Automated Goal Model Extraction from User Stories Using NLP

Tuğçe Güneş
Boğaziçi University
Turkey
tugce.gunes@boun.edu.tr

Fatma Başak Aydemir
Boğaziçi University
Turkey
basak.aydemir@boun.edu.tr

***Abstract*—User stories are commonly used to capture user needs in agile methods due to their ease of learning and understanding. Yet, the simple structure of user stories prevents us from capturing relations among them. Such relations help the developers to better understand and structure the backlog items derived from the user stories. One solution to this problem is to build goal models that provide explicit relations among goals but require time and effort to build. This paper presents a pipeline to automatically generate a goal model from a set of user stories by applying natural language processing (NLP) techniques and our initial heuristics to build realistic goal models. We first parse and identify the dependencies in the user stories, and store the results in a graph database to maintain the relations among the roles, actions, and objects mentioned in the set of user stories. By applying NLP techniques and several heuristics, we generate goal models that resemble human-built models. Automatically generating models significantly decreases the time spent on this tedious task. Our research agenda includes calculating the similarity between the automatically generated models and the expert-built models. Our overarching research goals are to provide *i.* an NLP-powered framework that generates goal models from a set of user stories, *ii.* several heuristics to generate goal models that resemble human-built models, and *iii.* a repository that includes sets of user stories, with corresponding human-built and automatically generated goal models.**

***Keywords*— natural language processing, requirements engineering, model driven development, user stories, agile development, goal models**

I. Introduction

Describing requirements in natural language (NL) is the most common practice when handling software requirements [1]. Stakeholder requirements are elicited in natural language, and documenting them in NL requires less time and effort compared to other formal or semi-formal representations [2]. There is no one standard method to capture requirements, requirements can be captured in free, semi-structured, or structured formats.

One semi-structured natural language notation is *user stories*, which are widely used in agile development processes [2], [3]. The most common standard template for user stories is "As a <role>, I want <action>, so that <benefit>" [4], which is easy to learn and apply. These qualities of user stories make them suitable for agile development and are the main reasons for their adoption by the practitioners.

One challenge that is not addressed by user stories is to capture the structure of the user needs. A flat list of user stories does provide neither the *raison d'être* of the user stories nor how they are related to each other. Filtering and sorting operations can provide a primitive overview, but in general it is difficult to have a high level view of the relations, identifying the user stories that are about the same concepts, or states the same benefits.

A vast number of publications have already demonstrated that goal models can be used to capture the structure of a set of requirements and many different relations [5]. The goals of actors can be visualized within the actor's boundaries [6], high and low-level goals are related through refinements, and several inter-dependencies such as cost and value contributions can be defined in a goal modeling language [7]. Goal models are also used for analysis and optimization (e.g. [8]). Despite the wide research literature on goal models, their adoption by industry is limited. The practitioners find the learning curve high, and it is a common perception that building goal models requires time and effort which makes them costly.

We aim to support the agile development processes that rely on user stories by helping them benefit from the advantages goal modeling provides. In this paper, we propose a natural language processing (NLP) pipeline for automatically generating and visualizing goal models that resemble human-built models from user stories and present our research agenda and initial heuristics.

This paper is structured as follows. Section II surveys the related work. Section III details our pipeline and explains the model generation heuristics and algorithms. Section IV reveals our research plan and Section V concludes the paper.

II. Related Work

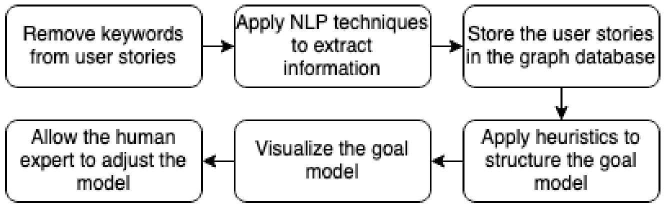
Automated model generation from text documents using NLP is an open research area. For example, Sanyal and Kumar [9] present the SUGAR tool, that generates both use case and class diagrams from NL requirements using both NLP technologies and syntactic rules. Deeptimahanti and Babar [10] describe a technique for generating UML models from complex NL requirements. The authors introduce a tool named UML Model Generator from Analysis of Requirements (UMGAR), that is powered by NLP tools including Stanford Parser[1] and WordNet[2] to deal with large requirements documents. More and Phalnikar [11] implement the RAPID tool to extract UML diagrams from NL specifications using NLP and domain ontology techniques. Similar to previous work by Herchi and Abdesselam [12], they aim to automate the transformation of textual user requirements to UML class diagrams using NLP and domain ontology. The approach finally applies some linguistic rules to capture concepts like class names, their attributes, and associations to draw UML. Zhou and Zhou [13] offer a method for generating class diagrams automatically from free-text requirements adopting both NLP tools and domain ontology techniques to refine the class identification tasks.

