The Effectiveness of Conceptual Models Auto-Generated from a Set of User Stories Written as BDD Scenarios: A Proposed Study

Abhimanyu Gupta[1] and Geert Poels [1,2]

[1] *Department of Business Informatics and Operations Management, Faculty of Economics and Business Administration, Ghent University, Ghent, Belgium*
[2] *FlandersMake@UGent – core lab CVAMO, Ghent, Belgium*

Abstract

While agile methodologies are commonly used in software development, researchers have identified many issues related to requirements elicitation in agile projects. Some of these issues relate to the complexity of managing and understanding user stories, which is a widely used requirements specification mechanism in Agile methodologies. This research proposes the automatic generation of conceptual models from the user stories and the corresponding Behavior-Driven Development acceptance criteria to help managing and understanding user stories. This paper discusses a proposed study to evaluate the usefulness of the auto-generated conceptual models from the user stories.

Keywords

user stories, agile methodologies, conceptual models, NLP, model generation

1. Introduction

In Agile software development, requirements documentation is mainly limited to user stories [1]. A user story is a simple description of a feature of the working software as it is expected by a user [2, 3]. Because of the substantial number of user stories that are written in Agile software development projects, the project team may encounter difficulties in maintaining, tracing, and managing user stories [4]. Thus, for moderately complex software, the number of user stories easily exceeds human capacity of overview and understanding. Considering that user stories might be the only documentation available to the project team, acquiring an overall understanding of the system's required features and their dependencies might be challenging. One way to address this problem is to use conceptual models in agile software development. Conceptual models describe specific perspectives on reality for the purpose of understanding and communication [5]. However, creation and maintenance of conceptual models in each iteration in agile development might be not feasible. Thus, a proposed solution is to automatically generate conceptual models from sets of user stories. When the user stories change, the models are automatically updated by generating them again.

With the current popular format of user stories (Connextra template [2]), it is difficult to develop some types of conceptual models (e.g., process models, state machines) as the information to generate conceptual models that allows analyzing interactions or dependencies between related user stories is simply not present in the user stories. With the use of Behavior-Driven Development (BDD) scenarios, more information about interactions and dependencies between user stories can be used in the model generation process. So, we developed an NLP based algorithm to generate several types of conceptual model (i.e., domain model, use case model, process model and state machine) automatically from BDD scenarios – which we refer to as 'extended' user stories. In this paper, we propose for discussion initial ideas of a study for testing how the auto-generated conceptual models from a set of extended user stories can be useful in the comprehension of the requirements and in the reduction of requirements ambiguities.

Agil-ISE 2022: Intl. Workshop on Agile Methods for Information Systems Engineering, June 06, 2022, Leuven, Belgium
EMAIL: Abhimanyu.Gupta@UGent.be (A. Gupta); Geert.Poels@UGent.be (G. Poels)
ORCID: 0000-0001-9247-6150 (G. Poels 2)

In the next section, we briefly outline the approach and the framework that we used to develop the algorithm that generates multiple conceptual models from a set of related extended user stories. This is followed by a demonstration of the inputs and outputs of a tool we developed by implementing the algorithm. Then in the next section we propose how to evaluate the usefulness of the models generated by the tool in terms of ensuring better comprehension of the requirements. The last section presents some further reflections.

2. Approach and framework

As stated in the Introduction, we define *extended user stories* as user stories that are supplemented by the acceptance criteria in BDD scenarios. A BDD scenario consists of a feature title, an associated user story and the scenario proper which is defined by three keywords – "Given", "When", and "Then" [6], for three distinct parts of the scenario – precondition, trigger and postcondition – which are the three components of the acceptance criteria for user stories. The "Given" indicator marks the precondition part in the scenario, in which the context is described that is assumed by the user story. The action to be performed on the object as described in the means part of the user story, can only be triggered in this context, which is expressed in terms of one or more system objects and their states, where object states can be the result of actions described in other user stories (i.e., thus indicating dependencies between user stories). The trigger part, with indicator "When", describes one or more events that trigger the action described in the user story to be performed. Finally, the "Then" indicator marks the postcondition part of the scenario that describes the outcome(s) of the user story in terms of object states achieved. The BDD scenario template that is recommended by the Agile community [6] is presented in Table 1.

Table 1
The BDD Scenario Template [2]

BDD Scenario
Feature: [title] *User story: As a [role] I want to [means] so that [ends].* *Scenario: [title]* *Given [context] And [some more context],* *When [some event occurs] And [some other event occurs],* *Then [outcome] And [some other outcome].*

Using the above template, an example of an extended user story is: As a customer, I want to cancel a service request so that the team can focus on other active requests. Given a service request is submitted or open, when the customer decides to cancel the service request, then the service request will be canceled.

