Spontaneous Re-documentation of Use Cases over a Naturally Maturing Project in an Agile Context

Alexandra Skyvová, Ján Lang

Faculty of Informatics and Information Technologies, Slovak university of technology in Bratislava, Ilkovičova 2, Bratislava, 842 16, Slovak republic, alexandra.skyvova@stuba.sk; jan.lang@stuba.sk

Abstract: The development of a brand new system is often a solution in case of insufficient or missing documentation or the very specification of requirements for an existing but already technologically or morally outdated system. An Ex-Post approach to automating the process of extracting the working logic from the existing system is not always easy. Even targeted Ex-Ante maintenance of still relevant artifacts supporting software development is non-trivial as time passes. The problem is not only the time factor, agile practices and intentional focus on functionality over documentation but also the amount of accompanying documentation itself. Updating the relevant parts of the documentation itself is a difficult problem, given the requirement of operational changes and significant flexibility to focus attention only on the most important parts of the software. Therefore, one of the appropriate solutions appears to be the support of ongoing but formal re-documentation of important artifacts stimulating development. In this article, we present an approach that activates the spontaneous re-documentation of use cases over a naturally maturing project in an agile context, in a non-invasive way, as an accompanying phenomenon of the scrum software development framework, already during the development process itself.

Keywords: Agile scrum artifacts; Software development documentation; Sprint backlog tasks; Use cases re-documentation, User stories

1 Introduction

One of the obstacles is that agile developers perceive a lack of documentation in their work [1, 2]. Although developers in agile software development prefer functionality over documentation (according to Agile manifesto[1]), documentation of software would be beneficial when new members on board to the team [3, 4], when an agile team is communicating with a non-agile team (e.g. waterfall team)[5], during the pre-planning phase to better estimate work (effort estimation) [6], when

[1] https://agilemanifesto.org/

adopting legacy software [7, 8], for a project handover [9], to deliver software on-time [10], when developing software of a special kind, for which documentation is required by law and standards, e.g. safety-critical systems [11].

Documentation in the form of use cases is important as well, as confirmed by several works [12, 13, 14]. Although agile developers do not have the time to produce documentation [1, 15], it is beneficial for them to have use cases available [16, 17, 18]. Nowadays teams use Project Management Systems (PMSs) such as Jira[2] to organize their projects. Jira presents an opportunity to produce documentation because it contains many different artifacts that would have once been stored separately. For example, in Jira, teams can organise epics, user stories, and tasks. These used to be written on individual cards made of paper, but now, they are written online and can be easily analyzed and automated. Based on this, we address this research question: Is it possible to generate use cases from the artifacts recognised in Scrum? Our research objective is to find a way to re-document use cases in a spontaneous way that does not put more workload on agile teams. The major contributions of this article are:

- A fully automated approach to generate use cases from user stories and tasks. The approach was implemented as a proof-of-concept tool that processes user stories and tasks in Jira and outputs use cases to Confluence[3] using natural language processing methods. Insofar, a fully automated approach had not been developed. As described in Section 2, in the research area of use case scenario generation from agile artifacts, the most automated approach was only semi-automated.

We assume that the use case specification is always useful to those involved because the specification is comprehensive, easy to understand, and exhaustive. The aim is to generate use cases based on Alistair Cockburn's notation [19]. We focus on the main success scenario, primary actors and use case title. Use case dependencies such as extend and include relationships are out of scope.

The remainder of this paper is organised as follows. Section 2 describes related work. Section 3 describes agile requirements. Section 4 describes how use cases are re-documented. Section 5 contains an evaluation of use cases re-documentation. Section 6 describes the absence of chronological continuity of the use cases. Section 7 evaluates the ordering of use case scenarios with the sequence diagrams. Section 8 contains a discussion and concludes the paper. Finally, Section 9 outlines the threats to validity and discusses limitations of the study.

[2] https://www.atlassian.com/software/jira

[3] https://www.atlassian.com/software/confluence

2 Related Work

In this section, we describe works that aim to generate use cases from user requirements.

Maatuk et al. [20] provide a set of rules based on NLP to transform natural language requirements into use case diagrams and activity diagrams. It is an extension of their previous work, in which they used NLP techniques to generate UML class diagrams. In this work, the authors did not provide a prototype for generating use cases and activity diagrams. According to their results, "the diagrams generated by their approach were significantly better than those generated by the other approach when compared to other existing approaches that have been presented in the literature." However, they made the comparison based on the functionalities of each approach—whether they can generate class, use case, and activity diagrams. They do not mention evaluating and comparing the accuracy or correctness of generated diagrams.

Kochbati et al. [21] provided a prototype to generate UML use case models from user stories. The authors cluster related user stories using a machine-learning-based approach to group together user stories that belong in a use case diagram. Specifically, they chose a hierarchical clustering method—the HAC algorithm. For natural language processing, the spaCy library was used to extract relevant elements from each user story. To evaluate their approach, they conducted three case studies and presented key performance indicators and evaluation metrics. They measured how much the identified clustering solution is accurate, identified the accuracy of the use case models generated from each cluster, and checked the execution time to establish whether the approach runs within a reasonable time in realistic settings.

Tiwari et al. [22] took as input a problem description and functional requirements to generate the textual form of a use case—actors, dependencies, pre-condition, post-condition, and basic flow. The approach uses a Natural Language (NL) parser to identify parts-of-speech tags, type dependencies, and semantic role labeling from the input text specification to populate use case elements. It further makes use of a questionnaire-based approach (who, what, when, where, why, how, and how much) to develop the remaining unpopulated parts of the use case template. The method was tested through both qualitative and quantitative evaluation.