Ibrahim and Ahmad [14] implement the RACE tool to translate user requirements expressed in NL into class diagrams which is quite similar to RAPID tool since it uses NLP and domain ontology knowledge. Letsholo et al. [15] provide the TRAM tool for automatically building analysis models from the natural language specifications by applying conceptual patterns on top of NLP techniques. Also the system enables to contact expert modeler to improve the quality of the model. Abdessalem et al. [16] propose an automated tool for UML class diagrams generation from NL requirements using NLP techniques like pattern matching to extract the class elements.

As agile development methodologies gain momentum in the software industry, user stories draw the attention of researchers. Robeer et al. [17] introduce a fully automated tool to derive a conceptual model from user stories using natural text processor tool spaCy[3]. In another work, Lucassen et al. [18] propose Visual Narrator tool using python and Natural Language Tool Kit (NLTK) to extract conceptual models from user stories. Elallaoui et al. [19] propose an automated transformation of user stories into UML use case diagrams by using NLP tools. This research differs from other studies since it analyzes user stories whereas others mostly work on requirements documents while producing UML diagrams.

Elallaoui et al. [20] define an algorithm that takes user stories contained files and produces XML files which are transformed into sequence diagrams using UML2 tool SDK plugin for Eclipse. Mesquita et al. Lucassen et al. [21] investigate possible solutions for applying algorithms that have semantic knowledge between concepts to visualize user stories by designing a Visual Narrator tool to extract conceptual models from user stories. Arora et al. [22] propose an automated process of constructing domain models from NL requirements based on NLP tools benefiting from existing rules and also offer new rules using NLP dependency parser.

There has been some work on user stories to goal model transformation. [23] develop a tool US2StarTool to automate the generation of iStar models from user stories. Lin et al. [24] introduce a goal-oriented method to model goal requirements from user stories. The method Goal Net indicates goal structure as from low-level user stories to high-level goals. Wautelet et al. [25]–[28] build rationale models for user story sets. These publications do not analyze the smaller linguistic components in user stories, they rather focus on slicing it into three main parts (actor, action, benefit). The difference between their research and ours is that we plan to have a finer granularity and do not generate goals for the whole action or benefit part of a user story but identify objects and verbs and connect them.


Remove keywords from user stories
Apply NLP techniques to extract information
Store the user stories in the graph database
Apply heuristics to structure the goal model
Visualize the goal model
Allow the human expert to adjust the model
Figure 1: Pipeline to generate goal models from user stories

We aim to generate realistic goal models, so our evaluation plan includes comparing automatically generated models with human-built models by measuring the similarities among them. Measuring graph similarity is an active research question. Raymond et al. [29] introduce a strategy for comparing labeled graphs with setting a minimum similarity threshold. The similarity is measured by using a maximum common edge subgraph (MCES) detection algorithm. Koutra et al. [30] propose DELTACON to measure the connectivity between two graphs which tackles the problem when the node correspondence is known and the approach covers measuring the overlap of graphs' edges. In another work by Koutra et al. [31] design a new framework for determining the degree of graph similarity using belief propagation and related ideas. They work on two graphs that have the same set of nodes with different edges. Zheng et al. [32] indicate that measuring the graph similarity over large graph databases are not efficient. Therefore they propose to retrieve graphs if they are similar to the given query graph defines a systematic method for edit-distance based similarity problem.

III. Generating Goal Models from User Stories

In this work, we aim to construct a goal model from a set of user stories. We use regular expressions and NLP techniques to extract who (role), what (action, object), and benefit information from user stories. We propose heuristics to combine this information in different ways to build goal models that resemble human-built models. We then visualize the produced model in a dynamic, editable format to allow human experts to adjust them if necessary. Fig.1 summarizes our process.

