Generating Use Case Scenarios from User Stories

Fabian Gilson
University of Canterbury
Christchurch, New Zealand
fabian.gilson@canterbury.ac.nz

Matthias Galster
University of Canterbury
Christchurch, New Zealand
mgalster@ieee.org

François Georis
University of Namur
Namur, Belgium
francois.georis@student.unamur.be

ABSTRACT

Textual user stories capture interactions of users with the system as high-level requirements. However, user stories are typically rather short and backlogs can include many stories. This makes it hard to (a) maintain user stories and backlogs, (b) fully understand the scope of a software project without a detailed analysis of the backlog, and (c) analyse how user stories impact design decisions during sprint planning and implementation. This paper proposes a technique to automatically transform textual user stories into visual use case scenarios in the form of robustness diagrams (a semi-formal scenario-based visualisation of workflows). In addition to creating diagrams for individual stories, the technique allows combining diagrams of multiple stories into one diagram to visualise workflows within sets of stories (*e.g.*, a backlog). Moreover, the technique supports "viewpoint-based" diagrams, *i.e.*, diagrams that show relationships between actors, domain entities and user interfaces starting from a diagram element (*e.g.*, an actor) selected by the analyst. The technique utilises natural language processing and rule-based transformations. We evaluated the technique with more than 1,400 user stories from 22 backlogs and show that (a) the technique generates syntactically valid robustness diagrams, and (b) the quality of automatically generated robustness diagrams compares to the quality of diagrams created by human experts, but depends on the quality of the textual user stories.

CCS CONCEPTS

• **Software and its engineering** → *Software system structures*; *Model-driven software engineering*; *System description languages*; *Designing software*; *Software development techniques.*

KEYWORDS

Agile software development, textual user stories, model-driven software development, natural language processing

ACM Reference Format:
Fabian Gilson, Matthias Galster, and François Georis. 2020. Generating Use Case Scenarios from User Stories. In *International Conference on Software and System Processes(ICSSP '20), October 10–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3379177.3388895

1 INTRODUCTION

1.1 Background and Problem Description

In agile software development, *textual user stories* are a simple yet effective way to record requirements from a user's perspective [30, 38]. While the format and style of user stories vary between organisations, literature recommends various forms [47]. For example, one of the most popular templates used in industry [30, 40, 47] is Cohn's template [4] which describes stories in the form "As a <*actor/user*> I want <*goal/action*> so that <*reason/value*>." (*e.g.*, *"As a customer I want to pay online by credit card so that I can keep track of my spendings."*). Agile practices also suggest adding acceptance criteria to each story. Since this requires a more detailed analysis of stories, acceptance criteria are often not defined [20].

User stories are rather high-level and each major user action is captured in a separate story. Hence, developers may lose the "bigger picture" of the system they are developing while the number of stories increases [39]. Also, the longer the backlog and the more a backlog evolves, the more effort is needed to maintain it [3], *e.g.*, to identify redundant stories or shared behaviour between stories [40]. Furthermore, implementation-related aspects (*e.g.*, architecturally significant requirements, quality attributes) are typically spread across many stories [16, 19]. Finally, large backlogs make it difficult to identify dependencies between user stories, *e.g.*, when prioritising stories and during release planning [22]. Therefore, managing growing backlogs remains challenging and time-consuming [3], but still must be done continuously [36].

Visual notations and graphical models help communicate information between different types of stakeholders as they convey information more effectively and are easier to understand and remember than textual artefacts [18, 31]. Alexander and Maiden [1] and Genon *et al.* [14] advocate the use of graphical visualisations throughout software development to support communication and analysis. Visual modelling languages, sometimes using ad-hoc notations [42], are used more frequently than formalised and restrictive UML models [17, 34].

1.2 Paper Goals and Contributions

To mitigate the problems with textual user stories and benefit from visual representations, we investigate the following:

(1) How can we *visualise* core concepts and processes captured in a *textual user story*? This would provide a representation of user stories that is easier to comprehend by different types of stakeholders (*e.g.*, technical and non-technical, domain experts and non-experts) [18].
(2) How can we combine *multiple* stories into one visualisation where similar sub-processes of different stories are merged? This would help identify and group common conceptual elements such as user interfaces or domain entities. It would

also help automatically identify dependencies between stories. For example, during sprint planning this could help pinpoint prioritisation issues (*e.g.*, stories that depend on each other should be implemented together) and reusable behaviour.

(3) How can we generate different graphical *views* of a backlog by "pruning" visual diagrams based on a particular element from the diagram (*e.g.*, a user, interface or entity)? This would help identify missing or redundant stories in the backlog. For example, a story may mention updating an entity such as a credit card bill when a new purchase is completed, but there is no story where this entity (credit card bill) is created.

In detail, this paper makes the following **contributions**:

(1) We propose a technique based on Natural Language Processing (NLP) and rule-based transformations to automatically generate visual use case scenarios from user stories in the form of robustness diagrams [24, 37]:
   (a) It generates one robustness diagram per story that visualises the core conceptual and behavioural concepts of that story, *e.g.*, actors, domain entities, processes, and relationships between these concepts.
   (b) It allows reconciling *multiple* user stories into a *combined* robustness diagram for a *set* of stories.
   (c) It allows creating "viewpoint-based" diagrams where a conceptual element is selected based on the concerns of a particular stakeholder [23] to "prune" a larger diagram so that only that element and its relevant relationships are visualised.

   Therefore, the technique not only *visualises* textual information from user stories, but also *integrates* information from different stories to provide insights not available from stories in isolation (*e.g.*, common actors).
(2) We develop a web-based prototype publicly available to support the use of the technique.
(3) We evaluate the technique with more than 1,400 user stories from 22 public backlogs. The evaluation investigates (a) whether the technique generates syntactically valid robustness diagrams, and (b) the semantic compliance of the generated robustness diagrams compared to manually created ones. We found that overall the technique produces syntactically and semantically correct diagrams, assuming that user stories are well-written.

Our work highlights the feasibility of generating robustness diagrams from user stories. Our work complements work on generating visual models from textual user requirements (see Section 2). It is the first-of-its-kind that uses agile user stories to generate visualisations not only for single stories, but also multiple stories. Furthermore, no other technique automatically identifies common conceptual or behavioural elements in a backlog and creates user-defined views of backlogs.

The remainder of this paper is structured as follows. We discuss related work in Section 2. In Section 3 we describe our technique to generate robustness diagrams and present the results of an evaluation in Section 4. Then, we discuss the results and our technique in Section 5. We summarise our contribution and describe future work in Section 6.