Textual elements of use cases, namely scenarios, are a vital part of use cases. Tiwari et al. generated scenarios from requirements in natural language via NLP techniques and their own defined heuristics. We, however, generate use case scenarios from tasks corresponding to individual user stories. Another difference is that Tiwari et al. employ a questionnaire-based approach to generate use cases, meaning the requirements in natural language are first processed and then the information is checked by related questions in yes/no or open answer format. From the demo provided, which included 7 use cases, we reckon that for the requirements nearly 50 questions were generated for the developers to answer. Such an approach can

become easily time-consuming, assuming that to generate, for example, 15 use cases, 100 questions have to be answered, with more and more questions added for each use case.

Gilson et al. [23] attempt to generate robustness diagrams from user stories in order to support understanding of the scope of user stories in a backlog. The proposed technique allows combining multiple user stories to generate a diagram. Since such a diagram would become incomprehensible with the increasing number of user stories, they implemented viewpoint-based diagram generation—a diagram that shows relationships between actors, domain entities, and user interfaces starting from a selected diagram element. The technique uses natural language processing and rule-based transformations. The prototype was evaluated on more than 1,400 real-world user stories from 22 backlogs. The technique created 81.4% syntactically valid diagrams for individual stories, and 50% of the diagrams were semantically equivalent to those manually created.

Vasques et al. [24] take business vision in natural language as input and generate a table of syntagmas (verb, who, what, etc.) which then serves as input to a concept map (high-level use case diagram). To process the input text, the Verbka [25] process was used. Verbka first decodes the original text by fragmenting it into minor blocks of information (or syntagmas) according to its linguistic (syntax/semantics) categorization. A syntagma is a sequence of words that have a dependency relationship between them. Based on these syntagmas, a concept map is generated. The accuracy of generated concept maps or overall results were not evaluated.

Elallaoui et al. [26] developed a plugin based on Natural Language Processing (NLP) techniques for the automatic transformation of user stories into UML use case diagrams. The approach was evaluated by comparing the number of diagram items detected manually and those detected automatically. Positive results have already been achieved for "actors," which represent 98% accuracy and recall. For use cases and their relationships, 87% accuracy and 85% recall were achieved.

Table 1 compares mentioned works in terms of the processed input and generated output. Out of the available papers aiming to extract use cases from user requirements, no other work attempted to generate use case scenarios. We are aware of the work of Tiwari et al. [22] that did attempt to provide use case scenarios; however, it is excluded from the table as it is not a fully automated approach since it requires manual answering of the questionnaire.

Every approach from related work that takes user stories as input extracts the use case title from the <goal> part of the user story template. None of them explains why such an equation would be correct though. Can we equate a goal to a use case? Does the actor's goal equate to functionality? In the next section, we look into how agile requirements are related to use cases.

Table 1
A comparison table of related works

Reference	Input	Output	Provided use case scenarios
Maatuk et. al. [20]	natural lang. requirements	use-case and activity diagrams	No
Kochbati et. al. [21]	user stories	UML use case models	No
Gilson etl. al. [23]	user stories	visual robustness diagrams - structural and behavioural information	No
Elallaoui et. al. [26]	user stories	UML use case diagram	No
Vasques et. al. [24]	business vision in natural language	concept map	No
Our approach	user stories and tasks	textual form of use cases	**Yes**

3 Mapping User Stories to Use Cases

In this section, agile requirements and use cases are defined. “An epic is a work item, too large to be completed by an agile team in one iteration and therefore has to be divided into several user stories” [27]. For the sake of explaining the context, we will use an example inspired by a use case of work by Abrahao et al. [28] throughout this work. Following is an example of an epic:

- As a user, I want to have a good online retail experience.

User stories are the most common form of requirements in agile software development [29]. A user story is “a structured natural language description of requirements. It follows a compact template that describes the type of user, what they want and (optionally) why” [30]. Example of a user story for the aforementioned epic:

- As a user, I want to order one or more products.

"A use case expresses a contract between the stakeholders of a system about its behaviour" [19]. What are the minimal properties a use case should have for it to be considered a use case at all? There are several notations of use cases and they all differ in style and purpose [13]. Alistair Cockburn's notation [19] distinguishes between two types of use cases: "a casual use case and a fully dressed use case." A casual use case shows the interaction between stakeholders in an informal language without numbered steps. A fully dressed use case is much more complex and contains elements such as the primary actor, main success scenario, extensions, etc.

In Scrum, requirements are collected in the form of user stories from stakeholders. These stories are comprehensible both to the development team and the stakeholders. However, user stories do not capture enough detail for developers to know how functionality should be implemented. To solve this issue, Scrum teams split user stories into several smaller tasks during the Sprint planning.

"Tasks play a key role in agile development, for they bridge the problem space (the requirements) and the solution space (the architecture and the code). The tasks refine the product requirements expressed as user stories" [31]. According to Müter et al. [32], "a task is expressed by a verb, followed by one or more follow elements, each being either a noun, a conjunction, an adjective, a 'to', or a cardinal number. Task labels describe an action for the developer to carry out to implement a part of a software function, or to improve existing code." An example of some tasks for the above-mentioned user story could look like this:

- Design products page.
- Code products page.
- Create database design for products.
- Code back-end functions for ordering products.
- Integrate front-end and back-end.

Several works describe the relationship between agile artifacts and use cases. Madanayake et al. [33] compared user stories to use cases: "the role in a user story is similar to an actor of a use case model, while the goal or desire in a user story is similar to a use case." However, the authors also suggested that "even if user stories may not be directly compatible with use cases, epics which are larger and broader in scope may be so."

