompt that is used to specify a pre-defined response from the LLM to a particular request by tagging that prompt with the 'system' role, which helps to guide the agent's behaviour and responses. This mechanism allows us to control how the LLM processes inputs and generates outputs, ensuring that the responses are consistent with our specific goals and requirements. The actual prompts can be found in our repository[3]; their general structure is: $M_{\text{Sequence}} = P_{\text{Context-Framing}} + \sum_{i=1}^{n} P^{i}_{\text{Example}} + P_{\text{Process-Request}} + \sum_{j=1}^{\tau}(P^{j}_{\text{Agent-Answer}} + P^{j}_{\text{Repair-Request}})$ where $n \in \mathbb{N}$ is the number of shots in a few-shot prompt, and $j \in \mathbb{N}$ is the number of repair attempts required to obtain a valid output, constrained by a preset repair

[5] https://platform.openai.com/docs/models
[6] https://mistral.ai/
[7] https://github.com/QwenLM/Qwen2
[8] https://llama.meta.com/

```
{
        "goalRedundancies": {
                "redundant": true,
                "redundancyDescription": "Both Publisher want to publish a
                    dataset",
                "tupleOfActivityRedundancies": [[["publish","dataset"],["
                    publish","dataset"]]], [...]
        },
        "benefitRedundancies": {
                "redundant": true,
                "redundancyDescription": "The datasets shall be visible",
                "tupleOfReasonRedundancies": [[["view","dataset"],["inspect
                    ","dataset"]]], [...]
        }
}
```

Fig. 3. Simplified example for the structure of a potential response of the *GPT* model

threshold τ that must not be exceeded. The '+'-operator indicates concatenation of strings which includes insertion of a system simulation. In addition,

1. $P_{\text{Context-Framing}}$ contains explanations and content for the actor role, the *redundancy* definition (Def. 3), and the output format,
2. P^i_{Example} consists of a US pair example containing an input and its corresponding redundancy output. We use such examples to frame the LLM agent for the specific task. Here, n is the number of examples to insert.
3. $P_{\text{Process-Request}}$ is a prompt instructing the agent to analyse a particular pair of USs and respond with a JSON string, and
4. if an invalid response is generated, the prompt is re-sent with additional prompts, namely the invalid JSON response $P^j_{\text{Agent-Answer}}$ and $P^j_{\text{Repair-Request}}$, a prompt describing why the JSON response does not match the expected schema.

Our implementation can integrate the pair of USs in $P_{\text{Process-Request}}$ either via their annotation graphs (see Fig. 2) or just via their text. In both cases, we expect the same JSON output format containing information about the detected redundancies. The expected format is illustrated in Fig. 3 for the pair of USs from Ex. 1.

6 Evaluation

The goal of our evaluation is to assess the strengths and limitations of both the graph- and LLM-based approaches in order to understand how they can be used effectively. As our definition of *strict redundancy* is clear-cut and the graph-based approach implements it directly, our main focus in this evaluation is to see how well the LLM-based approach finds semantic redundancies. Furthermore, we want to find out whether the identification of strict redundancies by the graph-based approach alone is sufficient or whether semantic redundancy detection with LLMs is also essential. For this, we answer the following research questions (RQs):

RQ1: Is the LLM-based approach suitable for finding redundancies?

RQ1a: How reliably does the LLM-based approach detect semantic redundancies?
RQ1b: To what extent is the set of strict redundancies covered by the redundancies identified through the LLM-based approach?
RQ2: Is the LLM-based approach able to detect redundancies without the help of annotation graphs?
RQ3: How common are semantic redundancies that are not strict?

6.1 Setup and Experiments

We evaluate our approaches using two datasets of USs from different domains. These were extracted from a dataset presented in [1,5]. The first dataset (originally named G19 and containing 137 USs) relates to 'Assistive Technology for Elderly Care' (ATE) and addresses user-centered needs. The second dataset (originally named G22 and containing 83 USs) relates to 'Data Management Plans' (DMPs) and focuses on formal data management. These datasets were chosen for their diversity in scope to assess the effectiveness of our approaches in different use cases. In total, our dataset for the evaluation consists of 12,719 US pairs. Note that 3 ATE and 8 DMP USs were excluded during the preprocessing step because they do not fit the conceptual model (cf. Fig. 1).