Our proposed framework (Figure 1) on developing multiple conceptual models is based on the generic framework proposed by [7] as similar stages related to NLP analysis and creation of intermediate data are followed. The inputs are a set of related extended user stories and thus the input has richer information than a set of standard user stories and more structure than textual requirements. Also, our framework generates multiple related conceptual models simultaneously unlike previous research where the focus is to generate one type of conceptual model.

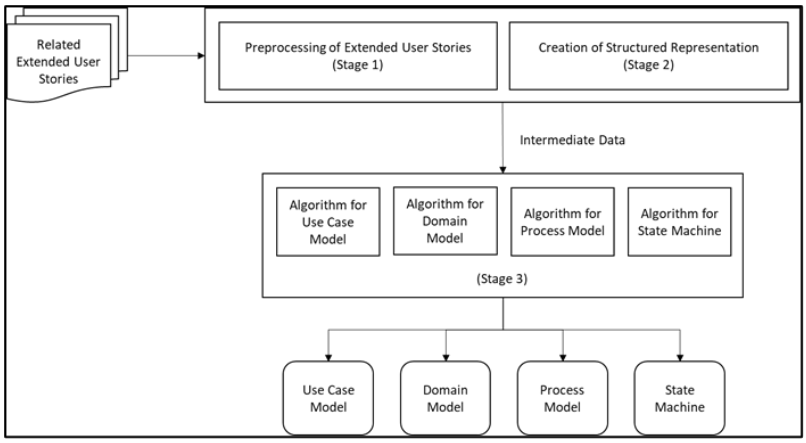

Figure 1: Framework of developing multiple conceptual models from extended user stories

Using the algorithm, we generate from the set of related extended user stories, four different types of models: use case models (for what purposes systems are used by users), process models (what actions are performed by/for which users and in what order), domain models (what objects the system needs to store data about, how these objects are related, and which actions changes their data), and state machines (how the state of objects, as represented by their data, changes through actions). The models we generate are stylized versions of these model types, which capture the most essential information as found in, for instance, their UML counterparts. Some model constructs are not used in the stylized versions because the information to generate the models is not present in the extended user stories.

3. NLP Based tool and Outputs

We demonstrate the creation of the multiple conceptual models using our NLP based tool. We consider an example set of user stories whose objective is to create an application for handling service requests of the IT users in an organization. There are two defined roles who can use the application – customer (i.e., IT users as customers of the IT support team) and support assistant. A customer can create and cancel a service request and approve or reject the work done by the support assistants in response to the service request. A support assistant can accept a service request, after which he or another support assistant can resolve the service request. The set of six extended user stories is shown below in Table 2. This set is then used as input file for the tool.

Table 2: **A sample set of extended user stories**

1. As a customer, I want to create a service request so that I can have my problem solved. Given that the customer is active, when he submits a service request then the service request should be submitted.
2. As a support assistant, I want to accept so that I can start working on it. Given it is submitted, when the team starts working on it then it is open.
3. As a support assistant I need to resolve so that the customer can close the ticket. Given a service request is open, when the team resolves it, then it is fixed.
4. As a customer I need to approve the service request so that it can be closed. Given a service request is fixed, when I approve it, then the service request becomes closed.
5. As a customer I need to reject the service request so that it can be reopened. Given a service request is fixed, when I reject it, then the service request is open.
6. As a customer I want to cancel a service request so that the team can focus on other active requests. Given a service request is submitted or closed when customer cancels it then it will be canceled.

After inputting, this sample set of extended user stories is processed by the tool to create multiple conceptual models. For preservation of space, we are showing the use case (model), state machine for the service request, and the process model in Figure 2. The domain model (showing relationships between concepts) is not included in Figure 2. Also, note that as we currently use our tool as a research tool, some features of the models are not shown as in their UML counterparts, although the information is present. For example, non-sequential sequence flow in processes is shown using symbols inside the

activities – the process model in Figure 2 indicates that "cancel service request" can follow after "create service request" or "approve service request."