Santos et al. [34] consider a user story to be smaller than a use case. "Additionally, it is discussed that a use case contains a set of interrelated user stories. For that reason, in our approach, this rule suggests that an epic is directly derived from a use case."

Jacobson et al. [35] present use case slices, which are carefully selected parts of a use case. "A use-case slice is one or more stories selected from a use case to form a work item that is of clear value to the customer." Please note that stories, in this

case, are not user stories, but "a part of the use-case narrative, one or more flows and special requirements, and one or more test cases." Then they compare these use case slices to user stories. "Use-case slices and user stories share many common characteristics. They both define slices of the functionality that teams can get done in a sprint and can be sliced up if they are too large, resulting in more, smaller items."

The overall sentiment seems to indicate that user stories do not provide enough functionality to be considered use cases, while epics do. What we may have here are two competing concepts: functionality for a user and deliverability in a single sprint. Consider that epics provide the functionality to a user but are too large to fit into one sprint. User stories can fit into one sprint and should deliver functionality to a user too (according to agile principles). Both epics and user stories can therefore be mapped into use cases, however, with different goal levels defined in Cockburn's notation [19]. The notation includes three levels of goals: summary level, user-goals level, and sub-functions level. Table 2 shows example use cases derived from the above-mentioned epic and user story and an example sub-function with their corresponding goal levels. Cockburn [19] recommends writing use cases that meet the user-goal level. Therefore, in this work, we focus on re-documenting use cases on the user-goal level derived from user stories. User stories should be large enough to be split into several tasks and to capture functionality.

Table 2
A table of use cases and their goal levels

Use case	Goal level
Online retail experience	Summary goal
Order one or more products	User-goal
Identify product	Sub-function

4 Re-documentation of Use Cases from Tasks Proposal

This section proposes a user story splitting method and describes details of implementation for use cases re-documentation from user stories and tasks.

There are several methodologies defined for agile development, and it is a common practice for teams to use them. Development teams usually do not strictly follow all the practices of the selected methodology but choose the subset of agile practices suitable for the specific project [36]. On a similar note, if an agile team chooses to

implement a method to their process, it does not mean that they have lost their agility [37].

The splitting of user stories is a key component of agile software development and is essential for breaking down complex user stories into smaller, manageable tasks that can be completed by the development team. Dells´en et al. [38] provide insights into how user story splitting is currently practised in the industry. In practice, there are many different ways teams that can split user stories, e.g. splitting by feature, functionality, expertise, etc. [38]

We propose a new user story splitting method, which is implemented by taking user stories and their corresponding tasks from Jira as input and generates textual specifications of use cases into Confluence. If agile teams split user stories into tasks using our approach, we could reverse engineer use cases from them. As Cohn [39] indicated, user stories should be written in a template: "As a << type of user >>, I want << goal >>[, so that << benefit >>]".

Most often, user stories are split into tasks vertically [38] - tasks contain work on all software layers such as the user interface, the API, and the database. The user story mentioned in Section 3, "As a user, I want to order one or more products", could be split vertically into the following tasks:

- Design products page.
- Code products page.
- Create database design for products.
- Code back-end functions for ordering products.
- Integrate front-end and back-end.

We propose splitting a user story in a way that each task represents a brief one-line summary of what the system does (or is supposed to do) in English. Think of it as tasks you would give to a system. Tasks should be in an imperative mood, starting with an active verb, which is the most common structure of tasks [32]. When user input is expected, the tasks should contain the verb 'request'. E.g. task "Request payment for the order." indicates that the system expects input from a user. According to the template, user stories have to contain all commas, otherwise, the generated use cases might have unexpected results.

For the same user story "As a user, I want to order one or more products" tasks could be produced in the following matter:

- Request information about the product/s related to the order created
- Create the order and notify User
- Request payment for the order
- Start the delivery process of the products
- Notify that the payment has been correctly accepted

We assume that each user story could correspond to at least one use case. A use case expresses an interaction between the system and the user. Based on the assumption that tasks represent what the system does, then each step in a scenario is the system's ability to execute something. We could at least partially reconstruct the use case scenario:

1. *Unknown user step.*
2. System requests information about the product/s related to the order created.
3. *Unknown user step.*
4. System creates the order and notifies User.
5. System requests payment for the order.
6. *Unknown user step.*
7. System starts the delivery process of the products.
8. System notifies that the payment has been correctly accepted.