We conducted the following experiments: In **Exp. 1**, the graph-based approach is used to compute all *strict redundancies* of the dataset. This establishes a ground truth for them. In **Exp. 2**, the LLM-based approach is used to compute *semantic redundancies* with only the annotation graphs of the USs as input. The annotations should help the LLM to focus on its task (detecting *redundancies*) and minimise the risk of errors due to misinterpretation of activities or reasons. In **Exp. 3**, the LLM-based approach is used to compute *semantic redundancies* using only the text of the USs as input. The aim here is to check to what extent the LLM is able to interpret the activities and reasons of a US on its own, i.e., whether it is able to extract the key information of a US. This could indicate whether it is sufficient to provide only the text of a US as input or whether annotation graphs are useful for redundancy detection. In order to determine the strict redundancies among the US pairs identified as redundant by the LLM-based approaches, we performed an automated check. For *semantic redundancies*, establishing a ground truth for a dataset of this size (12,719 US pairs) is hardly feasible: in the absence of reliable tool support, manual inspection would be the only option. To account for this, we (i) at least manually checked the plausibility of *all* semantic redundancies (goal redundancies: 4.242, benefit redundancies: 666) found in Exp. 2 and Exp. 3 and (ii) manually checked the correctness of the result of a further 1,600 randomly selected US pairs for which no experiment reported a redundancy. Four researchers were assigned to manually verify the US pairs marked as redundant. We organised two working sessions; the first focused on the research context and the presentation of the conceptual model (Fig. 1), while the second focused on discussing the perception of semantic redundancies. Each researcher was assigned to a junk of the data and uncertainties were discussed.

Our experiments were executed on custom hardware with the following specifications: Intel® Core™ i7-8565U CPU @ 1.80GHz (8 CPUs), 8070MB RAM, Windows 11 Home 64-bit. We used *GPT*-4o mini (gpt-4o-mini-2024-07-18) with a temperature of 0.2, OpenAI's default parameters otherwise, 12 threads to process the requests, a repair threshold of 3, and 7 given examples in the PMC. The results of our experiments and in particular the output of the LLM are available on GitHub[3].

6.2 Quantitative Analysis

Table 1 reports the basic data we measured in our experiments. The graph-based approach takes 0.6 seconds compared to roughly 3 hours for the LLM-based approaches. In case of the LLM-based approaches, Exp. 2 (annotation-based) requires more repair steps than Exp. 3 (text-based), namely 472 compared to 384, but fails considerably less often (51 compared to 120, where 'failure' means that no valid JSON output was produced before reaching the repair threshold). By design, the graph-based approach does not need repair steps and cannot fail. Finally, Exp. 2 reports fewer semantic redundancies in the goal of USs than Exp. 3 (1,913 compared to 2,329) but considerably more in the benefit (484 compared to 182).

Table 1. Overview of measured data (Rep.: Number of repair steps; Fails: Number of cases without valid JSON output; Reported redundancies are *strict* in case of Exp. 1 and *semantic* in case of Exp. 2 and 3)

	Basic data			Reported redundancies	
Approach & Results	Runtime	Rep.	Fails	in goal	in benefit
Graph-Based (Exp. 1)	0.6 s	—	—	582	35
LLM-Based (Exp. 2)	185.24 min	472	51	1,913	484
LLM-Based (Exp. 3)	179.84 min	384	120	2,329	182

Next, we analyse the reported redundancies in more detail; an overview can be found in Table 2. Our manual inspection of all reported *semantic redundancies* revealed that in Exp. 2 (annotation-based), 1,750 (goal) and 322 (benefit) of these are correct, which corresponds to a precision of 0.91 and 0.67, resp. For Exp. 3 (text-based), the corresponding numbers are 2,032 correctly identified semantic redundancies in the goal (precision: 0.87) and 158 in the benefit (precision: 0.87). Combining this manual inspection for correctness with an automated comparison of the reported redundancies, we find that Exp. 2 and 3 together detected 2,346 correct semantic redundancies in the goal and 446 in the benefit. Of these, 75 % (goal) and 72 % (benefit) were detected by Exp. 2 and 87 % (goal) and 35 % (benefit) were detected by Exp. 3. Since our further manual inspection of 1,600 pairs of USs that were not reported as redundant in any of the three experiments did not reveal any further redundancies, we assume that 2,346 is close to the true number of semantic redundancies in our dataset and that the proportions just reported are close to the true recall of our approaches.

