Towards Complementing User Stories

Christian Kop
Institute of Applied Informatics
Alpen-Adria-Universitaet Klagenfurt
Klagenfurt, Austria
chris@ifit.uni-klu.ac.at

***Abstract*—User stories are well established in agile software development processes. However, user stories should not be seen as detailed requirements specifications. In agile processes it is accepted that the end users do not know all the requirements at once. Therefore, user stories only give hints about the expectations of an end user. In order to get more details in a communications process, a computer supported strategy is proposed in this paper. This strategy focuses on the agile development of information systems. Namely, it is proposed that additional information (not found in a user story) is extracted from natural language queries. Afterwards, this information is compared with the already existing work in progress model. Thus, the natural queries should help to find gaps and misunderstandings in the current work in progress model.**

Keywords-agile process; user stories; domain models; natural language queries;

I. INTRODUCTION

In agile processes, user stories are a common way to gather the necessary information from the end user (e.g., an on-site customer). In a user story, the end user typically specifies what a certain actor (i.e., a system or person, which plays a role with respect to the system under development) can do with the system. Though, user stories are a well established technique in the early phase of agile software development, a user story is not a finished and well defined requirement or written contract. Instead, user stories are seen as short description of a piece of functionality, which act as a reminder for a communication process between end users and developers. In this process, details have to be negotiated [27]. So, the question is: How can this communication and negotiation process look like? Usually, developers can use questionnaires to ask for further information. They can make observation. If a first prototype is already developed, the communication is based on the prototype implementation. The software developers might ask what end users can do wrong if they are in a certain state of the prototype.

For information system development (ISD) an additional information gathering technique will be introduced here. Namely, natural language queries will be used. It will be shown how both steps

- extraction of model information from user stories and
- extraction of additional details from queries

can be supported by a tool.

The paper is therefore structured as follows. Firstly, an overview of the related work is given (Section II). Afterwards, an overall description is given, of how an agile process can be achieved in domain modeling. The next two sections (Section IV and Section V) focus on the two important parts of this agile process (user stories and natural language queries) and focus on computer supported information extraction. In Section VI, it will be argued, that this approach fits into the Agile Software Development Manifesto. Afterwards, it will be described how the computer support was tested. Finally, this section gives an overview of the prototype implementation. In Section VII conclusions and an outlook to future work are drawn.

II. RELATED WORK

Apart from user stories, some other research fields must be considered in the context of this work. These research topics will be described in the following sections.

A. Quality of Conceptual Models

The validation of artifacts in requirements engineering is always based on several techniques like inspections, walkthroughs, scenarios, verbalization, prototype evaluation, [16][21] or colored Petri Nets [22].

In [17] three dimensions for conceptual model quality are defined. They are syntax, semantics and pragmatics. If the model follows the rules and grammar defined in its meta-model then it is syntactically correct. Semantic quality is given if the model only contains true statements of the domain and is complete (no important concepts or statements are missing). Lastly, pragmatic quality relates the model to the interpretation of the user. A pragmatic quality is given, if it is understandable to the user.

An extensive research on quality of models was also made in [19]. It was proposed that conceptual modeling must shift from an art to an engineering discipline. Any engineering discipline aims at continuous quality checks of products and intermediate products.

There are also many other research results how to check and improve model quality [1], [4], [5], [8], [18]. The several research activities focused on model transformations [1], graphical aspects on conceptual models [18], verbalization strategies of conceptual models [4] [8] and viewpoints [5].

 ISBN: 978-1-61208-165-6

B. Queries

Rumbaugh et al. use queries for checking model completeness [35]. In their book [35], the authors give the exercise to check a partially completed object model with natural language queries. However, no computer support was found for this kind of task.

Current research on natural language query processing [2], [9], [12], [14], [13], [20], [23], [24], [15], [7],[26] only focuses on the retrieval of data or the generation of SQL. That means, these approaches expect that a finished and stable database already exist. A work in progress model and the consequences for natural language query processing was not the focus of these approaches. In these approaches, it is thus not the task of the natural language query to validate if something in the model is missing. Research works that describe visual queries [3], [10], [11] and form based query languages [6], [25] are based on the same assumption (i.e., existing and finished conceptual model or database). In visual query languages an SQL statement is generated by navigating through a final conceptual model.