To generate use case scenarios from tasks of a user story, these steps are followed: In **step 1**, all tasks related to a user story are gathered. **Step 2**: Each task is written to a separate line and empty lines are added - an empty line for the first step and an empty line after each system step. **Step 3**: The first line of the use case reflects the user's selection. Thus, we construct the first line by saying "User selects to" <use case name >. For each line in which the system requests something ("system requests"), the user provides what the system requested. The word "provide" ("user provides") is deliberately chosen because we do not yet know in what format the user will provide the information - text input, checkbox. We delete leftover blank lines because we assume that the user no longer interacts with the system and thus only system actions remain. Listing 1 presents the pseudocode summarizing these steps for generating use case scenarios from tasks of a user story.

```
FUNCTION                          GetUCTitle(user_story)
     goal      ←      user_story.splitAfter(@hasComma)
     goal     ←     user_story.splitBefore(@hasComma)
     goal.match("I         want         to").remove()
     RETURN                                      goal
END                                          FUNCTION


FUNCTION                          GetUCActor(user_story)
      role     ←     user_story.splitBefore(@hasComma)

     IF          role.has("As         a")         THEN
           role.match("As                a").remove()
```

```
     ELSE      IF      role.has("As      an")      THEN
           role.match("As                an").remove()
     END                                              IF

     RETURN                                         role
END                                               FUNCTION


FUNCTION        GetUCScenario(user_story,        tasks)
     role           ←           GetUCActor(user_story)
     goal           ←           GetUCTitle(user_story)

     first_step  ←  role  +  "  selects  to  "  +  goal

     all_steps                  ←                  ""
     all_steps              +=              first_step

     FOR       EACH       d       IN       tasks       DO
           system_step              ←               d
           system_step.insertBefore("System         ")
           system_step.matchFirst(#verb).conjugate()
           all_steps          +=          system_step

           IF  d.matchFirst(#verb)  =  "requests"  THEN
                 user_response           ←           d
                 user_response.insertBefore(role
                 +        "        provides        ")
                 all_steps      +=      user_response
           END                                      IF
     END                                           FOR

     scenario      ←      all_steps.addLineNumbering()
     RETURN                                  scenario
END                                          FUNCTION

documentation                  ←                 UC[]

FOR    EACH    user_story    IN    user_stories    DO
     tasks           ←           GetTasks(user_story)

     UC                        ←                       {
           title:              GetUCTitle(user_story),
           primary_actor:      GetUCActor(user_story),
           scenario:  GetUCScenario(user_story,  tasks)
```

```
        }

      documentation.append(UC)
END FOR
```

Listing 1: Pseudocode for generating use cases from user stories and their tasks

By applying the above-described approach, the final use case scenario could be generated:

1. User selects to order one or more products.
2. System requests information about the product/s related to the order created.
3. User provides information about the product/s related to the order created.
4. System creates the order and notifies User.
5. System requests payment for the order.
6. User provides payment for the order.
7. System starts the delivery process of the products.
8. System notifies that the payment has been correctly accepted.

From the user story template: "As a << type of user >>, I want << goal >>[, so that << benefit >>]" we need to extract the part << type of user >> which would become the actor and << goal >> which would represent the use case title. From the related work section, we reckon there are generally two approaches to extracting said parts - heuristic rules based on part-of-speech tagging and relying on the template to remove unnecessary parts, leaving us with a type of user and goal. By analysing related works and by capabilities of NLP libraries, we chose to extract part of use cases by extracting chunks, similarly to work by Kochbati et. al. [21]. This approach has a disadvantage though that the user story template has to match completely, commas included. With a missing comma, the algorithm will return incorrect results. People tend to make mistakes when writing, so it must be taken into account. However, it can be mitigated by implementing a template for user stories in Jira, which would help developers to comply with the user story template.

The user story is split into two parts based on the comma placement in the user story template. The first part called Actor would be "As a << type of user >>, " and the second part called Goal would be "I want << goal >>". The third part of the user story - the benefit is omitted, because we do not need it to generate use cases.

The implementation called STUART (uSer-sTory Use-case) is a Confluence macro that takes user stories and tasks as an input and outputs textual specification of use cases. User stories and their corresponding tasks are from Jira REST API available for Forge[4] app. In the Forge app, user stories and tasks are processed and

[4] https://developer.atlassian.com/platform/forge/

transformed to use cases via NLP techniques. Processing of text requires tokenization, part-of-speech tagging and dependency parsing. NLP library is chosen in JavaScript since that is the programming language of the Forge app. Once the use cases are generated, they are sent to Confluence as documentation.

5 Use Cases Re-Documentation Evaluation

This section provides details about the evaluation of the proposed user story-splitting method. The method was evaluated by a qualitative approach, by conducting two interviews and by auto-evaluation of generated use cases.

First, we assess whether the generated use cases could be indeed considered use cases. The minimal properties of a use case can be found in the description of a casual use case in Cockburn's notation [19]. The casual use case contains a use case title, at least one primary actor, and interaction between actors, as can be seen in Figure 1. The resulting generated use cases contain all three, the actors and the use case titles were successfully extracted from user stories, and the interaction between two actors was obtained by transforming tasks into the main success scenario. Figure 2 shows a generated use case. We self-assessed this methodology on three user stories, which we split into tasks using our new method to produce the use cases.

Secondly, we evaluate if the generated use case is also considered 'correct' use case - the use case does indeed represent the interaction between user and system to order products in an e-shop environment. Seven software engineering students were asked to write use case scenarios for functionality 'Order one or more products'. The resulting use cases were then compared with the generated use case. As anticipated, the written use cases are similar to the generated; all of them contain steps related to product selection and payment. In fact, the average semantic similarity between the generated use case and each individual use case written by the participants was 0,7514 using model Paraphrase Multilingual Mpnet Base v2 provided by nlpcloud[5].

[5] https://nlpcloud.com/nlp-semantic-similarity-api.html

Use case: Buy something
The Requestor initiates a request and sends it to her or his Approver. The Approver checks that there is money in the budget, check the price of the goods, completes the request for submission, and sends it to the Buyer. The Buyer checks the contents of storage, finding best vendor for goods. Authorizer: validate Approver's signature. Buyer: complete request for ordering, initiate PO with Vendor. Vendor: deliver goods to Receiving, get receipt for delivery (out of scope of system under design). Receiver: register delivery, send goods to Requestor. Requestor: mark request delivered. At any time prior to receiving goods, Requestor can change or cancel the request. Canceling it removes it from any active processing. (delete from system?)Reducing the price leaves it intact in process. Raising the price sends it back to Approver.

Figure 1
A casual use case by Cockburn [19]

Use case title: Order one or more products
Primary actor: User
Main success scenario:

1. User selects to order one or more products.
2. System requests information about the product/s related to the order created.
3. User provides information about the product/s related to the order created.
4. System creates the order and notifies User.
5. System requests payment for the order.
6. User provides payment for the order.
7. System starts the delivery process of the products.
8. System notifies that the payment has been correctly accepted.