Table 2. Analysis of the reported semantic redundancies (Rep.: No. of reported redundancies; Corr.: No. of correct redundancies; P: Precision; $\sum$: Total number of correctly detected semantic redundancies together in Exp. 2 and Exp. 3; PSem.: Detected proportion of correct semantic redundancies; Cont: No. of strict redundancies contained in Corr.; PStr.: Detected proportion of strict redundancies)

		semantic					strict	
Type	Exp.	Rep.	Corr.	P	$\sum$	PSem.	Cont.	PStr.
Goal	Exp. 2	1,913	1,750	0.91	2,346	0.75	565	0.97
	Exp. 3	2,329	2,032	0.87		0.87	478	0.82
Benefit	Exp. 2	484	322	0.67	446	0.72	35	1.00
	Exp. 3	182	158	0.87		0.35	23	0.66

Since the graph-based approach is a direct implementation of the definition of *strict redundancy*, we assume that we have found all *strict redundancies* contained in our dataset in Exp. 1 (completeness) and that all redundancies reported there are indeed *strict redundancies* (correctness). Regarding the LLM-based approaches, the (semantic) redundancies reported in Exp. 2 (annotation-based) contained 565 of the 582 strict redundancies in the goal (97 %) and 35 of 35 (100 %) in the benefit. In contrast, Exp. 3 (text-based) returned 478 (87 %) and 23 (66 %) of the strict redundancies in goal and benefit.

Our *key observations* from the quantitative analysis are the following:

1. There are many non-strict redundancies in our dataset (which the graph-based approach, while fast and correct, is unable to detect).
2. The LLM-based approach using annotations requires slightly more repair steps but fails significantly less often than the text-based approach.
3. For the LLM-based approaches, the proportion of detected strict redundancies is always higher than the proportion of detected semantic redundancies, with the exception of the goal redundancies in Exp. 3. This suggests that both LLM-based approaches detect strict redundancy more easily than semantic redundancy.
4. Both LLM-based approaches perform better for the goal than for the benefit of USs, both in terms of precision and proportion of detection. In particular, for the goal, both provide convincing results for both precision and proportion of detection (and our additional manual checks indicate that the proportion of detection might be close to the true recall).
5. The LLM-based approach using annotations detects a comparable proportion of strict redundancies in goal and benefit, and both are higher than the corresponding proportions of the text-based approach. There, additionally, the proportion decreases significantly from goal to benefit.
6. The annotation-based approach has a higher precision but detects a lower proportion of semantic redundancies in the goal compared to the text-based approach and vice versa for the benefit. In addition, the precision of the annotation-based approach decreases significantly from goal to benefit (from

0.91 to 0.67), but the detection rate in the benefit clearly exceeds that of the text-based approach (0.72 to 0.35).

These last observations suggest that the text-based approach has great difficulty in identifying the benefit of a US (resulting in far fewer redundancies being found there). However, it can reliably identify the goal, where it trades precision for recall, compared to the annotation-based approach.

6.3 Discussion

To answer **RQ1**, we first consider **RQ1a**: The results presented above indicate that the LLM-based approach reliably detects *semantic redundancies* with reasonably high precision, especially in the goal. The manual assessment of $1,600$ randomly selected US pairs that were not detected as redundant by any experiment suggests that Exp. 2 and 3 found most of the semantic redundancies contained in the dataset. For **RQ1b**, even though *strict redundancy* is defined without any fuzziness (as we are looking for equality of words), the results show that the LLM-based approaches achieve good results in identifying them with coverages of 0.97 and 0.82 for the goal and 1.00 and 0.66 for the benefit in Exp. 2 and 3. *Overall, the LLM-based approach is well suited for finding redundancies, especially in the goal of a US.*

Regarding **RQ2**, the results presented in Table 2 show that Exp. 3 (text-based) achieves a lower coverage of *strict redundancies*. Regarding redundancies in the goal, the text-based approach trades precision for recall when compared to the annotation-based approach. In particular, it detects more semantic redundancies than the latter. While the differences are not large, *this suggests that the text-based approach may even have advantages in situations where it is important to find all redundancies in the goal of USs.* The text-based approach, however, misses a large portion of the existing redundancies in the benefit of USs, probably by not being able to correctly locating these. *Thus, the annotation-based approach has clear advantages in analysing the benefit of USs.*

To answer **RQ3**: In total, we identified 2,346 US pairs that are semantically redundant in the goal and 446 semantic redundancies in the benefit, which establishes a lower bound on the total number of redundancies contained in the dataset. Therefore, the ratio of strict redundancies is at most 0.25 in the goal and at most 0.08 in the benefit. *This implies that* semantic redundancy *must be considered if the redundancies in a set of USs are to be fully investigated.*