2 RELATED WORK

Automatic Extraction of Domain Concepts. Techniques exist to automatically extract domain concepts (*e.g.*, objects, classes) from requirement documents and generate class or entity-relationship diagrams [2, 13, 15, 21, 26, 28, 44, 50]. Most techniques produce rather technical and implementation-focused *structural* UML diagrams (*e.g.*, class diagrams that map directly to code) or specifically focus on either structural or behavioural aspects [49]. Our work aims at models that also describe *behavioural* aspects of software together with domain knowledge for various types of stakeholders.

Automatic Generation of Use Case Models. Works that generate behavioural models from user requirements include El Attar and Miller who produce UML use case diagrams from textual use case scenarios written as structured steps [9]. Nguyen *et al.* create a view of textual use cases and hierarchical goal models from requirement documents [32]. Similarly to our approach, Elallaoui *et al.* generate use case diagrams from user stories [12]. However, the use case diagrams created in that work do not provide insights into behavioural aspects in terms of execution semantics. Moreover, even though use case diagrams share similarities with robustness diagrams, they do not formally identify domain entities other than actors.

Automatic Generation of Behavioural Diagrams from Requirements. Takhur and Gupta transform textual use case descriptions written as numbered steps into UML sequence diagrams [43]. Similarly, Tiwari *et al.* also create adapted actor-focused activity diagrams from numbered steps [45]. Elalloui *et al.* generate UML sequence diagrams from user stories [11]. In all those works, the generated diagrams do not use the full expressive power of sequence diagrams, *e.g.,* actions are aggregated into (one-way) single interactions and common behaviour is either not or only partially identified. Also, diagrams in these approaches do not identify design aspects, e.g., domain entities or user interfaces.

Backlog Analysis. Little research has been done on reconciling user stories and extracting the bigger picture out of a backlog. Lin *et al.* [27] and Wautelet *et al.* [48] build goal models from (annotated) user stories, but these two techniques are mostly manual, whereas our technique is fully automated. Kamalrudin *et al.* propose MaramaAIC to evaluate the consistency of a backlog using a custom notation, the Essential Use Cases [25]. However, this tool neither merges common elements, nor creates views or identifies domain concepts and deals with longer requirements than user stories.

Summary. Our technique differs to earlier works as follows:

(1) It generates diagrams from user stories, the de-facto standard for requirements in agile development;
(2) It considers both structural *and* behavioural aspects, hence combining information about how a system is used with potential implementation details and reusable sub-processes imposed by one or more user stories;
(3) It automatically integrates multiple stories into a single diagram where common elements are merged to help analyse sets of stories;
(4) Diagrams for multiple stories can be pruned to provide a particular "view" of a system based on individual elements

selected by developers to enhance the readability of potentially large diagrams;

(5) Diagrams target a wider audience beyond technical stakeholders by using robustness diagrams that can be understood by a broader range of stakeholders common in agile projects (*e.g.,* product owners, domain experts, developers).

3 PROPOSED TECHNIQUE

3.1 Robustness Diagrams in a Nutshell

Our technique generates robustness diagrams as a visual representation of textual user stories. Robustness diagrams are a scenario-based description of interactions and workflows introduced by Jacobson [24] and developed further by Rosenberg and Stephens [37]. These diagrams visualise domain elements of a scenario and behavioural aspects of a system.

During software analysis and implementation, robustness diagrams help identify critical sub-processes and entities involved in a use case scenario [7, 10]. Furthermore, developers can use robustness diagrams to systematically identify user interfaces, persistent design objects as well as to define acceptance criteria for stories [10]. This is particularly useful in agile development where visual representations of user stories can facilitate discussions during sprint planning meetings. Figure 1 provides an overview of the basic conceptual elements and the notation of robustness diagrams.

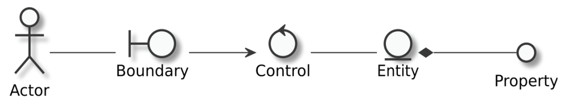

Figure 1: Conceptual elements in a robustness diagram.

Robustness diagrams include five types of elements:

(1) An Actor corresponds to a type of user or persona that interacts with the system. An Actor in robustness diagrams is identical to the concept of an actor in user stories. An Actor may only be linked to a Boundary.

(2) A Boundary represents the interface between an Actor and the system. Robustness diagrams have a loose definition of interfaces which include graphical interfaces, peripherals (*e.g.*, touchpads, printers) or API endpoints. A Boundary may only be linked to a Control.

(3) An Entity is any domain object that is part of the application domain of the system, *e.g.*, an online bookstore has domain objects like book, shopping card or order.

(4) A Control bridges a Boundary to an Entity. It represents the "glue" or logical steps to access, process and transform data, *e.g.*, to process search queries. A Control may be linked to one or more Control, Boundary or Entity elements.

(5) A Property is an attribute of an Entity, *e.g.*, a book would have a title. A Property may only be linked to an Entity.

From a design point of view, the element types of robustness diagrams map to the Model-View-Controller pattern: a Boundary maps to the View, a Control maps to the Controller and an Entity maps to the Model. We chose robustness diagrams as the visual representation of user stories for the following reasons:

(1) They are understandable by different types of stakeholders as they rely on few well-defined concepts. For example, it has been shown empirically that robustness diagrams do not require in-depth knowledge of object-oriented concepts [8] and business analysts and product owners can quickly learn how to use them effectively.

(2) Robustness diagrams provide a richer notation than standard UML use case diagrams which focus on behavioural issues only and are similar to UML collaboration diagrams. Therefore this notation is readable by developers familiar with UML.

(3) Robustness diagrams have already been successfully used in industry [24], for example to define behavioural specifications of use case scenarios [37] and to support user acceptance testing [7, 10].

(4) Robustness diagrams are supported by a wide range of tools (*e.g.*, *draw.io*[1], *MagicDraw*[2], *Visual Paradigm*[3]).

3.2 Overview of Proposed Technique

The *input* into our technique are textual user stories following Cohn's template [4], since it is widely used in industry [30, 47]. The *output* is either one robustness diagram per user story (see Section 3.4), one combined robustness diagram for a set of user stories, *e.g.*, a complete backlog (see Section 3.5), or one robustness diagram derived by selecting one particular "viewpoint" (see Section 3.6). The technique follows a pipelined process shown in Figure 2.