Figure 2
Generated use case

Third, an interview was conducted with 27 software engineering students who were divided into six teams of different sizes. All participants were pursuing a master's degree and possessed adequate knowledge of agile user story splitting. The teams develop software as a project for the University as a part of the class assignment. Each team was shown how to split the user stories into tasks as described in Section 4. The study was conducted in accordance with institutional policy for classroom-based research activities; therefore, formal IRB approval was not required, and participation was voluntary.

The participants were asked two questions: Would you be able to split the user stories in the proposed method and would you recognise any obstacles that would prevent you from using it? All teams, but one, agreed that they would be able to use the method. The last team pointed to an obstacle; they deem using this method too time-consuming. We attribute this obstacle to the novelty effect. In addition, the participants appreciated that they do not have to recall how to write use cases since

they no longer remember how use cases should be written. For the second part of the interview, participants were asked to produce tasks for a set of user stories to receive use cases. The participants first split the user stories as they normally would, then using our defined method. We generated use cases from created tasks and compared the results. The goal is to evaluate whether the participants are able to use the method correctly.

When participants split user stories without using our method, the resulting main success scenarios did not make much sense. In contrast, when they used the method, the generated use case scenarios were comprehensible. Each of them was able to use the method successfully. Although the final use cases were comprehensible, the quality of the use cases varied according to the effort individuals put into creating the tasks.

Based on the results, we conclude that it is possible to learn using our method to split user stories into tasks and successfully generate use cases from them. The most important moment of this approach is splitting user stories into tasks; the rest are automated.

6 The Absence of the Chronological Continuity of the Use Case Scenario

This section deals with solving the problem of unordered steps in the use case scenario. The scenario of the use case ”Order one or more products” described in Section 4 was merely a happy-day scenario when tasks are received in the order of system execution. To remedy this, we explore the use of UML sequence diagrams to order the scenario steps. Sequence diagrams can be obtained from existing tools that generate sequence diagrams from the available source code [40].

Let us have an example of a group of tasks inspired by a use case of work by Abrahao et al. [28] in random order from a user story ”As a user, I want to order one or more products”:

- Request payment for the order.
- Start the delivery process of the products.
- Request information about the product/s related to the order created.
- Notify that the payment has been correctly accepted.
- Create the order and notify User.

By generating a use case from these unordered tasks using the approach described in Section 4, we would get the following use case scenario:

1. User selects the functionality “Processing an Order”

- System requests payment for the order.
- User provides payment for the order.
- System starts the delivery process of the products.
- System requests information about the product/s related to the order created.
- User provides (inserts) information about the product/s related to the order created.
- System notifies that the payment has been correctly accepted.
- System creates the order and notifies User.

The first step is ordered because that is how every use case scenario starts, however, the rest of the steps are unordered and do not create a cohesive flow. If we extracted messages from the corresponding sequence diagram 5 we should be able to map those to use case steps, which would order the mentioned steps. The detail of the outcome is depicted in Figure 4. Figure 4 contains three columns: the leftmost column consists of unordered steps for a use case 'Processing an Order'. The column in the middle consists of messages from the sequence diagram in Figure 3 for the same use case. The rightmost column shows the resulting use case scenario after unordered steps were ordered by messages from the sequence diagram.

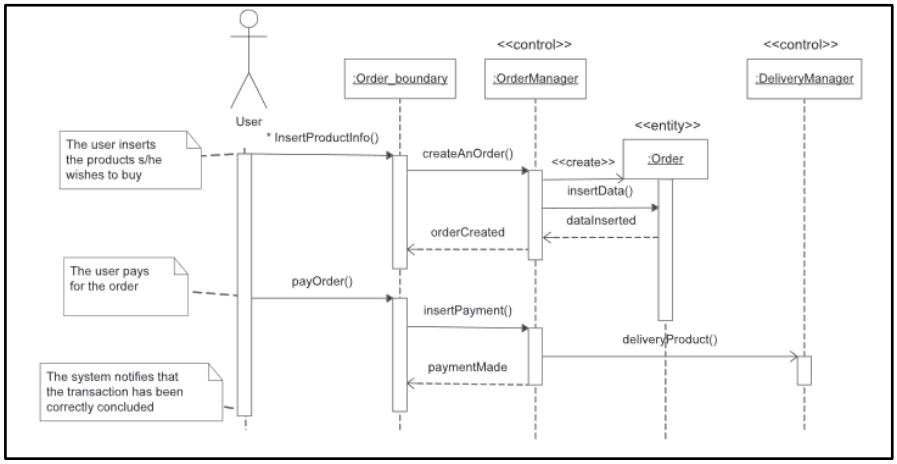

Figure 3
Sequence diagram for use case 'Processing an Order' by Abrahao et al. [28]

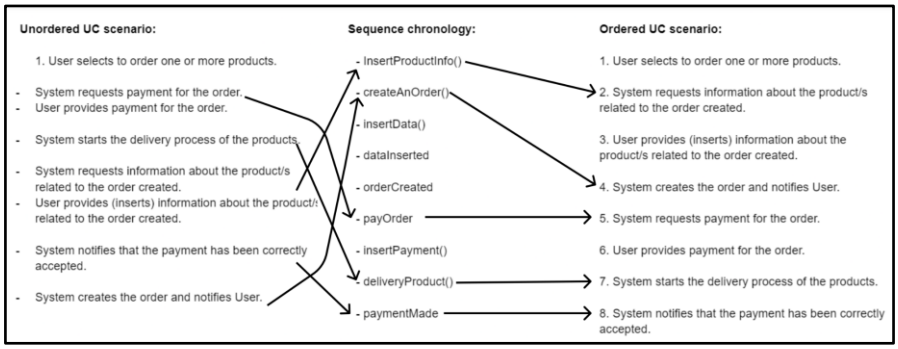

Figure 4
Ordering of use case scenario by using chronology from sequence diagram

7 Ordering Use Case Scenario with Sequence Diagrams Evaluation

This section describes the evaluation of using Large Language Models (LLMs) to order the use case scenario steps with sequence diagrams to provide the chronological continuity of use case scenarios.