C. Test Driven Development

Test driven development [31] is another agile method. The main idea is to write a first test case before a requirement is implemented. Afterwards, the developer tries to develop an implementation, which can pass the test case. Both the test cases and implementations to successfully pass the test cases are refined and improved iteratively in test driven development. The paradigm behind this is pointed out by Kent Beck: "**Failure is progress**" (see [31] p. 5).

User stories and natural language queries match very well to this paradigm. User stories represent the initial expectations ("requirements") of the end users. Natural language descriptions represent the test cases.

D. Linguistic Instruments

In the succeeding sections and sub sections the linguistic instruments tagging and chunking are needed. A tagger is a tool, which takes as input a text and returns a list of sentences with tagged (categorized) words (i.e., words categorized as noun, verb, adjective etc.). The chosen Stanford Tagger categorizes the words according to the Penn-Treebank Tagset [32]. In this tagset the word categories together with some important syntactical features of a word are encoded. For instance if a noun is in plural then the category NNS is chosen. If a proper noun is detected then NNP is used. Current taggers can achieve about 97 % percent correctness [30]. This means, that in at least 3% of the categorization cases, a word is wrongly categorized. This has to be considered if a tagger is used.

Chunking is based on the tagger result. Chunking is useful to group words to so called chunks that can be seen as a phrase (e.g., a verb phrase or a noun phrase). This grouping is based on patterns found in the preceding tagging result. For instance the following groups of word categories can be subsumed to a noun phrase: Noun + noun; article + noun; article + adjective + noun. Details of chunking are described in [32].

III. Overview of the Agile Development Scenario

Before details of the domain concepts extraction and the completion process will be explained, this section gives an overview. The main idea is to combine user stories and natural language queries. Whereas user stories are needed to get a first initial model, natural language queries are used as test cases.

A. User Stories

A user story is a small piece of text. According to [27] it describes a functionality that will be valuable to either an end user or purchaser of a system or software. A user story follows a certain pattern. Currently two patterns are discussed and used. The first pattern is a declarative sentence in active voice that follows the SPO (subject, predicate, object) style. Examples for the first pattern are: "A user can fill out a resume"; "A customer can place an order". A possible tool support for such a pattern is mentioned in [36]. The second pattern [34] emphasizes that in a user story sentence an actor is involved. Therefore, the pattern looks as follows "As a(n) <role/actor> , I can <feature>. The above examples would look as follows in the second style: "As a user, I can fill out a resume"; "as a customer, I can place an order". With the words "as a", the speaker makes it more explicit, that with customer or user respectively not a concrete thing inside the system is described. Instead, the role of an external thing with respect to the future system is defined. Although, user stories are well accepted in agile processes, they provide minimal information. The developer must either strongly communicate with the end user or he has to rely on his personal experiences in a certain domain. Since it is always good that an end user is involved, the focus of agile development processes is on communication! In this paper a natural way of doing this in the area of information system development (ISD) and data centric applications is proposed. Namely, additional natural language queries should help to get more information about the model.

B. Completing the Story with Natural Language Queries

User stories help to get a first impression of what the user wants. For instance, for the user story "a customer can place an order" the following information can be extracted:

- The noun "customer" might be an actor in this story
- The noun "order" might be the domain class
- The verb "to place" is an operation or service, which probably belongs to order.

In agile processes, the developer now has to design and implement this user story. In order to implement this properly, additional information is necessary. For instance: Which attributes does order have? Can the notion customer be treated only as an actor or can it be treated as a full domain class? Usually this information is collected in a communication process with an end user. The question now is: Can this communication process be somehow supported? Here, it is proposed to use natural language queries. Continuing the given scenario with the user story "A customer can place an order". The developer can now ask the end user, which queries should be executed later on in the

 ISBN: 978-1-61208-165-6

information system. Particularly, he should ask, which queries will be needed later for a database table, which is currently only represented as the domain class “order”. Since it cannot be expected that the end user knows SQL, natural language queries should be used instead.