Threats to Validity. We have identified several potential threats to the validity of our approaches. The first threat relates to *internal validity*, as errors in the annotation graphs or their (partial) non-existence affect the graph-based approach, emphasising the need for correctness and completeness of the annotation graph of each US to ensure reliable detection. To counter this, we used a dataset [5] whose annotations graphs have been verified correct [1]. The second threat to validity lies in the inconsistency of LLM models. As Ouyang et al. [16] noted, the

behaviour of LLM models varies between different runs. This variability threatens *reproducibility*. Another threat concerns the *usability and generalisability* of the approaches, since well-formed USs are required as input. However, recent research has shown that poorly formed US data can be automatically improved [11]. Another threat concerns *construct validity*, as we manually checked the correctness of the redundancies identified. Possible differences in the perception of redundancies between the four researchers who carried out this work were mitigated by discussing them in joint working sessions. To mitigate the risk that not all US pairs were checked for redundancy, we checked a large subset of US pairs by randomly selecting the pairs. By doing so, we verified the absence of redundancies in a subset of 1,600 US pairs. To mitigate the threat of *external validity*, we used the datasets of [1,5], which were developed independently of our work.

Qualitative Observations during Evaluation. Finally, we would like to highlight some observations made during the manual evaluation of Exp. 2 and 3. As described in Section 5, the LLM was asked to provide a *redundancy description* in order to gain insight into why the redundancy was detected. When a false redundancy was reported, the descriptions were over-generalised, meaning that most false redundancies were due to the LLM's greater scope for interpretation rather than misidentification of activities and reasons. For example, the description 'In both stories, someone wants to know something' was returned for a non-redundant pair of USs; in the corresponding USs, the personas wanted to know about timetables and software, respectively. However, when a redundancy was correctly identified, the *redundancy descriptions* were mostly appropriate and correct with respect to the detected redundancy. Further investigation of quality aspects and how the *redundancy descriptions* can guide REs and developers is future work.

7 Conclusion

In this paper, we define two forms of redundancy: *strict redundancy*, when two USs contain equal activities or reasons, and *semantic redundancy*, when two USs contain semantically equivalent activities or reasons. Both definitions assume the use of annotated user stories, introduced in [1], to extract and organise the key information of a US. We present two redundancy detectors: A graph-based approach for detecting strict redundancy, which relies solely on annotation graphs, and an LLM-based approach for detecting semantic redundancy. This uses the natural language capabilities of LLMs (GPT in our implementation) to detect semantically equivalent activities and reasons. Since annotation graphs are additional artefacts to be generated, the approach supports (i) annotations only and (ii) US text only as input. Both redundancy detectors have been evaluated on a large dataset published by Dalpiaz [5]. The results show that the graph-based approach achieves significantly faster runtimes and is well suited for detecting strict redundancies. However, at least 75% of the redundancies contained in the

analysed dataset are not strict redundancies, which makes the use of semantic analysis necessary to fully investigate a dataset for redundancies. For both types of input (annotation graphs and US text only), the LLM-based approach achieves high precision. A manual inspection of 1,600 US pairs that were not detected by the LLM-based approach indicates that we have found most of the redundancies contained in the dataset. Although we got slightly better results when using annotation graphs for the LLM-based approach, it is also works well for redundancy detection when annotation graphs are not available and cannot be generated. Overall, our results indicate a great potential for the use of LLMs in the quality assurance of USs, especially for semantic redundancy detection. Compared to traditional similarity checkers based on natural language processing, LLMs can structurally separate goal and benefit of the US text by themselves and provide insight into why a redundancy occurs by generating a redundancy description.

After finding redundancies between USs, a recommendation engine would be helpful to resolve redundant USs, e.g., by merging redundant USs and to create a new US using LLMs. In the future, we plan to further explore how LLMs can be used to automate the analysis and improvement of USs also for further quality aspects, such as minimality, conflict-freeness and independence [12]. The LLM-based quality assurance will then act as an oracle to assist requirement engineers in analysing and improving the quality of user stories in agile development.