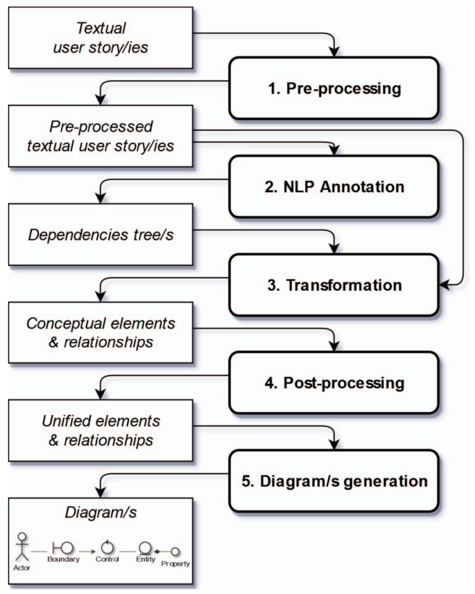

Figure 2: Overview of technique to generate robustness diagrams.

The five steps shown in Figure 2 are the following:

[1]See https://www.draw.io/
[2]See https://www.nomagic.com/products/magicdraw
[3]See https://www.visual-paradigm.com/

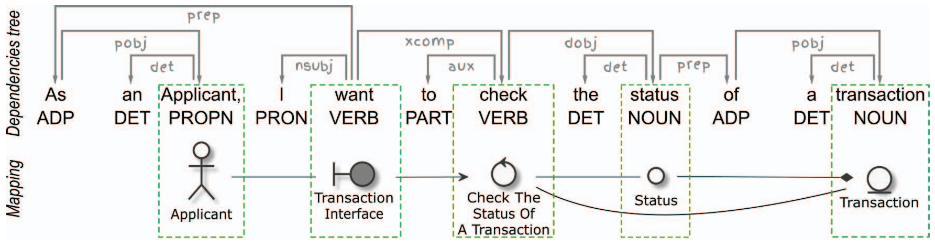

Figure 3: Dependencies tree of a user story and mapping to a robustness diagram (part-of-speech tagging created by spaCy using Google's tagset [6]).

(1) **Pre-processing** applies rules to "clean" and prepare textual user stories for processing, see Section 3.3.
(2) **NLP annotation** first applies stemming and lemmatisation to the (still textual) stories. Then, it parses the stories to identify types of words (*e.g.*, verb, noun) and creates a Universal Stanford dependencies tree [6]. In short, a dependencies tree connects the words of input text by typed dependency relations, see *e.g.*, the upper portion of Figure 3 ("Dependencies tree") where the subject *"I"* is linked to *"want"* with a nominal subject `nsubj` relationship. The full list of types of words and dependency relations is available in the universal part-of-speech tagset developed by Google [6].
(3) **Transformation** takes the textual representation and the dependencies tree of stories from the previous step and transforms annotated elements of a story into conceptual elements of a robustness diagram and their relationships (see Section 3.4 for details). An example is shown in the lower portion of Figure 3 ("Mapping").
(4) **Post-processing** merges similar conceptual elements from the previous step (*e.g.*, elements of the same type with same or similar names) in order to avoid duplicates when creating the visualisation of a robustness diagram. This step also handles combining multiple stories into one diagram and generating viewpoint-based diagrams.
(5) **Diagram generation** produces the final visualisation by reconciling unique elements from step (4) with their relationships created in step 3 and maps them to an implementation-specific diagramming language.

3.3 Pre-processing User Stories

We remove special characters or punctuation errors from user stories, since they may compromise diagram generation. Furthermore, due to the wide range of textual forms of a user story (which often violate good specification principles such as INVEST [46] or principles described in the QUS framework [29]), we need to remove common anti-patterns in user stories (*e.g.*, non-atomic stories). We therefore perform the following corrective actions:

(1) Remove unknown actors from the <*goal*>-part of stories (for example, in a story like "As users, me and my team want..." does not define "me and my team" as <*actor*> but considers them as part of the <*goal*> (only "users" is considered an actor). Such undefined actors could be partially resolved as domain objects rather than actors and thus result in invalid diagrams. Therefore, we remove these undefined actors from the <*goal*>.
(2) Split non-atomic stories composed of two or more sub-stories combined with conjunctions. For example, a story such as "As a user I want to <*goal*> AND <*goal*>") will be broken into two separate stories with two different goals but with the same actor.
(3) Replace pronouns by the referenced term to simplify the co-referencing of diagram elements and their properties when establishing relationships between diagram elements. For example, in a story "... I want to edit my profile and see it on my landing page...", the word "it" will be replaced by the referenced word "profile".
(4) Add missing "action" verb after the "I want" part of user stories since it may cause problems when identifying Control and Entity elements of the generated robustness diagram. We decided to insert a neutral *"to be provided with"* action, *e.g.*, a story "I want a profile" would be modified to "I want *to be provided with* a profile".

3.4 Processing Individual User Stories

Figure 3 depicts how a (pre-processed) user story is mapped to a robustness diagram in step 5 of Figure 2. A robustness diagram is always composed of at least four elements, *i.e.*, an Actor, a Boundary, a Control and an Entity. Property elements are optional. We use the following mapping rules:

(1) Each noun from the <*actor*>-part becomes an Actor and is typically identified as a noun (`NOUN`) or a proper noun (`PROPN`) by the universal part-of-speech tagging [6]. For example, in Figure 3, "Applicant" is transformed into an Actor in the robustness diagram.
(2) The (atomic) <*goal*> of a story becomes a Control. For example, in Figure 3, the <*goal*> with the `VERB` "check" and its direct object (`DOBJ`) "status" is mapped to a Control.
(3) Nouns present in the <*goal*>-part are mapped to Entity elements. For example, in Figure 3, the term "transaction" is identified as an Entity.
(4) Any Entity referring to another Entity (*e.g.*, using an adposition (`ADP`) like "of" or a particle) is transformed into a Property of that Entity. For example, "Status" in Figure 3 is defined as a Property of "Transaction".
(5) From the <*goal*>-part, any term semantically close to "interface" or "screen" is mapped to a Boundary. For example, the story from Figure 3 does not mention any term referring

to an interface. However, such element is mandatory in a robustness diagram. Therefore, we create a Boundary out of the identified "Transaction" Entity (shown as darker shape to distinguish from directly mapped elements).

3.5 Processing Multiple User Stories

To understand the relationships between stories, we need to integrate multiple diagrams for a set of stories into one merged diagram. As an example, Figure 4 shows a combined robustness diagram for five stories taken from the backlog for the Scrum Alliance website (see Section 4.2) listed in Table 1.