For the evaluation, three student projects were used. AntiQ is a student project made with the for e-auction of antiques, it has 3 use cases. ”Get instrumental Again” is a project made for the e-auction of musical instruments and has 1 use case. Lastly, USPF is a simple accounting project for processing invoices and has 3 use cases. Each project is of small size; we measure each project’s size in tokens. There are different ways of counting the number of tokens in a text, but the simplest is to split the text into tokens based on whitespaces. Table 3 summarises the number of tokens for each project, the first column from the left to right describes how many tokens each project has in its source code, the second column describes the number of tokens in sequence diagrams represented in XML format, and the last column denotes the number of tokens in sequence diagrams represented in PlantUML format in a text file.

To evaluate the output, LLMs output is compared to the source of truth, the ordered use case scenarios of the system written by their developers beforehand. To calculate the accuracy of the ordering, the order provided by LLM is compared with the correct order, such as:

Correct order: 1, 2, 3, 4

LLM order: 3, 1, 2, 4

Comparison step by step:

1. Step 1: In LLM order, it is in position 2. Score = 0
2. Step 2: In LLM order, it is in position 3. Score = 0
3. Step 3: In LLM order, it is in position 1. Score = 0
4. Step 4: In LLM order, it is in position 4. Score = 1

Total score = 0 + 0 + 0 + 1 = 1

Total number of steps = 4

$$\texttt{Accuracy} = (\texttt{Total score}/\texttt{Total number of steps}) * 100$$

$$\texttt{Accuracy} = (1/4) * 100 = 25\%$$

First, we look at how well LLM is capable of performing when tasked with ordering use case scenario steps based on the sequence diagram of the system. The LLM receives the unordered steps in random order and sequence diagrams in textual form (plantUML notation) to order the mentioned scenario steps. Then, we evaluate the results by comparing them with ordering scenario steps using source code and without additional information. The LLM receives the unordered steps and source code to order the mentioned scenario steps. Lastly, the LLM receives a list of unnumbered steps of a scenario and is tasked to put them in the correct (logical) order. Additionally, 7 participants were gathered to perform the task given to LLM, to logically put the unordered steps into the correct order, to calculate how well humans are able to solve the task, and to compare that with the LLM accuracy. The participants received the same unordered steps of scenarios that the LLMs received, without prior knowledge of the system. Every participant was a software engineering student who had an adequate understanding of use cases.

Table 3
Summary of the number of tokens for each project repository

Repository	Code tokens	XML tokens	PlantUML tokens
antiQ	723	6 362	2 181
Get instrumental Again	1 235	3 519	1 201
USPF	2 717	40 110	14 315

Table 4 compares the accuracy of two models: gpt-3.5-turbo and gpt-4-turbo, and participants. The values are rounded to two decimal places. The leftmost column contains the accuracy when LLM and participants are tasked with ordering the UC steps without any further information. The next column describes how accurate LLM was when tasked with ordering the steps based on sequence diagrams.

The last column describes how accurate LLM was when tasked with ordering the steps based on the source code. The results show that gpt-4-turbo performs better than gpt-3.5-turbo on ordering the steps from unordered steps only, and ordering steps with sequence diagrams. Gpt-3.5-turbo performs better when ordering the steps using the source code. Overall, the best result (accuracy 84,38%) was produced by gpt-4-turbo when ordering the steps without additional information. Although the participants performed better than the gpt-3.5-turbo, the gpt-4-turbo was more accurate.

Table 4
Comparison of accuracy based on the artifact used to order the scenario steps

Method	Baseline accuracy	Accuracy with SD	Accuracy with source code
gpt-3.5-turbo	56,25 %	34,36 %	59,36 %
gpt-4-turbo	84,38 %	78,13 %	56,25 %
participants	71,43 %	NA	NA

8 Discussion and Conclusion

This method was evaluated on data either manually created or acquired from students' projects. Real data from agile companies could help to analyse the current methods of user story splitting and to evaluate our method. However, open-source Jira repositories or user stories and their corresponding tasks have shown to be difficult to find since companies generally keep such confidential data private.

Is it possible to introduce our approach even though the team had started work months ago? Well, an agile mindset embraces change and new methodologies can be implemented on demand. However, successful reverse engineering of use cases will be possible only up to the point where an agile team adopts the method.

Jira contains project pages and provides many templates[6] for them. The most popular are templates for documenting requirements, meeting notes, and documenting retrospectives. All of these templates have predefined fields and tables which have to be written into manually. However, Jira does not provide any documentation tools for use cases, but can be supplemented by apps from the Atlassian marketplace[7]. At the time of writing, there is only one app on the

[6] https://www.atlassian.com/software/confluence/templates
[7] https://marketplace.atlassian.com

marketplace that deals with use cases - Use Case Dog[8]. Use Case Dog, however, provides a user interface on Confluence pages to help with the writing of use cases; it does not generate use cases from any artifacts.