References

1. Arulmohan, S., Meurs, J., Mosser, S.: Extracting domain models from textual requirements in the era of large language models. In: MODELS 2023 Companion. pp. 580–587. IEEE (2023). https://doi.org/10.1109/MODELS-C59198.2023.00096
2. van Berkel, N., Skov, M.B., Kjeldskov, J.: Human-AI interaction: intermittent, continuous, and proactive. Interactions **28**(6), 67–71 (2021). https://doi.org/10.1145/3486941
3. Brown, T.B., et al.: Language models are few-shot learners. In: Larochelle, H., et al. (eds.) NeurIPS 2020 (2020), https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html
4. Cohn, M.: User Stories Applied: For Agile Software Development. Addison-Wesley signature series, Addison-Wesley (2004)
5. Dalpiaz, F.: Requirements data sets (user stories) (2018). https://doi.org/10.17632/7zbk8zsd8y.1
6. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) NAACL-HLT 2019. pp. 4171–4186. Association for Computational Linguistics (2019). https://doi.org/10.18653/V1/N19-1423
7. Duszkiewicz, A.G., et al.: On Identifying Similar User Stories to Support Agile Estimation based on Historical Data. In: Bera, P., Dalpiaz, F., Wautelet, Y. (eds.) Agil-ISE 2022. pp. 21–26. CEUR-WS.org (2022), https://ceur-ws.org/Vol-3134/paper-4.pdf
8. Heck, P., Zaidman, A.: A quality framework for agile requirements: A practitioner's perspective. CoRR (2014), http://arxiv.org/abs/1406.4692

9. IEEE: IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1993 (1994). https://doi.org/10.1109/IEEESTD.1994.121431
10. Kong, A., et al.: Better Zero-Shot Reasoning with Role-Play Prompting. In: Duh, K., Gómez-Adorno, H., Bethard, S. (eds.) NAACL 2024. pp. 4099–4113. Association for Computational Linguistics (2024). https://doi.org/10.18653/V1/2024.NAACL-LONG.228
11. Lucassen, G., Dalpiaz, F., van der Werf, J., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requirements engineering **21**, 383–403 (2016). https://doi.org/10.1007/S00766-016-0250-X
12. Lucassen, G., Dalpiaz, F., van der Werf, J.M., Brinkkemper, S.: Forging high-quality User Stories: Towards a discipline for Agile Requirements. In: Zowghi, D., Gervasi, V., Amyot, D. (eds.) RE 2015. pp. 126–135. IEEE (2015). https://doi.org/10.1109/RE.2015.7320415
13. Lucassen, G., Dalpiaz, F., van der Werf, J.M., Brinkkemper, S.: The Use and Effectiveness of User Stories in Practice. In: Daneva, M., Pastor, O. (eds.) REFSQ 2016. pp. 205–222. Springer (2016). https://doi.org/10.1007/978-3-319-30282-9_14
14. Miller, D.: Leveraging BERT for Extractive Text Summarization on Lectures. CoRR (2019), http://arxiv.org/abs/1906.04165
15. Mosser, S., Pulgar, C., Reinhar, V.: Modelling agile backlogs as composable artifacts to support developers and product owners. J. Object Technol. **21**(3), 3:1–15 (2022). https://doi.org/10.5381/JOT.2022.21.3.A3
16. Ouyang, S., Zhang, J., Harman, M., Wang, M.: LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. CoRR (2023). https://doi.org/10.48550/ARXIV.2308.02828
17. Pan, A., Jones, E., Jagadeesan, M., Steinhardt, J.: Feedback Loops With Language Models Drive In-Context Reward Hacking. In: ICML 2024. OpenReview.net (2024), https://openreview.net/forum?id=EvHWlYTLWe
18. Ronanki, K., Daniel, B.C., Berger, C.: ChatGPT as a Tool for User Story Quality Evaluation: Trustworthy Out of the Box? In: Kruchten, P., Gregory, P. (eds.) XP 2022 Workshops. pp. 173–181. Springer (2023). https://doi.org/10.1007/978-3-031-48550-3_17
19. Wang, X., Zhao, L., Wang, Y., Sun, J.: The Role of Requirements Engineering Practices in Agile Development: An Empirical Study. In: Zowghi, D., Jin, Z. (eds.) APRES 2014. pp. 195–209. Springer (2014). https://doi.org/10.1007/978-3-662-43610-3_15
20. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I.: Unifying and Extending User Story Models. In: Jarke, M., et al. (eds.) CAiSE 2014. pp. 211–225. Springer (2014). https://doi.org/10.1007/978-3-319-07881-6_15
21. Wei, J., et al.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: Koyejo, S., et al. (eds.) NeurIPS 2022 (2022), http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html
22. White, J., et al.: A prompt pattern catalog to enhance prompt engineering with ChatGPT. CoRR (2023). https://doi.org/10.48550/ARXIV.2302.11382
23. Zhang, Z., et al.: LLM-Based Agents for Automating the Enhancement of User Story Quality: An Early Report. In: Smite, D., et al. (eds.) XP 2024. pp. 117–126. Springer (2024). https://doi.org/10.1007/978-3-031-61154-4_8