Our initial step towards building a goal model from a set of user stories is parsing each user story with NLP techniques. NLP starts with taking the semi-structured user stories as input and extract the necessary concepts and relations for our model design. The structure of a user story can be divided into three parts. The first part indicates the type of the role in the user

[1]https://nlp.stanford.edu/software/lex-parser.shtml
[2]https://wordnet.princeton.edu/
[3]https://spacy.io/

story. The second part declares the action of the user story and finally, the third part includes the benefit of realizing the user story [25]. Fig. 2 provides an example user story where the italicized words are the components of the user story structure.

As a user, *I want* to click on the address, *so that* it takes me to a new tab with Google Maps.

Figure 2: An example user story

The first step of our pipeline is to remove the special keywords of the template from each user story using regular expressions. We split each user story into three parts by using *'as a'*, *'want to'* and *'so that'* keywords as they construct the template of user stories. After this cleaning, the next step is to tokenize and apply part-of-speech (POS) tagging. We use the spaCy tool for these tasks. Fig. 2 contains an example user story as the input of the pipeline. We require the role, the verb of the action, and object in each user story to collect the information necessary to construct the goal model. In first element of user story: `<user>` is the type of role having `NOUN` tag, `<click>` is the main verb having `VERB` tag and lastly `<the address>` is the main object having `NOUN` tag (an object can also be a noun phrase with the `NP` tag). Fig. 3 demonstrates the output of the first two steps of our pipeline on this input.

For example, the user story mentioned in Fig. 2 is split into parts and then POS tagged in Fig. 3. As we focus on building a goal model, our attempt is deriving the goals from user stories. Therefore we require a type of role, action verb, and object in each user story to gain goal information. In the first element of a user story: <user> is the type of role having NOUN tag, <click> is our main verb having VERB tag and lastly <the address> is our main object having NOUN tag.

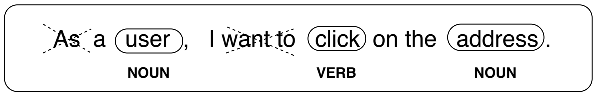


Figure 3: Part of user story with pos tags

When it comes to storing extracted knowledge, we benefit from graph databases to easily observe the connections between different elements. Graph databases are effective when handling data relationships. A node and a relationship are two units of graph databases which allows to create nodes and associations between them simultaneously. We choose **Neo4j** [4] for the implementation due to its support for handling multiple formats as input and expressive query language *Cypher*. *Cypher* has essential concepts and clauses of Structured Query Language (SQL). When applying the heuristics, we query the database with semantic queries to understand the relations among the nodes and visualize the inter-connected data.

Next, we develop basic heuristics to build realistic goal models. We observe that these heuristics are too primitive for this, and we aim to combine them to effectively construct more complex structures within our research agenda.

Heuristic 1: Grouping Similar Verbs This heuristic generates parent nodes from verbs of action of the user stories, and child nodes using the different objects associated with the roles and these verbs.

Consider a set S that consists of three user stories (examples used in this section are taken from [33]):

s1: As a user, I want to be able to view a map display of the public recycling bins around my area.

s2: As a user, I want to be able to view the safe disposal events currently being organised around my area.

s3: As a user, I want to view all locations of recycling centers on a map, so that I can check which routes to take to drop off waste.

These user stories have a common role, `user`, and a verb, `view`, and each has a different object: `map display`, `safe disposal events`, and `recycling centers`, respectively (due to space limitations we use the short versions of the noun phrases). The corresponding goal model is provided in Fig. 4.