Now let us suppose that only “order” is currently collected as a domain class. At the moment, customer is only seen as an (external) actor, which can communicate with the system. Furthermore let us assume that the end user states the following natural language queries:

- Which customer has placed Order 123
- Tell me the order items that belong to an order.

From these queries, the following can be learned.
Customer is not only an actor. Since he is mentioned in a query, which will be later on used as a database query, customer information is also needed in the database. In other words the information about customers represented by the word customer must be also modeled as a domain class.
Since both words “customer” and “order” are mentioned in the query, a path between them must exist. In its simplest form an association between customer and order must be modeled. Not any order is mentioned in the first query but a special order, which has a certain value. Since order is a class and values are instances of attributes the developer must ask the right attribute. An answer of the end user can be a revised and improved query (e.g., which customer has placed the order with order number 123). With this information the first domain model, which only contains “order” can be extended. Order gets the attribute order number. Furthermore “customer” is inserted as a domain class in the model. An association can be created between order and customer. The same procedure is applied on the second query. From this query, the developer can derive the information, that there will be a domain concept (i.e., the domain class “order item”), which can be related to order. Hence, the initial model, which had only “order” as its model element is iteratively refined.

C. Summary of the Overview

In the above two subsections it was explained how an initial domain model was generated using a user story. Since, such a model is still very incomplete; natural language queries can be used to complete it.

In the next Sections IV and V it will be shown how certain needed information (e.g., concepts) can be extracted automatically from user stories and natural language queries. Section IV focuses on the automatic extraction of actors and domain concepts (domain classes and attributes) from user stories. Section V will focus on the domain concept extraction from natural language queries.

IV. Extracting Concepts From User Stories

Domain concepts can be extracted from user stories by using the linguistic instruments tagging and chunking.

A. Tagging

At the first level, a tagger analyzes the user story text. The several word tokens are analyzed. Since taggers do not work 100 % correctly and a failure rate of 3 % must be considered, the result of the tagger is analyzed once again for failures. This is done by broadening the context window. In this step a certain categorized word is compared with its previous and its succeeding neighbors. If a certain pattern appears, which can be seen as a wrong combination of word categories (i.e., the tagger has detected a noun but in this context a verb is the correct categorization), then the categorization is changed.

B. Chunking

After the tagging step, the chunking reanalyzes the tagger output. Chunking subsumes certain combination of categorized words (e.g., noun + noun; article + noun; article + adjective + noun) to noun phrases. Chunking helps to reduce the pattern variations and therefore is a good basis for the next step (interpretation).

C. Interpretation

In the interpretation step the concepts are extracted from the linguistically analyzed user story. The SPO pattern that is introduced in Section III looks like follows with the support of the chunker: <noun phrase> <verb phrase> <noun phrase>. After chunking, the second mentioned pattern (“as a <actor/role>, I can <feature>”) follows the linguistic pattern: <preposition> <noun phrase>, <personal pronoun> <verb phrase> <noun phrase>. The interpretation collects the noun phrases of the sentence. Since a user story itself is based on patterns, the interpretation can consider this for noun phrase selection and categorization. The first <noun phrase> is always treated as the actor/role that is mentioned in the user story. The second <noun phrase> contains the domain concept (class or attribute). The domain concept and the actor respectively are extracted from the noun phrases by ignoring the article.

V. Extracting Concepts from Natural Language Queries

The query analyzer that extracts concepts from natural language queries is also based on the linguistic instruments tagging and chunking. Upon these, an interpretation and matching component is built.

A. Tagging

The same tagger that is used for analyzing user stories, is also used for analyzing the queries. In addition to the general optimizations which were introduced, the query tagger also have some additional optimization rules. These rules are necessary since query sentences might start with a verb. Furthermore, noun phrases are more complex than noun phrases which appear in the user story patterns.