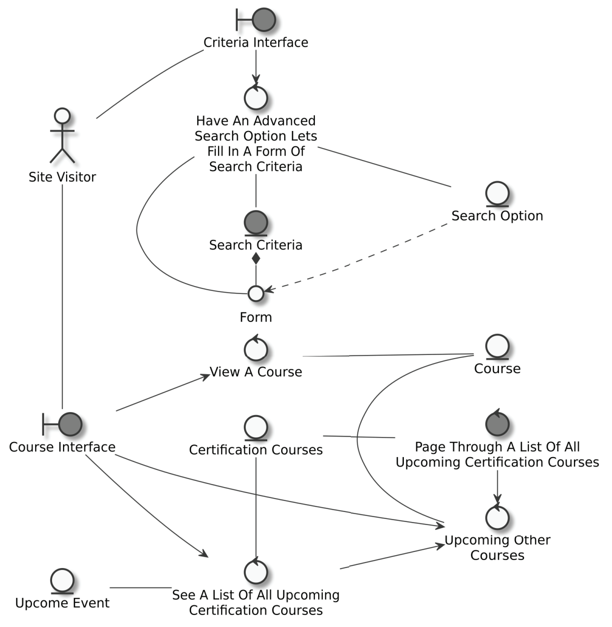
Criteria Interface
Have An Advanced Search Option Lets Fill In A Form Of Search Criteria
Site Visitor
Search Option
Search Criteria
Form
View A Course
Course
Course Interface
Certification Courses
Page Through A List Of All Upcoming Certification Courses
Upcoming Other Courses
Upcome Event
See A List Of All Upcoming Certification Courses

Figure 4: Combined robustness diagram for the five user stories listed in Table 1.

Merged diagrams are created during post-processing if multiple stories are chosen as input into the technique (see Figure 2) as follows:

(1) Similar elements from different stories are merged into one single element (based on names, synonyms, etc.).
(2) The final type of Entity elements and their relationships are evaluated as follows:
  (a) When an Entity is identified as a Property in any of the stories, it is transformed to a Property.
  (b) When there is a close relationship between two Entity elements (*e.g.*, one Entity is used to update the value of another Entity), we define a Dependency relationship between these two Entities.
(3) After the final set of elements and relationships is obtained, the intersection of diagram elements and the unique union of relationships between elements compose the combined diagram.

Table 1: User stories combined into one diagram in Figure 4.

	User story
1.	As a site visitor, I want to see a list of all upcoming Certification Courses and can page through them if there are a lot, so that I can choose the best course for me.
2.	As a site visitor, I want to see a list of all upcoming Other Courses and can page through them if necessary, so that I can choose the best course for me.
3.	As a site visitor, I want to see a list of all upcoming Events, so that I can decide if I want to attend any.
4.	As a site visitor, I want to have an advanced search option that lets me fill in a form of search criteria, so that I can quickly find what I am looking for.
5.	As a site visitor, I want to view a course I can click on the trainer's name and be taken to the trainer's profile, so that I can read more about a trainer before registering for a course.

As visible in Figure 4, two Boundaries are correctly inferred (*i.e.* the "Course Interface" and "Criteria Interface") to which all Control elements identified from the stories in Table 1 are linked to, creating a combined visualisation for all five stories.

3.6 Viewpoint-Based Diagrams

With long backlogs of stories, creating one combined diagram may result in a visualisation that is difficult to grasp. Our technique can create a view of a larger diagram based on a selected viewpoint being any Boundary, Entity or Actor that is part of a complete diagram. Figure 5 shows a view for the stories in Table 1 using the Boundary "Course Interface" as the viewpoint and is a sub-diagram of Figure 4. Similar views could have been generated with any other element as viewpoint, resulting in a sub-diagram of the complete diagram.

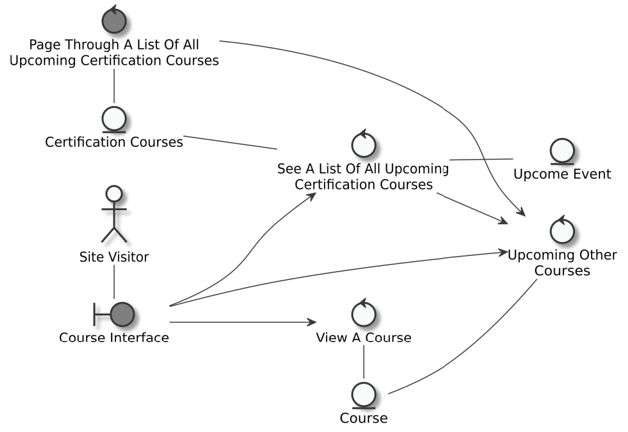
Page Through A List Of All Upcoming Certification Courses
Certification Courses
See A List Of All Upcoming Certification Courses
Upcome Event
Site Visitor
Upcoming Other Courses
Course Interface
View A Course
Course

Figure 5: Viewpoint-based robustness diagram (viewpoint: "Course Interface").

To create views, our technique utilises a classical graph traversal approach during post-processing (step 4 in Figure 2). A combined diagram (See Section 3.5) is then traversed according to the selected viewpoint:

(1) If an Actor is selected as viewpoint, all linked Boundaries are added to the view from which all linked Controls and subsequent Entities and Properties are added to the view.
(2) If a Boundary is chosen, the process is similar to an Actor-based viewpoint to retrieve all directly connected Controls, their linked Entities and Properties, along with all Actors directly linked to this Boundary.
(3) If an Entity is selected, we retrieve all directly linked Properties and Controls, as well as all Dependency relationships. From the Control elements, we add all directly linked Boundaries and their connected Actors.

Our implementation of the technique (see Section 3.7) also generates a list of stories that are represented in the current view (as a csv file). This list is another means of offering traceability of diagrams to their stories.

3.7 Prototype Implementation

The implementation of the pipeline shown in Figure 2, available online[4], is as follows:

(1) **Pre-processing** is implemented as a set of custom rules written in Python.
(2) **NLP annotation** is handled by spaCy, a general purpose NLP parser[5] and extension of Google's universal part-of-speech tagset [35]. spaCy also performs stemming and lemmatisation. spaCy performs well on unconstrained texts written in common language [33].
(3) **Transformation** is implemented in Python and creates conceptual elements as Python objects from named entities and dependencies trees created by spaCy.
(4) **Post-processing** relies on custom Python code. Similarity detection utilises WordNet[6] through the Natural language ToolKit (NLTK)[7]. WordNet is a large lexical database of synonyms and is designed to be integrated into NLP tools.
(5) **Diagram generation** maps the conceptual elements (in Python) to human-readable specifications interpreted by PlantUML[8]. PlantUML is a lightweight open source library that creates UML diagrams (including robustness diagrams) from text files. Each identified conceptual diagram element together with its relationships is converted into the PlantUML syntax that is processed by the library to create and visualise the diagram.

4 EVALUATION

4.1 Research Questions

We defined the following research questions to evaluate the syntactic and semantic correctness of the proposed technique:

RQ1 Are robustness diagrams generated from user stories syntactically correct? This research question aims at evaluating to what extend an automatic process based on natural language processing creates syntactically valid robustness diagrams from real-world user stories.