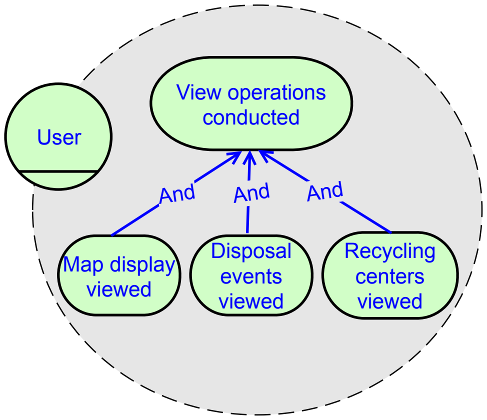


Figure 4: Goal model design with Heuristic 1

Algorithm 1 describes this process. It takes a graph database populated from the set of user stories as input. The graph database includes the role nodes, their corresponding verb nodes, and their corresponding object nodes and can be queried in several ways, such as the verbs of a certain role. For each role in the graph database, the algorithm creates an actor node in the goal model (Line 2). Then, it queries the graph database to get the verbs associated with this role (Line 3). Next, it creates lists of verbs where each list includes similar verbs (we use the similarity scores from spaCy. For example, view and see are considered similar as their score is above a certain threshold. Line 4). For each of these lists, the algorithm creates a parent node within the actor boundaries (Line 6). Our current strategy is to use the template "`verb` operations conducted" for the label of this node. Then, for each verb in the list, the algorithm queries the graph database for the objects associated with the role and this particular verb (Line 8). It creates a child node by combining the verb and the object

[4]https://neo4j.com/

Algorithm 1 Grouping verbs by similarity

Input: a graph database USG of user stories
Output: a goal model *G* for user stories

```
 1: for each role r in USG do
 2:     create an actor for r in G
 3:     VL = Verbs of r
 4:     SVL = GroupSimilarVerbs(VL)
 5:     for each list verbtree in SVL do
 6:         Create a parent goal for the list
 7:         for each v in verbtree do
 8:             OL is the list of objects connected to v and r
 9:             for each o in OL do
10:                 create a child goal combining v and o in G
11:                 link the child node with its parent in G
12:             end for
13:         end for
14:     end for
15: end for
16: return G
```

(Line 10) and links it to the parent goal (Line 11). It is not shown in the algorithm due to space limitations but the users can specify the default type of refinements as AND or OR. It returns the structure of the goal model (Line 16). In our implementation we return a JSON file.

Heuristic 2: Grouping Similar Objects The second heuristic generates parent nodes from the objects of user stories. The child nodes of these parents are the verbs that are associated with them. Each object tree is created within the actor boundaries of a role.

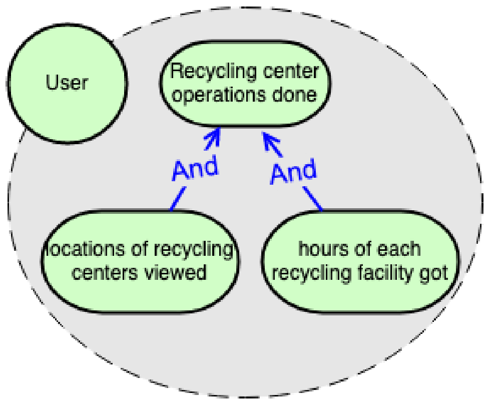
Consider a set S that consists of three user stories:

s1: As a user, I want to view all locations of recycling centers on a map, so that I can check which routes to take to drop off waste.

s2: As a user, I want to be able to get the hours of each recycling facility, so that I can arrange drop-offs on my off days or during after-work hours.

These user stories have a common role, `user`, however different verbs and objects. The verbs are `view`, and `get`; and the objects are `locations of recycling centers`, and `hours of each recycling facility`. The resulting goal model is presented in Fig. 5.

Algorithm 2 describes the transformation from the set of user stories to the goal model. Similar to Algorithm 1, it takes a graph databases populated from the set of user stories as input. For each role in the graph database, the algorithm creates an actor node in the goal model (Line 2). Then, it queries the graph database to get the objects associated with this role (Line 3). Next, it creates lists of objects where each list includes similar objects (we use the similarity scores from spaCy for phrases. Line 4). For each of these lists, the algorithm creates a parent node within the actor boundaries (Line 6). Our current strategy is to use the template "`object` operations done" for the label of this node. Then, for each object in the list, the


User
Recycling center operations done
And
And
locations of recycling centers viewed
hours of each recycling facility got
Figure 5: Goal model design with Heuristic 2

Algorithm 2 Grouping objects by similarity

Input: a graph database USG of user stories
Output: a goal model *G* for user stories

```
 1: for each role r in USG do
 2:     create an actor for r in G
 3:     OL = objects associated with r
 4:     SOL = GroupSimilarObjects(OL)
 5:     for each list objecttree in SOL do
 6:         Create a parent goal for the list
 7:         for each o in objecttree do
 8:             VL is the list of verbs connected to o and r
 9:             for each v in VL do
10:                 create a child goal combining o and v in G
11:                 link the child node with its parent in G
12:             end for
13:         end for
14:     end for
15: end for
16: return G
```

algorithm queries the graph database for the verbs associated with the role and this object (Line 8). It creates a child node by combining the verb and the object (Line 10) and links it to the parent goal (Line 11) as in Algorithm 1. Finally, it returns the structure of the goal model (Line 16).