Treude et al. [41] proposed the idea of task-based navigation of documentation. The authors created a tool that extracts tasks from software documentation using NLP. Their research is similar to ours, as they transform documentation into tasks, and we transform tasks into documentation (use cases). They evaluated whether the extracted tasks were meaningful to the developers; however, the authors did not assess the usefulness of task-based navigation - how do developers benefit from such documentation? To base the evaluation on the meaningfulness of the extracted tasks does not seem to capture the full potential of this approach; it would be beneficial to evaluate whether task-based documentation shortens the time to find information in the documentation or whether it increases the number of completed tasks.

There have been attempts to preserve use case flows in source code to maintain software comprehensibility [42, 43]. We believe that our approach could provide the missing link between the use case steps and the source code [44] by enabling the connection task to commit that modern PMSs use.

Agile teams do not have enough time to produce documentation, but it is beneficial for them to have use cases available. We have analysed the textual form of use cases, confronted our solution with existing works, described the terminology, and proposed our own solution which uses artifacts such as user stories' tasks from which no one else generated use cases. We implemented a Confluence macro that generates a textual form of use cases from Jira user stories. Interviews were conducted with six agile teams of student participants. The teams did not see any obstacles that would prevent them from using the method and were able to use the method successfully. The results show it is possible to generate use case specifications from artifacts in Jira. We conclude that using the method is feasible for developers.

9 Threads to Validity

This study has several potential threats to validity. First, the participants were master's students enrolled in a software engineering course. While they possessed sufficient theoretical and practical knowledge of agile methodologies, their level of professional experience may differ from that of practitioners in industry. Therefore, the findings may not fully generalize to professional software development contexts.

[8] https://marketplace.atlassian.com/apps/1220631/use-case-dog

Second, the proposed method relies on the consistent use of a user story template ("As a << type of user >>, I want << goal >>[, so that << benefit >>]") and on the disciplined creation of tasks following the recommended imperative style. In real-world projects, teams often adapt templates, omit punctuation, or define tasks inconsistently depending on their chosen agile methodology and project conventions. Such deviations may reduce the accuracy of automated use case generation and affect the applicability of the method across diverse teams.

Acknowledgements

This work is funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project Artificial Intelligence for Legal Professions (AILE) No. 09I05-03-V02-00038 and by the Cultural and Educational Grant Agency of Slovak Republic (KEGA) under grant No. KG 014STU-4/2024, also by the Slovak Research and Development Agency under Contract no. APVV-23-0408.

Declarations

High similarity to a number of literature - Due to my own thesis https://opac.crzp.sk/?fn=detailBiblioForm&sid=4F55101800366CEF55F4C3DB109A, The manuscript is not under review anywhere currently.

References

[1] Christoph Johann Stettina and Werner Heijstek. "Necessary and neglected?An empirical study of internal documentation in agile software developmentteams". In: Proceedings of the 29th ACM international conference on Design of communication. 2011, pp. 159-166

[2] Xin Xia et al. "Measuring program comprehension: A large-scale field study with professionals". In: IEEE Transactions on Software Engineering 44.10 (2017), pp. 951-976

[3] Peggy Gregory et al. "Onboarding: How Newcomers Integrate into an Agile Project Team". In: International Conference on Agile Software Development. Springer, Cham. 2020, pp. 20-36

[4] Rick Kazman et al. "Evaluating the effects of architectural documentation: A case study of a large scale open source project". In: IEEE Transactions on Software Engineering 42.3 (2015), pp. 220-260

[5] Rashina Hoda, James Noble, and Stuart Marshall. "Documentation strategies on agile software development projects". In: International Journal of Agile and Extreme Software Development 1.1 (2012) pp. 23-37

[6] Jirat Pasuksmit, Patanamon Thongtanunam, and Shanika Karunasekera. "Towards Just-Enough Documentation for Agile Effort Estimation: What Information Should Be Documented?" In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) IEEE, 2021, pp. 114-125

[7] Yamini Pathania and Gaurav Bathla. “A review on re-documentation approaches and their comparative study”. In: Int. J. Comput. Sci. Trends Technol 2 (2014), pp. 48-51

[8] Sugumaran Nallusamy and Suhaimi Ibrahim. “A Review of Redocumentation Approaches”

[9] Christoph Johann Stettina and Egbert Kroon. “Is there an agile handover? an empirical study of documentation and project handover practices across agile software teams”. In: 2013 International Conference on Engineering, Technology and Innovation (ICE) & IEEE International Technology Management Conference, IEEE, 2013, pp. 1-12

[10] Elvan Kula et al. “Factors affecting on-time delivery in large-scale agile software development”. In: IEEE Transactions on Software Engineering 48.9 (2021), pp. 3573-3592

[11] Yang Wang, Ivan Bogicevic, and Stefan Wagner. “A study of safety documentation in a scrum development process”. In: Proceedings of the XP2017 Scientific Workshops. 2017, pp. 1-5

[12] Ivar Jacobson. “Use cases–Yesterday, today, and tomorrow”. In: Software & Systems Modeling 3.3 (2004), pp. 210-220

[13] Saurabh Tiwari and Atul Gupta. “A systematic literature review of use case specifications research”. In: Information and Software Technology 67 (2015), pp. 128-158

[14] Erik Arisholm et al. “The impact of UML documentation on software maintenance: An experimental evaluation”. In: IEEE Transactions on Software Engineering 32.6 (2006), pp. 365-381

[15] Tony Clear. “Documentation and agile methods: striking a balance”. In: ACM SIGCSE Bulletin 35.2 (2003), pp. 12-13

[16] Henrique F Soares et al. “Investigating the link between user stories and documentation debt on software projects”. In: 2015 12th International Conference on Information Technology-New Generations. IEEE. 2015, pp. 385-390