B. Chunking

The chunker module has also an extra mode for query sentences. If this mode is switched on, one exception exists regarding the grouping of words and word categories. If the words “many” or “much” follow the word “how” (e.g., “how many persons”) then the implemented chunker behaves slightly different. A word like “many” is not chunked with “person” to a noun phrase but it is grouped with “how”.

 ISBN: 978-1-61208-165-6

Hence, instead of the output [how] [many persons] the query chunker generates the output [How many] [persons].

C. Interpretation

Interpretation of linguistically analyzed query sentences is a combination of noun phrase extraction and more refined parsing of specific patterns.

In a first step the query interpreter extracts all the noun phrases. This guarantees that at least query notions can be extracted, even if a more specific pattern cannot be detected. The found query notions are used to check if they match against existing concepts, views or examples (see subsection D).

More specific patterns are constraint sentences (e.g., "The age must be greater than 20"). These sentences can be used within a query text to constrain the concept. Such constraint sentences can also have adjectives at the end (e.g. "must be old"). If such adjectives are found, then these adjectives are collected as value descriptor candidates. Constraint patterns can also appear within a query (e.g., "Which customer has placed Order 123"). In this example not any order is meant but a specific order (Order 123).

Another task of the interpretation module is to filter out most often used meta-information (e.g., "list of …", "set of … ").

D. Matching

If all the notions are extracted the system tries to match the notions found in the query with elements in the model. The matching procedure also checks if a constraint is applied on an attribute in the model. If this is not the case (e.g., Order 123), a warning is given to the user. If all the notions in the queries are found in the model or in model related information, then the query is successful. To achieve this, the extracted notions are firstly compared with the concepts in the model (i.e., can the extracted notion, or its singular form be found in the model). If this does not work then the extracted notion is searched in the list of similar words or examples which can be stored during modeling as additional information. Since the similar notions as well as the examples are related to a model concept, these notions can be traced back. Therefore, in any of the above mentioned cases, the notion found in the query can be replaced by the concept in the model to accomplish the next step (path finding). If all the notions extracted from the query are found in the model, then the tool can determine a path between these model concepts. Path finding is done by checking if all the concepts, which are necessary for the query belong to the same connected component within the conceptual model graph.

VI. Discussion, Test and Prototype

A. Discussion

The four important factors of agile software development are [33]

1. Individuals and interactions over process and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

Comparing the approach with the above manifesto, the following can be said. Yes, there is a tool and tool support was one aim of this approach. However, it should be clear from the previous sections, that the tool does not dictate any process. In contrary, the aim of the tool is to enforce the interaction between individuals (stakeholders). If a query is not successful, then the stakeholders must discuss the failures and the communication process between them is improved!

Natural language queries are created for a very special purpose. From natural language queries, developers can manually and easily derive SQL queries, which can be embedded into an implementation of the future information system. Hence, these natural language queries represent parts of the future software. Thus, this approach fulfills "working software over comprehensive documentation".

Since the queries represent expectations of the customers, these customers will not understand themselves as a part in a contract negotiation but as an important part of a collaboration.

Finally, the need for responding to a change request is not restricted by using natural language queries as test cases.

B. Tests

Natural language queries, which were found in literature and own created queries were taken as test cases to test and improve the linguistic instruments. Among these test cases, the greatest set of natural language queries came from the Geo Query Project [28]. In this project 880 query sentences are used. Theses sentences can be categorized in queries starting with "What", "How", "Which", "Where" and other queries. These other queries do not start with an interrogative but start with a verb (e.g., "list", "give me", "name the", etc.) or they neither start with a verb nor with an interrogative (e.g., only a noun phrase is used for the query). The majority of query sentences is provided for queries starting with "What" followed by "How" and "Which". With the Geo Query Corpus a substantial test set was used. This high number of test cases is also important to get a good impression about the several different possibilities to express queries. All the queries in the test sets were applied on the query analyzer.

The user story interpreter currently accepts the two user story patterns as described in Section IV.