Heuristic 3: Getting rid of actors Consider a set S that consists of three user stories:

s1: As an executive, I want to have full access to data related to my company, so that I can have a sense of my company's performance.

s2: As an employee from the HR department, I want to modify the data of the company.

s3: As an employee, I want to view the data of the company.

These user stories have the same or similar objects, `data of the company`. Their roles are different: `executive`, and `employee`. They also have different verbs, namely, `access`, `modify`, and `view`. Fig. 6 presents the resulting goal model.

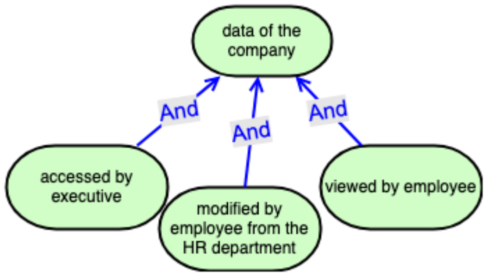


Figure 6: Goal model design with Heuristic 3

Heuristic 3 is similar to Heuristic 2 in the sense that it creates subtrees of objects, yet this time the child node labels include the role information as well and the goal model does not include any actor boundaries.

Algorithm 3 Getting rid of the actors

```
Input: a graph database USG of user stories
Output: a goal model G for user stories
 1: OL is the list of all objects in USG
 2: SOL = GroupSimilarObjects(VL)
 3: for each list objecttree in SVL do
 4:     Create a parent goal for the list
 5:     for each o in objecttree do
 6:         VL is the list of verbs connected to o
 7:         for each v in VL do
 8:             RL is the list of roles connected to o and v
 9:             for each r in RL do
10:                 create a child goal combining r and v in G
11:                 link the child node with its parent in G
12:             end for
13:         end for
14:     end for
15: end for
16: return G
```

Algorithm 3 starts by extracting the objects from the graph database (Line 1), and group them according to their similarity (Line 2). For each group, it creates a parent node (Line 4). For each element in a given group, it processes all the verbs (Line 6), and for each verb, it retrieves the related roles (Line 8). The algorithm creates a child node for each role and verb combination (Line 10), and links it to the parent goal (Line 11). At the end, it returns the structure of the goal model (Line 16).

The next step in the pipeline is to transform the output file in the JSON format to an editable visualization. Currently we are working on a browser-based modeling editor that parses the JSON file which has the structural information and automatically generates a layout for the goal model using the joint.js[5] JavaScript library.

[5]www.jointjs.com

IV. Research Plan

So far we have implemented three basic heuristics and are planning to expand our basic heuristics library by adding heuristics related to the benefit component of the user stories. Our current work also includes the implementation of the interactive modeling editor. Our first milestone is to integrate the editor with the whole pipeline to have a minimum viable product. It is important to reach this milestone before proceeding with the next steps as it will be used for the evaluation of the heuristics.

The first evaluation planned with goal modeling experts. During the experiments, the experts will be given a set of user stories and will be asked to build a goal model for the set. Then, the experts will be shown the output of the heuristics and their qualitative feedback will be collected. They will be asked to improve the automatically generated models. Quantitative feedback on the usability of the editor and perceived quality of the generated models as well as the qualitative feedback on the automatically generated models will be collected.

The original and the modified models by the experts, and their qualitative feedback will be used to devise new heuristics and strategies to combine the existing ones to produce better models. The editor will also be improved if necessary.

Our second phase of evaluation will be the replication of the first one where we collect the qualitative and quantitative feedback of goal-model experts. Based on their feedback we plan to do the final adjustments on our heuristics.