RQ2 Are robustness diagrams generated from user stories semantically valid? This research question aims at evaluating the semantic correctness of generated diagrams compared to robustness diagrams written by human experts.

RQ3 Does the automatic composition of user stories into combined robustness diagrams identify common processes (*i.e.* Control elements), domain concepts (*i.e.* Entity elements) and user interfaces (*i.e.* Boundary elements)? We evaluate if the combination of multiple stories into one diagram correctly reconciles identical elements from the individual stories.

RQ4 Does pruning of a merged robustness diagram remove all relationships that are not relevant for the created view based on that element? We evaluate if a viewpoint-based diagram correctly removes all indirect relationships to a particular Actor, Boundary or Entity.

Note that we did not perform a user study to evaluate the usefulness of the generated robustness diagrams, since this notation has been previously evaluated [8, 37] and is supported by many tools (*e.g.*, MagicDraw, Visual Paradigm). All data as well as the diagrams generated during the evaluation are available online[4].

4.2 User Stories for Evaluation

We used real-world user stories collected by Dalpiaz *et al.* [5], see Table 2. In total we included backlogs for 22 projects and 1,675 user stories. The number of user stories ranges from 50 to 114 with an average of 76 per backlog. The average length of user stories over all backlogs is 24 words. The backlogs came from various domains, *e.g.*, a recycling app, a management system for holiday camps or a website for Scrum Alliance. Since user stories were taken from real-world projects, we cannot assume that those who created the stories have evaluated their quality. We relied on how user stories were captured by their stakeholders.

4.3 Data Collection and Analysis

To answer **RQ1** of the evaluation (are robustness diagrams syntactically correct), we determined the following metrics by manually inspecting the generated diagrams:

(1) *Number of valid diagrams*: a diagram is valid if all of its elements are connected to at least one other element (*i.e.* there are no orphans in the graph) and the diagram complies with the syntax of robustness diagrams.
(2) *Number of invalid diagrams*: A diagram is invalid if at least one element in the diagram is not connected to at least one other element or does violate the syntax.
(3) *Error*: We recorded errors if no diagram could be generated (*e.g.*, if spaCy could not create a dependencies tree).

To answer **RQ2** (are diagrams semantically correct), two of the researchers randomly selected three syntactically correct stories per backlog (*i.e.* 66 stories in total) and manually created robustness diagrams. We consider these manually created diagrams as the "ground truth". We then compared the manually created diagrams with the automatically generated diagrams based on these criteria:

[4]See https://bitbucket.org/mde4asd/ucscenario-gen
[5]See https://spacy.io/
[6]See https://wordnet.princeton.edu/
[7]See https://www.nltk.org
[8]See http://plantuml.com/

Table 2: Product backlogs and results for RQ1 (one diagram is created for each story on each backlog).

Backlog	Description	Valid	Invalid	Error	Total
FederalSpending	Web platform for sharing US government spending data	72 (76.6%)	22 (23.4%)	0 (0%)	94
Loudoun	Land management system for Loudoun County, Virginia	53 (92.9%)	4 (7.1%)	0 (0%)	57
Recycling	Online platform to support waste recycling	46 (92.0%)	4 (8.0%)	0 (0%)	50
OpenSpending	Website to increase transparency of government expenses	49 (92.4%)	4 (07.6%)	0 (0%)	53
FrictionLess	Platform for obtaining insights from data	57 (86.4%)	7 (10.6%)	2 (3%)	66
ScrumAlliance	First version of the Scrum Alliance website	84 (86.6%)	13 (13.4%)	0 (0%)	97
NSF	New version of the NSF website	53 (73.6%)	19 (26.4%)	0 (0%)	72
CamperPlus	App for camp administrators and parents	49 (92.4%)	4 (7.6%)	0 (0%)	53
PlanningPoker	First version of the PlanningPoker.com website	48 (92.3%)	4 (7.7%)	0 (0%)	52
DataHub	Platform to find, share and publish data online	60 (89.5%)	7 (10.5%)	0 (0%)	67
MIS	Management information system for Duke University	66 (79.5%)	16 (19.3%)	1 (1.2%)	83
CASK	Toolbox to for fast and easy development with Hadoop	50 (78.1%)	14 (21.9%)	0 (0%)	64
NeuroHub	Research data management portal	90 (88.2%)	12 (11.8%)	0 (0%)	102
Alfred	Personal interactive assistant for active aging	117 (87.3%)	17 (12.7%)	0 (0%)	134
BadCamp	Conference registration and management platform	62 (89.9%)	7 (10.1%)	0 (0%)	69
RDA-DMP	Software for machine-actionable data management plans	54 (65.9%)	28 (34.1%)	0 (0%)	82
ArchiveSpace	Web-based archiving information system	46 (83.6%)	9 (16.4%)	0 (0%)	55
UniBath	Institutional data repository for the University of Bath	51 (96.2%)	2 (3.8%)	0 (0%)	53
DuraSpace	Repository for different types of digital content	80 (80.0%)	20 (20.0%)	0 (0%)	100
RacDam	Software for archivists	94 (94.0%)	6 (6.0%)	0 (0%)	100
CulRepo	Content management system for Cornell University	88 (77.2%)	26 (22.8%)	0 (0%)	114
Zooniverse	Platform that allows anyone to help with research tasks	50 (83.3%)	10 (16.7%)	0 (0%)	60
	Average	**81.4%**	**18.4%**	**0.2%**	100%

(1) *Do automatically generated and manually identified Actor elements match?* We checked whether all Actor elements were generated (*Yes* in Table 4), if a *Superset* or *Subset* of Actor elements was generated, or if none of the expected Actor elements was generated (*No*);
(2) *Do automatically generated and manually identified Entity elements match?* We checked whether all expected Entity elements were generated (*Yes* in Table 4), if a *Superset* or *Subset* of Entity elements was generated or if none of the expected Entity elements was generated (*No*);
(3) *Does the automatically generated description of Control elements match that in the manually created diagrams?* We checked whether the description of Control elements made sense (valid English sentences) and if their meaning matched those of the Control elements manually identified.
(4) *Are the generated relationships between diagram elements semantically valid?* We compared relationships in manually and automatically generated diagrams.
(5) *Are the automatically generated and manually created diagrams similar in terms of the number of elements, their relationships and semantically similar names*? We checked whether the number and semantics of elements are identical in the generated and manually created diagram, or if the generated diagram includes a *Superset* of semantically equivalent Entities.