[17] Rosalva E Gallardo-Valencia, Vivian Olivera, and Susan Elliott Sim. “Are use cases beneficial for developers using agile requirements?” In: 2007 Fifth International Workshop on Comparative Evaluation in Requirements Engineering. IEEE. 2007, pp. 11-22

[18] Wojciech J Dzidek, Erik Arisholm, and Lionel C Briand. “A realistic empirical evaluation of the costs and benefits of UML in software maintenance”. In: IEEE Transactions on software engineering 34.3 (2008), pp. 407-432

[19] Alistair Cockburn. Writing effective use cases. Pearson Education India, 2001

[20] Abdelsalam M. Maatuk and Esra A. Abdelnabi. “Generating UML Use Case and Activity Diagrams Using NLP Techniques and Heuristics Rules”. In: International Conference on Data Science, E-learning and Information Systems 2021. 2021, pp. 271-277

[21] Takwa Kochbati et al. “From User Stories to Models: A Machine Learning Empowered Automation.” In: MODELSWARD. 2021, pp. 28-40

[22] Saurabh Tiwari, Deepti Ameta, and Asim Banerjee. “An approach to identify use case scenarios from textual requirements specification”. In: Proceedings of the 12th Innovations on Software Engineering Conference (Formerly Known As India Software Engineering Conference). 2019, pp. 1-11

[23] Fabian Gilson, Matthias Galster, and Fran¸cois Georis. “Generating use case scenarios from user stories”. In: Proceedings of the International Conference on Software and System Processes. 2020, pp. 31-40

[24] Dildre G Vasques et al. “Use Case Extraction through Knowledge Acquisition”. In: 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). IEEE. 2019, pp. 0624-0631

[25] Dildre Vasques et al. “Verbka: an approach to building causal concept maps based on verbal semantics”. In: Sept. 2016

[26] Meryem Elallaoui, Khalid Nafil, and Raja Touahni. “Automatic transformation of user stories into UML use case diagrams using NLP techniques”. In: Procedia computer science 130 (2018), pp. 42-49

[27] Stefan Voigt et al. “Agile Documentation Tool Concept”. In: Developments and Advances in Intelligent Systems and Applications. Springer, 2018, pp. 67-79

[28] Silvia Abrahao et al. “Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: Results from a family of five experiments”. In: IEEE transactions on software engineering 39.3 (2012), pp. 327-342

[29] Kashumi Madampe, Rashina Hoda, and John Grundy. “A faceted taxonomy of requirements changes in agile contexts”. In: IEEE Transactions on Software Engineering 48.10 (2021), pp. 3737-3752

[30] Marcela Ruiz and Bj¨orn Hasselman. “Can We Design Software as We Talk?” In: Enterprise, Business-Process and Information Systems Modeling. Springer, 2020, pp. 327-334

[31] Bianca Minetto Napole˜ao et al. “Synthesizing researches on Knowledge Management and Agile Software Development using the Meta-ethnography method”. In: Journal of Systems and Software 178 (2021) p. 110973

[32] Laurens M¨uter et al. “Refinement of user stories into backlog items: Linguistic structure and action verbs”. In: International Working Conference

on Requirements Engineering: Foundation for Software Quality. Springer. 2019, pp. 109-116

[33] Ravi Madanayake, GKA Dias, and ND Kodikara. “Use stories vs UML use cases in modular transformation”. In: International Journal of Scientific Engineering and Applied Science 3.1 (2016), pp. 1-5

[34] Nuno Santos et al. “Deriving user stories for distributed Scrum teams from iterative refinement of architectural models”. In: Proceedings of the 19th International Conference on Agile Software Development: Companion. 2018, pp. 1-4

[35] IVAR JACOBSON, IAN SPENCE, and BRIAN KERR. “Use-Case 2.0”. In:(2016)

[36] Laurie Williams. “Agile software development methodologies and practices”. In:Advances in computers. Vol. 80, Elsevier, 2010, pp. 1-44

[37] Marco Kuhrmann et al. “What makes agile software development agile?” In: IEEE transactions on software engineering 48.9 (2021) pp. 3523-3539

[38] Emanuel Dells´en, Karl Westg˚ardh, and Jennifer Horkoff. “Invest in Splitting:User Story Splitting Within the Software Industry”. In: Requirements Engineer-ing: Foundation for Software Quality: 28th International Working Conference,REFSQ 2022, Birmingham, UK, March 21-24, 2022, Proceedings. Springer.2022, pp. 115-130

[39] Mike Cohn. User stories applied: For agile software development. Addison-Wesley Professional, 2004

[40] Lionel C Briand, Yvan Labiche, and Johanne Leduc. “Toward the reverse engineering of UML sequence diagrams for distributed Java software”. In: IEEE Transactions on Software Engineering 32.9 (2006), pp. 642-663

[41] Christoph Treude, Martin P Robillard, and Barth´el´emy Dagenais. “Extracting development tasks to navigate software documentation”. In: IEEE Transactions on Software Engineering 41.6 (2014), pp. 565-581

[42] Peter Berta et al. “Employing issues and commits for in-code sentence based use case identification and remodularization”. In: Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems. 2017, pp. 1-8

[43] Michal Bystrický and Valentino Vranic. “Modularizing code by use cases and tests for better maintainability”. In: Companion Proceedings of the 1st Interna-tional Conference on the Art, Science, and Engineering of Programming. 2017, pp. 1-1

[44] Michal Bystrický and Valentino Vranic. “Preserving use case flows in source code: Approach, context, and challenges”. In: Computer Science and Information Systems 14.2 (2017), pp. 423-445