In our final round of evaluation, our first goal is to measure the similarity of the human-built models and the automatically generated ones. In this experiment, the experts will be asked to build a goal model from a set of user stories. We will measure the similarity between the graphs, and collect the perceived similarity, and quality of models from the experts. Our second goal is to understand whether our tool supports the agile practitioners. We plan to conduct a case study at a software development company where agile methods are applied for multiple sprints and measure the perceived usefulness of the tool and whether they would incorporate it into their workflow. We are also curious to see if using the tool will decrease the time spent to build goal models for academic use. We plan to conduct an experiment where the control group uses the tool starting with a blank canvas and builds a goal model from a set of user stories, and the treatment group starts with the auto-generated model and the tool, and modifies it. We will measure the time spent to complete the model by two groups. Fig. 7 summarizes our research agenda.

V. Conclusion

This paper presents the initial work and the research agenda for an NLP-powered, rule-based tool to generate and modify goal models from a set of user stories. Our motivation is to bring the advancements of research of goal models to the industry by facilitating building goal models.

To reach our research goals, we present the initial basic heuristics to combine pieces of information extracted from user stories by applying NLP techniques and stored in a graph

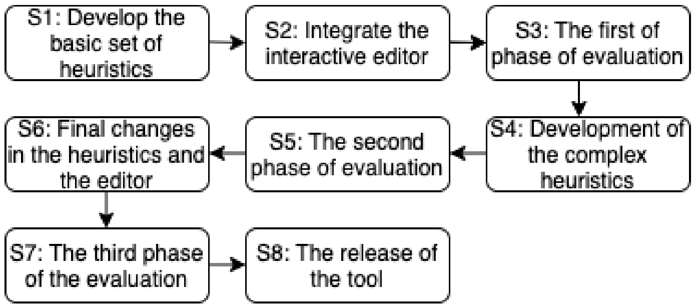

S1: Develop the basic set of heuristics
S2: Integrate the interactive editor
S3: The first of phase of evaluation
S4: Development of the complex heuristics
S5: The second phase of evaluation
S6: Final changes in the heuristics and the editor
S7: The third phase of the evaluation
S8: The release of the tool

Figure 7: Summary of our research agenda

database. Existing literature on generating goal models from user stories does not analyze the goal models in the same detail as we propose and remain in the level of three basic components: why, what, and who. We also layout an extensive plan for the evaluation of our tool where each evaluation phase provides feedback for the next iteration of development.

Our contributions are the set of heuristics to generate goal models from user stories, and the collected human-built models which can be used for other research. If enough models are collected, machine learning techniques can be employed to build systems that directly learn from the human experts in the future.

Acknowledgments This work has been partially supported by the Scientific and Technological Research Council of Turkey through BIDEB 2232 grant 118C255, Requirements Engineering for Digital Transformation (RE4DigiTR).

References

[1] M. Kassab, C. Neill, and P. Laplante, "State of Practice in Requirements Engineering: Contemporary Data," *Innovations in Systems and Software Engineering*, vol. 10, no. 4, pp. 235–241, 2014.

[2] M. Kassab, "The Changing Landscape of Requirements Engineering Practices Over The Past Decade," *EmpiRE*, pp. 1–8, 2015.

[3] G. Lucassen, F. Dalpiaz, J. M. E. M. v. d. Werf, and S. Brinkkemper, "The use and effectiveness of user stories in practice," *Springer Int'l Pub*, pp. 205–222, 2016.

[4] M. Cohn, *User Stories Applied: for Agile Software Development*. Redwood City, CA, USA: Addison-Wesley Professional, 2004.

[5] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, and P. Giorgini, "Goal-oriented requirements engineering: an extended systematic mapping study," *Requir Eng*, vol. 24, no. 2, pp. 133–160, 2019.

[6] F. Dalpiaz, X. Franch, and J. Horkoff, "istar 2.0 language guide," *arXiv preprint arXiv:1605.07767*, 2016.

[7] F. B. Aydemir, F. Dalpiaz, S. Brinkkemper, P. Giorgini, and J. Mylopoulos, "The next release problem revisited: a new avenue for goal models," in *RE*. IEEE, 2018, pp. 5–16.

[8] C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Multi-objective reasoning with constrained goal models," *Requir Eng*, vol. 23, no. 2, pp. 189–225, 2018.

[9] D. K. Deeptimahanti and R. Sanyal, "Static UML Model Generator from Analysis of Requirements (SUGAR)," *In Advanced software engineering and its applications (ASEA)*, 2008.