To answer **RQ3** (are common elements automatically merged in the combined diagrams) we focus on the correctness of the merged diagrams, rather than the semantic correctness of individual merged diagrams (this has already been assessed in RQ2). We investigate the scalability of our technique by merging sets of user stories of different sizes (the batches of stories are shown in Table 3). Similar to RQ2, we only consider syntactically valid robustness diagrams and their related stories. We analysed each merged diagram regarding the following aspects:

(1) *How many elements are expected in the combined diagram, i.e., does the merged diagram combine individual diagram elements without omitting elements?* We compared the number of expected elements to the number of generated elements.
(2) *How many duplicate elements were not correctly merged, i.e., how many Actor, Entity, Boundary or Control elements are semantically similar and should have been merged but were not?* We counted the number of duplicates that were not merged.

Finally, to answer **RQ4** (are non-relevant elements removed correctly from a view), we use the same batches of user stories as for RQ3 (Table 3) and checked the following:

(1) *For a given set of user stories, are all expected views correctly generated?* We compared the number of expected sub-graphs (one per unique diagram element, except for Controls) to the number of generated views. The technique should generate one view per viewpoint rather than multiple sub-graphs per viewpoint.
(2) *Are all expected user stories linked to a view correctly identified when building a viewpoint-based robustness diagram from multiple stories?* We compared the number of expected

stories to the number of stories associated to each generated sub-graph.

Table 3: Batches of user stories used for RQ3 and RQ4 (actual stories are available online).

Batch	Backlog	Stories
Batch 1	ScrumAlliance	5
Batch 2	Alfred	10
Batch 3	Loundoun	15

4.4 Evaluation Results

4.4.1 RQ1 – Syntactical Correctness of Diagrams. As can be seen in Table 2, there are only three stories for which we cannot create a robustness diagram. Two stories in the *FrictionLess* backlog did not follow Cohn's template and could not be parsed. The error from the *MIS* backlog comes from an unresolved reference to the *metadata batch editor* Actor of the part *"I'm not editing"* in the story *"As a metadata batch editor, I want to export the CSV, remove the fields I'm not editing, edit the remaining fields, and upload the remediated file to overwrite values"*. Stories for NSF's website and RDA-DMP have the highest ratio of invalid diagrams because they are written with complex grammatical structures (*e.g.*, with pronouns such as "what" or "how") or contained acronyms and URLs (with dots) that caused parsing issues with spaCy. However, invalid diagrams are rather rare (18.45% overall, see Table 2), even though some backlogs with poorly written user stories (such as RDA-DMP) lead to more invalid diagrams (34.1%). Despite sanitising stories during preprocessing, the syntactic validity of diagrams relies on spaCy's output which is not always fully correct.

4.4.2 RQ2 – Semantic Correctness of Diagrams. We present the results of the comparison between the manually created and automatically generated diagrams in Table 4 and Table 5.

Table 4: Comparison of manually created and automatically generated diagrams (RQ2 – number of elements).

Criteria	Yes	Superset	Subset	No
Actors match	62 (93.9%)	1 (1.5%)	0 (0%)	3 (4.6%)
Entities match	22 (33.3%)	30 (45.5%)	13 (19.7%)	1 (1.5%)

Table 5: Comparison of manually created and automatically generated diagrams (RQ2 – correctness of elements).

Criteria	Yes	No
Do Control descriptions match	46 (69.7%)	20 (30.3%)
Are relationships valid	52 (78.8%)	14 (21.2%)
Are diagrams identical	18 (27.3%)	48 (72.7%)
Are diagrams superset	33 (50%)	33 (50%)

Actors are created rather reliably with a very high accuracy of 93.3%, see Table 4. Furthermore, one third of the diagrams contained all expected Entities and only one diagram had no valid Entity at all. If we consider supersets of the expected Entities as acceptable results, our technique reaches an overall accuracy of 78.8% for Entities (*Yes* plus *Superset* in Table 4).

On the other hand, identifying matching description of Controls achieves lower accuracy (69.7%, see Table 5). Regarding the identified relations, we found semantically valid relationships in the generated diagrams with an accuracy of 78.8%, see Table 5. Finally, 27.3% of generated diagrams were perfectly matching the manually created ones in terms of their number of elements, their semantics and the relationships. However, if we add to the perfect matches the diagrams covering superset of elements, the accuracy increases to 50%.

4.4.3 RQ3 – Combined Diagrams. In Table 6, we present the results regarding RQ3 analysing the generated diagrams for all batch sizes. The generation of combined diagrams performs well. We found two duplicate elements that were not merged where plurals of words were not correctly lemmatised.

Table 6: Correctness of combined diagrams (RQ3).

Criteria	Batch 1	Batch 2	Batch 3
Expected elements	16	39	44
Observed elements	14	40	45
Number of duplicates	0	1	1

4.4.4 RQ4 – Viewpoint-based Diagrams. In Table 7, we present the results regarding RQ4 with our analysis of the generated diagrams for all batch sizes.

Table 7: Correctness of views (RQ4).

Criteria	Batch 1	Batch 2	Batch 3
Expected Actor sub-graphs	1	3	1
Observed Actor sub-graphs	1	3	1
Expected Boundary sub-graphs	5	8	13
Observed Boundary sub-graphs	5	9	13
Expected Entity sub-graphs	2	14	16
Observed Entity sub-graphs	2	14	16
Expected stories per view	all	superset	superset

The expected number of sub-graphs for different element types matches the observed number for all element types except the Boundary elements for backlog *Alfred* in batch 2. In the combined diagram, two Entity elements have been merged that were kept separated in the viewpoint-based generation, hence creating two separated sub-diagrams.

4.5 Threats to Validity of Evaluation

4.5.1 External Validity. Our evaluation is based on a limited number of user stories from an open dataset. There is no evidence that this set is representative of industrial practices in general. However, since how user stories are specified in practice varies (*e.g.*, phrasing and writing guidelines), it is hard to identify a truly representative set of stories. Similarly, to investigate how our approach combines and prunes diagrams, we randomly sampled stories from the dataset.

4.5.2 Internal Validity. "Ground truth" diagrams were created and checked by the authors and did not involve original project stakeholders. Significant effort has been put into ensuring the rigour of creating the diagrams and two researchers cross-checked diagrams.

4.5.3 Construct Validity. In order to evaluate how well our approach works, we defined several metrics, *e.g.*, counting the number of elements in the automatically generated diagrams and comparing them to elements in manually created diagrams. These metrics might appear simplistic, but are typically used to evaluate the quality of automatically generated diagrams in related works [11, 13, 50]. Also, as mentioned above, we treated manually created diagrams as "ground truth". Thus, our findings depend on the syntactical and semantic quality of manually created diagrams.

5 DISCUSSION

5.1 Summary of Evaluation

We did not assess the quality of user stories in terms of how they are written. We did this so that we were able to evaluate the feasibility and flexibility of an NLP-based and rule-based approach to generate robustness diagrams. In practice, we cannot always expect "perfect" stories. Still, using 1,675 user stories, on average, our technique created 81.4% syntactically valid diagrams for individual stories.