C. Tool Prototypes

The tools are implemented in Java. Currently, there are three sorts of tools. The first tool accepts a user story and extracts the actors and domain concepts (e.g., customer, order, order number, order item, etc). It stores this extracted concepts into an XML file, which can then be imported to the second tool, the concept editor. The concept editor graphically displays domain concepts (i.e., domain classes and attributes) but not actors. The third tool is the query analyzer tool. It is an add-on of the concept editor. A query analyzer window is opened if the user presses the Q-Button

 ISBN: 978-1-61208-165-6

in the concept editor. The query analyzer tool has a text area for the query and an error and warnings display area. Figure 1 and Figure 2 show the concept editor and the query analyzer interface. Figure 1 shows the model after the end user has stated the user story example: “As a customer, I can place an order”. Since “customer” firstly is seen as an actor and not as a domain class, it does not appear in the model presented by the concept editor. Only “order” is presented. Let’s continue with the already known natural language query examples and suppose, the end-user would like to execute the following two queries:

- Tell me the order items that belong to an order.
- Which customer has placed Order 123.

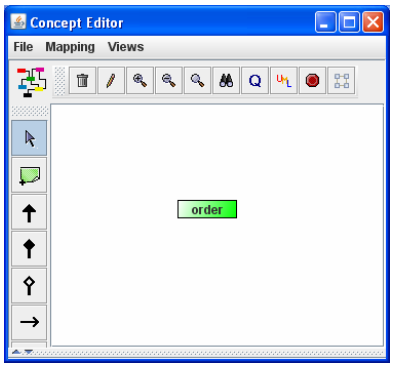

Figure 1: Initial model

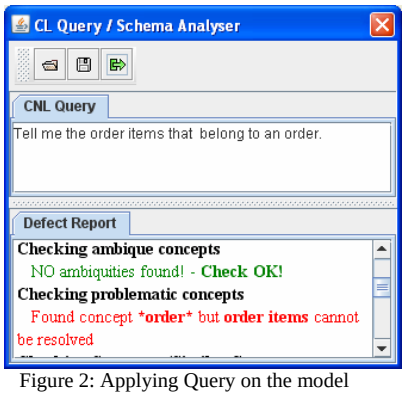

Figure 2: Applying Query on the model

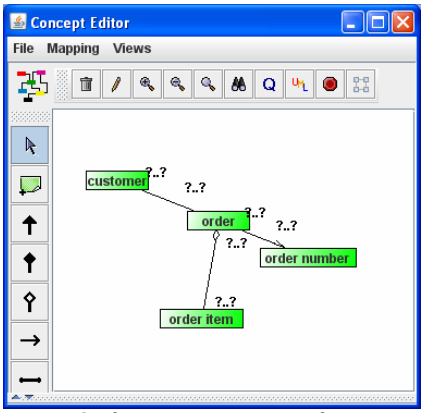

Figure 3: Result of an interaction process after 2 queries

After applying the first query on the query analyzer, the query analyzer returns failures (Figure 2). In addition, also the query “which customer has placed Order 123” is applied on the initial model. For the second query, the tool returns the information, that a constraint can only be defined on an attribute. The stakeholders must discuss what is missing (e.g., order number). If afterwards a refinement step is done, then the improved model might look as follows (see Figure 3). This iterative refinement by testing with natural language queries is similar to the paradigm of test driven development.

VII. Conclusion and Future Work

In this paper the tool supported extraction of concepts out of user stories were described. Since such user stories are not detailed requirements but should give an idea what an end user might expect from the future systems, these stories must be complemented by additional information. In this paper one strategy of gathering additional information was presented. Particularly, natural language queries can be used even in an early phase of information system development. It was also shown how this strategy itself can be supported by a tool using linguistic instruments.

This work will be continued with a technical refinement of the query analyzer. Beside the explained interactive mode (see Figure 2), a batch mode will be implemented. In the batch mode many queries will be executed on the actual model. All problems will be stored in a report file.