[10] D. K. Deeptimahanti and M. A. Babar, "An Automated Tool for Generating UML Models from Natural Language Requirements," *Automated Software Engineering*, pp. 680–682, 2009.

[11] P. R. More and R. Phalnikar, "Generating UML Diagrams from Natural Language Specifications," *International Journal of Applied Information Systems*, 2012.

[12] H. Herchi and W. Abdessalem, "From User Requirements to Uml Class Diagram," *in CoRR abs/1211.0713*, 2012.

[13] N. Zhou and X. Zhou, "Automatic Acquisition of Linguistic Patterns for Conceptual Modeling," *Course INFO 629: Concepts in Artificial Intelligence*, 2004.

[14] M. Ibrahim and R. Ahmad, "Class Diagram Extraction from Textual Requirements Using NLP Techniques," *IOSR-JCE*, pp. 27–29, 2015.

[15] K. J. Letsholo, L. Zhao, and E.-V. Chioasca, "TRAM: A tool for transforming textual requirements into analysis models," in *In Proceedings of ASE*, 2013.

[16] W. Abdessalem, Z.B.Azzouz, A.Singh, N.Dey, A.S.Ashour, and H.B.Ghezala, "Automatic Builder of Class Diagram(ABCD): an Application of UML Generation From Functional Requirements," *Journal of Software Practice and Experience*, 2016.

[17] M. Robeer, G. Lucassen, J. M. E. M. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated Extraction of Conceptual Models from User Stories via NLP," *RE'16*, pp. 196–205, 2016.

[18] G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Extracting conceptual models from user stories with Visual Narrator," *Requir Eng*, vol. 22, no. 3, pp. 339–358, 2017.

[19] M. Elallaoui, K. Nafil, and R. Touahni, "Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques," *Procedia Computer Science*, vol. 130, pp. 42–49, 2018.

[20] ——, "Automatic generation of UML sequence diagrams from user stories in Scrum process," *International Conference on Intelligent Systems: Theories and Applications*, 2015.

[21] G. Lucassen, F. Dalpiaz, J. M. E. M. v. d. Werf, and S. Brinkkemper, "Visualizing User Story Requirements at Multiple Granularity Levels via Semantic Relatedness," *ER*, 2016.

[22] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Extracting Domain Models from Natural-Language Requirements: Approach and Industrial Evaluation," in *in Proc. ACM/IEEE 19th MoDELS*, 2016.

[23] R. Mesquita, A. Jaqueira, C. Agra, M. Lucena, and F. Alencar, "US2StarTool: Generating i* Models from User Stories," *International i* Workshop (iStar)*, 2015.

[24] J. Lin, H. Yu, Z. Shen, and C. Miao, "Using goal net to model user stories in agile software development." 15th IEEE/ACIS International Conference on SNPD, Jun 2014.

[25] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, "Unifying and Extending User Story Models." CAiSE, Jun 2014, pp. 211–225.

[26] Y. Wautelet, S. Heng, M. Kolp, I. Mirbel, and S. Poelmans, "Building a rationale diagram for evaluating user story sets," *RCIS*, 2016.

[27] Y. Wautelet, S. Heng, D. Hintea, M. Kolp, and S. Poelmans, "Bridging user story sets with the use case model," *In: Link S, Trujillo JC (eds) Proceedings of ER workshops*, 2016.

[28] Y. Wautelet, S. Heng, S. Kiv, and M. Kolp, "User-story driven development of multi-agent systems: A process fragment for agile methods," *Computer Languages Systems & Structures*, 2017.

[29] J. W. Raymond, E. J. Gardiner, and P. Willett, "RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs," *The Computer Journal*, vol. 45, no. 6, pp. 631–644, 2002.

[30] D. Koutra, J. T. Vogelstein, and C. Faloutsos, "DELTACON: A Principled Massive-Graph Similarity Function," *International Conference in Data Mining (SDM)*, 2013.

[31] ——, "Algorithms for Graph Similarity and Subgraph Matching," *Ecological Inference Conference*, 2011.

[32] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao, "Efficient Graph Similarity Search Over Large Graph Databases," *TKDE*, 2014.

[33] F. Dalpiaz, "Requirements data sets (user stories)," http://dx.doi.org/10.17632/7zbk8zsd8y.1, 2018.