From the full set of valid diagrams, we sampled three stories per backlog and compared their semantics to manually drawn diagrams. From that sample, 4.6% of the generated diagrams did not include all expected Actor elements and 21.2% of diagrams missed at least one expected Entity. Moreover, 50% of the diagrams were semantically equivalent to those manually created (*i.e.*, including a superset of the expected elements), out of which 27.3% were perfect matches. Despite a rather high accuracy of syntactically valid diagrams generated by our technique, the semantic validity is also dependent to the grammatical complexity of the stories. Without a strictly enforced quality of stories (*e.g.*, based on INVEST [46] or QUS [29] framework), the ability for a rule-based NLP technique to generate diagrams comparable to the ones created by human is limited. Therefore, from our results, it seems reasonable that the ability to trust the generated diagrams is mostly related to the quality of the writing of stories.

The viewpoint-based generation of diagrams creates the expected number of views (with only one unexpected generated view) due to lemmatization problems. Additionally, each view is complemented with a list of stories that contribute at least one diagram element to the view. Our technique is very permissive and therefore includes any story in that list that contributes at least one diagram element to that view (instead of for example returning the stories having *all* elements displayed in the view). Such set of stories (*i.e.*, stories without all elements present in the view) is an additional benefit since it creates a minimal view visually, but still identifies all dependent user stories of that view for analysis purposes.

5.2 Limitations of Technique

Our technique has several limitations. First, it relies on stories that follow Cohn's template and require both users and actions. Hence, user stories not complying with that template may be transformed into invalid robustness diagrams. However, users of our technique may assess the quality of their stories before applying it. For example, user stories might be checked using more advanced quality assessments as proposed in other research, *e.g.*, [29, 41].

Moreover, our technique is subject to typical limitations of natural language processing. For example, typos might lead to diagram elements not being merged, even though they refer to the same element. On the other hand, inconsistent use of synonyms could mean that elements are merged even though they should not be merged. Misused synonyms would however also cause problems when manually analysing user stories and require developers to check with the creators of the story.

Furthermore, stories written from a developer's point of view (*e.g.*, "As a tester, I would like to understand the structure of code...") might lead to invalid diagrams or at least confuse merged diagrams. This is because such stories typically do not describe goals of a system. We observed that those types of stories usually do not comply to the INVEST principles and should be filtered out from the analysis of the product backlog.

Also, as with any visual representation, our technique is limited regarding the number of elements in one diagram. Our technique would not scale well for a large number of user stories as it would result in incomprehensible diagrams with many elements and relationships. However, to mitigate this problem, our technique supports viewpoint-based diagrams. In addition, even if a robustness diagram is not readable due to the large number of elements it contains, our technique provides information about related stories since we not only generate the diagrams, but also a list of stories relevant for a viewpoint (as `csv` files generated together with the diagrams).

6 CONCLUSIONS AND FUTURE WORK

This paper has presented a first-of-its-kind technique for automatically generating visual robustness diagrams from textual user stories. In contrast to previous approaches which generate visual models from more structured text or which generate structural diagrams, we generate robustness diagrams which combine structural and behavioural information from user stories into *one* diagram. Furthermore, we allow merging and "pruning" of diagrams to (a) show the bigger picture of user stories on the same backlog and to visualise commonalities in stories, and (b) to highlight behavioural and structural elements of stories from the perspective of a particular diagram element. Therefore, software engineers can identify reusable sub-processes or conflicts between stories more easily compared to performing a manual analysis of stories. In summary, our technique supports planning as well as development activities.

In our future work we will explore three main directions. First, we plan to expand the technique to provide analysts not only with a visual view of a backlog, but also with a text report of overlapping domain elements, common actors and processes. Such a report can be used as input into sprint planning sessions. Second, we will evaluate the usefulness of the generated robustness diagrams and reports in an agile context (note that the intrinsic usefulness of this notation has been discussed in prior work). Lastly, we will incorporate a quality evaluation framework in order to flag non-compliant or poorly written stories early.

REFERENCES

[1] Ian F. Alexander and Neil Maiden. 2004. *Scenarios, stories, use cases: through the systems development life-cycle* (1st ed.). Wiley Publishing.

[2] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. 2016. Extracting domain models from natural-language requirements: approach and industrial evaluation. In *ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. ACM, 250–260.

[3] Julian M. Bass. 2016. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology* 75 (2016), 1–16.

[4] Mike Cohn. 2004. *User stories applied: for agile software development.* Addison Wesley.

[5] Fabiano Dalpiaz, Ivor van der Schalk, Sjaak Brinkkemper, Fatma Basak Aydemir, and Garm Lucassen. 2019. Detecting terminological ambiguity in user stories: Tool and experimentation. *Information and Software Technology* 110 (2019), 3–16.

[6] Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal Stanford dependencies: a cross-linguistic typology. In *9th International Conference on Language Resources and Evaluation (LREC)*. ELRA, 4585–4592.

[7] Mohamed El-Attar and Hezam Akram Abdul-Ghani. 2016. Using security robustness analysis for early-stage validation of functional security requirements. *Requirements Engineering* 21, 1 (2016), 1–27.

[8] Mohamed El-Attar, Mahmoud Elish, Sajjad Mahmood, and James Miller. 2012. Is in-depth object-oriented knowledge necessary to develop quality robustness diagrams? *Journal of Software* 7, 11 (2012), 2538–2552.

[9] Mohamed El-Attar and James Miller. 2008. Producing robust use case diagrams via reverse engineering of use case descriptions. *Software & Systems Modeling* 7, 1 (2008), 67–83.

[10] Mohamed El-Attar and James Miller. 2010. Developing comprehensive acceptance tests from use cases and robustness diagrams. *Requirements Engineering* 15, 3 (2010), 285–306.

[11] Meryem Elallaoui, Khalid Nafil, and Raja Touahni. 2015. Automatic generation of UML sequence diagrams from user stories in Scrum process. In *10th International Conference on Intelligent Systems: Theories and Applications (SITA)*. IEEE, 1–6.

[12] Meryem Elallaoui, Khalid Nafil, and Raja Touahni. 2018. Automatic transformation of user stories into UML use case Diagrams using NLP techniques. *Procedia Computer Science* 130 (2018), 42–49.

[13] Mosa Elbendak, Paul Vickers, and Nick Rossiter. 2011. Parsed use case descriptions as a basis for object-oriented class model generation. *Journal of Systems and Software* 84, 7 (2011), 1209 – 1223.