Refinement of the user story interpreter is another future task. Though, the above two mentioned patterns are the most famous ones and are very often mentioned in the context of user stories, variations of these patterns exist. For instance, the “As a <role> I can <feature>” pattern can be extended to “As a <role> I can | want <feature> (so that | because) <reason>. Attention will also be paid on these variations and automatic extraction of necessary information from these variations.

References

[1] P. Assenova and P. Johannesson, “Improving the Quality in Conceptual Modelling by the Use of Schema Transformations,” Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, Lecture Notes in Computer Science (LNCS), Vol. 1157. Springer Verlag Berlin Heidelberg New York, 1996, pp. 277 – 291.

[2] H. Berger, M. Dittenbach, and D. Merkl, “Quering Tourism Information Systems in Natural Language,” Information Systems Technology and its Applications – Proceedings of the 2nd Conference ISTA 2003, GI Lecture Notes in Informatics, Vol. p-30, Koellen Verlag,, Bonn, 2003, pp. 153 – 165.

[3] A.C. Bloesch and T.A. Halpin, “ConQuer: A Conceptual Query Language,” Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, Lecture Notes in Computer Science (LNCS), Vol. 1157. Springer Verlag Berlin Heidelberg New York, 1996, pp. 121 – 133.

[4] H. Dalianis, “A method for validating a conceptual model by natural language discourse generation,” Proceedings of the Fourth International Conference CAiSE’92 on Advanced

 ISBN: 978-1-61208-165-6

Information Systems Engineering, Lecture Notes in Computer Sciences (LNCS) Vol. 594, Springer Verlag, pp. 425 - 444.

[5] St. Easterbrook, E. Yu, J. Aranda, Y. Fan, J. Horkoff, M. Leica, and R.A. Quadir, "Do Viewpoints Lead to Better Conceptual Models? An Exploratory Case Study," Proceedings of the 13th IEEE Conference on Requirements Engineering (RE'05). IEEE Press, pp. 199 – 208.

[6] D. W. Embley, "NFQL: The Natural Forms Query Language," ACM Transactions on Database Systems, Vol. 14(2), 1989, pp. 168 – 211.

[7] R. Ge and R.J. Mooney, "A Statistical Semantic Parser that Integrates Syntax and Semantics," Proceedings of the Ninth Conference on Computational Natural Language Learning, Ann Arbor, MI, 2005, pp. 9-16.

[8] T. Halpin and M. Curland, "Automated Verbalization for ORM 2," Proceedings of OTM 2006 Workshop - On the Move to Meaningful Internet Systems 2006, Lecture Notes in Computer Science (LNCS 4278), Springer Verlag, pp. 1181 – 1190.

[9] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide, "Exploring Fact Verbalizations for Conceptual Query Formulation," Proceedings of the Second International Workshop on Applications of Natural Language to Information Systems, IOS Press, Amsterdam, Oxford, Tokyo, 1996, pp. 40 – 51.

[10] H. Jaakkola and B. Thalheim, "Visual SQL – High Quality ER Based Query Treatment," Proceedings of Conceptual Modeling for Novel Application Domains, Leture Notes in Computer Science (LNCS), Vol. 2814, Springer Verlag, Berlin, Heidelberg, New York, 2003, pp. 129 – 139.

[11] K. Järvelin, T. Niemi, and A. Salminen, "The visual query language CQL for transitive and relational computation," Data & Knowledge Engineering, Vol. 35, 2000, pp. 39 – 51.

[12] Z.T. Kardovácz, "On the Transformation of Sentences with Genetive Relations to SQL Queries" Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB 2005), Lecture Notes in Computer Science (LNCS), Vol. 3531, pp. 10 – 20.

[13] M. Kao, N. Cercone, and W.-S. Luk, "Providing quality responses with natural language interfaces: the null value problem," IEEE Transactions on Software Engineering, Volume 14 (7), 1988, pp. 959 – 984.

[14] E. Kapetainos, D. Baer, and P. Groenewoud, "Simplifying syntactic and semantic parsing of NL-based queries in adavanced application domains," Data & Knowledge Engineeing Journal , Vol. 55, 2005, pp. 38 – 58.