[14] Nicolas Genon, Gilles Perrouin, Xavier Le Pallec, and Patrick Heymans. 2016. Unlocking visual understanding: towards effective keys for diagrams. In *International Conference on Conceptual Modeling (ER)*. Springer, 505–512.

[15] Sutirtha Ghosh, Prasenjit Mukherjee, Baisakhi Chakraborty, and Rezaul Bashar. 2018. Automated generation of E-R diagram from a given text in natural language. In *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)*. IEEE, 91–96.

[16] Fabian Gilson, Matthias Galster, and François Georis. 2019. Extracting quality attributes from user stories for early architecture decision making. In *3rd International Workshop on decision Making in Software Architecture (MARCH)*. IEEE, 129–136.

[17] Tony Gorschek, Ewan Tempero, and Lefteris Angelis. 2014. On the use of software design models in software development practice: An empirical investigation. *Journal of Systems and Software* 95 (2014), 176–193.

[18] David Harel and Bernhard Rumpe. 2004. Meaningful modeling: what's the semantics of "semantics"? *IEEE Computer* 37, 10 (2004), 64–72.

[19] Anne Hess, Philipp Diebold, and Norbert Seyff. 2018. Understanding information needs of agile teams to improve requirements communication. *Journal of Industrial Information Integration* 14 (2018), 3–15.

[20] Rashina Hoda and Latha Murugesan. 2016. Multi-level agile project management challenges: a self-organizing team perspective. *Journal of Systems and Software* 117 (2016), 245–257.

[21] M. G. Ilieva and O. Ormandjieva. 2006. Models derived from automatically analyzed textual user requirements. In *Fourth International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE, 13–21.

[22] Irum Inayat, Siti Salwah Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. 2015. A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior* 51 (2015), 915–929.

[23] ISO/IEC/IEEE. 2011. ISO/IEC/IEEE 42010: Systems and software engineering – Architecture description.

[24] Ivar Jacobson. 1987. Object-oriented development in an industrial environment. In *Object-oriented Programming Systems, Languages and Applications (OOPSLA)*. ACM, 183–191.

[25] Massila Kamalrudin, John Hosking, and John Grundy. 2017. MaramaAIC: tool support for consistency management and validation of requirements. *Automated Software Engineering* 24, 1 (2017), 1–45.

[26] Beum-Seuk Lee and Barrett R. Bryant. 2002. Automated conversion from requirements documentation to an object-oriented formal specification language. In *ACM Symposium on Applied Computing (SAC)*. ACM, 932–936.

[27] J. Lin, H. Yu, Z. Shen, and C. Miao. 2014. Using goal net to model user stories in agile software development. In *IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 1–6.

[28] Dongsheng Liu, Kalaivani Subramaniam, Behrouz Far, and Armin Eberlein. 2003. Automating transition from use-cases to class model. In *Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 831–834.

[29] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. 2016. Improving agile requirements: the Quality User Story framework and tool. *Requirements Engineering* 21, 3 (2016), 383–403.

[30] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn van der Werf, and Sjaak Brinkkemper. 2016. The use and effectiveness of user stories in practice. In *Requirements Engineering: Foundation for Software Quality (REFSQ)*. Springer, 205–222.

[31] Daniel Moody. 2009. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering* 35, 6 (2009), 756–779.

[32] Tuong Huan Nguyen, John Grundy, and Mohamed Almorsy. 2015. Rule-based extraction of goal-use case models from text. In *10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 591–601.

[33] Fouad Nasser A Al Omran and Christoph Treude. 2017. Choosing an NLP library for analyzing software documentation: a systematic literature review and a series of experiments. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 187–197.

[34] Marian Petre. 2013. UML in Ppactice. In *International Conference on Software Engineering (ICSE)*. IEEE, 722–731.

[35] Slav Petrov, Dipanjan Das, and Ryan T. McDonald. 2012. A universal part-of-speech tagset. In *International Conference on Language Resources and Evaluation (LREC)*. ELRA, 2089–2096.

[36] Francisca Ribeiro, André L. Ferreira, Anabela Tereso, and Deborah Perrotta. 2018. Development of a grooming process for an agile software team in the automotive domain. In *World Conference on Information Systems and Technologies (WorldCIST)*. Springer, 887–896.

[37] Doug Rosenberg and Matt Stephens. 2007. *Use case driven object modeling with UML: theory and practice.* Apress, Berkely, CA, USA.

[38] Eva-Maria Schön, Jörg Thomaschewski, and María José Escalona. 2017. Agile requirements engineering: a systematic literature review. *Computer Standards & Interfaces* 49 (2017), 79–91.

[39] Eva-Maria Schön, Dominique Winter, María José Escalona, and Jörg Thomaschewski. 2017. Key challenges in agile requirements engineering. In *International Conference on Agile Software Development (XP)*. Springer, 37–51.

[40] Todd Sedano, Paul Ralph, and Cecile Peraire. 2019. The product backlog. In *41st International Conference on Software Engineering (ICSE)*. IEEE, 200–211.

[41] Arjen Spaans. 2018. Automatic detection of requirements smells in user stories. Bachelor Thesis, Department of Informatics, Technishe Universität Müchen.

[42] Harald Störrle. 2017. How are conceptual models used in industrial software development?: a descriptive survey. In *21st International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 160–169.

[43] Jitendra Singh Thakur and Atul Gupta. 2014. Automatic generation of sequence diagram from use case specification. In *7th India Software Engineering Conference (ISEC)*. ACM, 20:1–20:6.

[44] Jitendra Singh Thakur and Atul Gupta. 2016. Identifying Domain elements from textual specifications. In *31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 566–577.

[45] Saurabh Tiwari, Santosh Singh Rathore, Abhijeet Singh, Abhinav Singh, and Atul Gupta. 2012. An approach to generate actor-oriented activity charts from use case requirements. In *19th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 270–275.

[46] Bill Wake. 2003. INVEST in Good Stories, and SMART Tasks. https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/ accessed on 12/09/2018.

[47] Yves Wautelet, Samedi Heng, Manuel Kolp, and Isabelle Mirbel. 2014. Unifying and extending user story models. In *25th International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 211–225.

[48] Y. Wautelet, S. Heng, M. Kolp, I. Mirbel, and S. Poelmans. 2016. Building a rationale diagram for evaluating user story sets. In *International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 1–12.

[49] Tao Yue, Lionel C. Briand, and Yvan Labiche. 2011. A systematic review of transformation approaches between user requirements and analysis models. *Requirements Engineering* 16, 2 (2011), 75–99.

[50] Tao Yue, Lionel C. Briand, and Yvan Labiche. 2015. aToucan: An automated framework to derive UML analysis models from use case models. *ACM Transactions on Software Engineering and Methodology* 24, 3 (2015), 13:1–13:52.