[15] R.J. Kate and R.J. Mooney, "Using String-Kernels for Learning Semantic Parsers," COLING/ACL Proceedings, Sydney, 2006, pp. 913-920.

[16] G. Kotonya and I. Sommerville, Requirements Engineering, Wiley Publ. Company, New York, 1998.

[17] O. Lindland, G. Sindre, and A. Solvberg, "Understanding Quality in Conceptual Modeling," IEEE Software, March 1994, pp. 29 – 42.

[18] D. Moody, "Graphical Entity Relationship Models: Towards a More User Understandable Representation of Data," Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, Lecture Notes in Computer Science (LNCS), Vol. 1157. Springer Verlag Berlin Heidelberg New York, 1996, pp. 227 – 245.

[19] D. Moody, "Theoretical and practical issues in evaluating quality of conceptual models: current state and future directions," Data & Knowledge Engineering Volume 55, 2005, pp. 243 - 276.

[20] V. Owei, H-S. Rhee, and Sh. Navathe, "Natural Language Query Filtration in the Conceptual Query Language," Proceedings of the 30th Hawaii International Conference on System Science, Vol. 3. IEEE Press , 1997, pp. 539 – 550.

[21] K. Pohl, Requirements Engineering, Grundlagen, Prinzipien, Techniken, dPunkt Publ. Company, Heidelberg, 2007.

[22] O. R. Ribeiro and J. M. Fernandes, "Validation of Scenario-based Business Requirements with Coloured Petri Nets," Proceedings of the 4th International Conference on Software Engineering Advances, IEEE Press (IEEE Digital Library), 2009, pp. 250 – 255.

[23] N. Stratica, L. Kosseim, and B.C. Desai, "Using semantic templates for a natural language interface to the CINDI virtual library," Data & Knowledge Engineering Volume 55, 2005, pp. 4 – 19.

[24] L.R. Tang and R.J. Mooney, "Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing," Proceedings of the 12th European Conference on Machine Learning (ECML-2001), 2001, pp. 466 - 477.

[25] J.F. Terwillinger, L.M. Delcambre, and J. Logan, "Querying through a user interface," Data & Knowledge Engineering, Volume. 63, 2007, pp. 774 – 794.

[26] Y.W. Wong and R.J. Mooney, "Learning for Semantic Parsing with Statistical Machine Translation," Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-2006), New York, 2006, pp. 439-446.

[27] M. Cohn, "User Stories Applied – for Agile Software Development", Addison Wesley Publ.Company, 2004.

[28] Geo Query Project http://www.cs.utexas.edu/users/ml/geo.html (last access: 31. May .2011)

[29] Penn-Treebank TagSet: http://www.cis.upenn.edu/~treebank (last access: 31. May 2011).

[30] C. Manning and H. Schütze, "Foundations of Statistical Natural Language Processsing," MIT Press, 2003.

[31] K. Beck, "Test Driven Development by Example," Addison Wesley Publishing Company, 5th Printing, 2004.

[32] E.F.T. K. Sang, and S. Buchholz, "Introduction to the CoNLL-2000 Shared Task: Chunking," Proceedings of CoNLL-200 and LLL-2000, 2000, pp.127-132.

[33] Manifesto for Agile Software Development: http://agilemanifesto.org/ (last access: 31. May 2011)

[34] User Stories: http://www.codesqueeze.com/the-easy-way-to-writing-good-user-stories/ (last access 31. May 2011)

[35] J. Rumbaugh, M. Blaha, W. Premelani, F. Eddy, and W. Lorensen, "Object-Oriented Modeling and Design," Prentica Hall Internatianal Inc. Publ. Company, 1991

[36] M. Smialek, J. Bojarski, W. Nowakowski, and T. Strazak, "Writing Coherent User Stories with Tool Support," H. Baumeister, M. Marchesi, M. Holcombe (eds), 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005), LNCS, Vol. 3556, 2005, pp. 1217 – 1221.

 ISBN: 978-1-61208-165-6