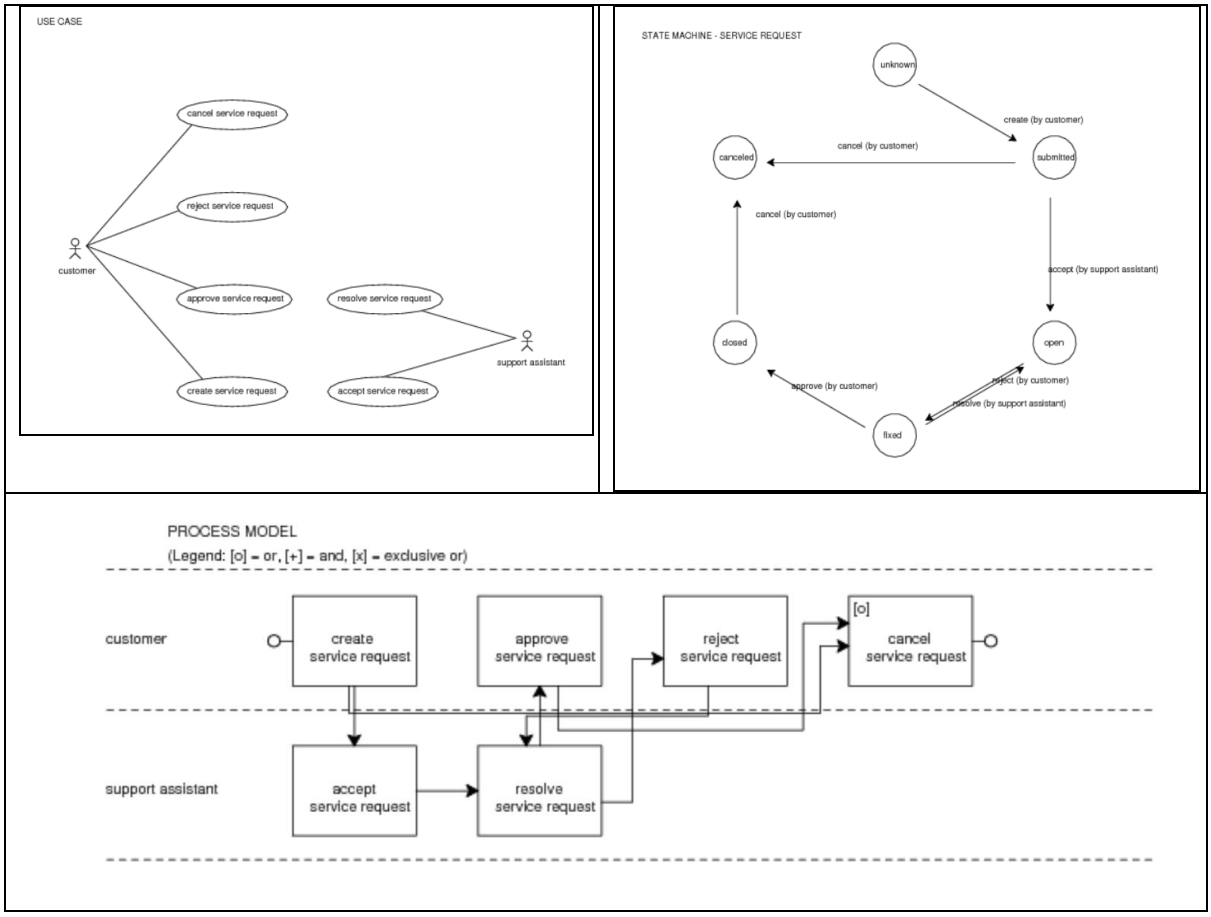

Figure 2: Multiple conceptual models generated from Table 2

4. Evaluating the usefulness of the generated models

In the current stage of the research, we intend to conduct a qualitative study to evaluate the usefulness of the auto-generated conceptual models from a set of extended user stories. Our primary objective is to identify the value that automatically generated conceptual models can bring to Agile practitioners. We will conduct in-depth semi-structured interviews with ten agile practitioners. We prefer to conduct a qualitative study rather than a quantitative study (e.g. an experiment), as qualitative studies help to achieve depth of understanding of a phenomenon rather than breadth of understanding [8]. The Agile practitioners will be selected based on their experience (i.e., at least five years of experience in practicing Agile software development). Because of the small sample size and specific experience required to participate in the study, we will use a purposeful sampling strategy [8].

Research suggests that some organizations use high level conceptual models such as domain models and mind maps [9, 10] in Agile development but the use of multiple conceptual models (as described in this paper) is not common. Therefore, we plan to gradually introduce and engage our interviewees with the auto-generated conceptual models created by our tool. We will conduct the interviews in three stages.

In the first stage, the goal is to identify the perceived benefits of using conceptual models in Agile software development. In this stage, we will not disclose the tool to create auto-generated conceptual models from user stories. The interview questions that guide this stage of the interview are: (1) what

are the potential benefits of using conceptual models when using an Agile methodology? (2) what are the potential pitfalls in using conceptual models using an Agile methodology? (3) what specific conceptual models will you use and for what task? At this stage, we do not anticipate in-depth responses to these questions. This could be because our interviewees may not be familiar with multiple conceptual models. However, we wish to find out their prior expectations regarding the use of conceptual models even if they do not have much experience in using models.

In the second stage, we will introduce the tool to create auto-generated conceptual models from user stories. We will first show the practitioners the BDD template (Table 1) and then demonstrate with the tool how a set of extended user stories (Table 2) generates conceptual models (Figure 2). We will then continue the interview by revisiting the same questions as in stage 1. We now anticipate different responses as the interviewees might now have more concrete ideas about the use of conceptual models and that they can be generated automatically from a requirements artifact (i.e., user stories) that they are used working with.

In the third stage, we will actively engage the interviewees with the tool such that they can experience using auto-generated models. The motivation for this step is to demonstrate to the participant how minor changes in a set of user stories can drastically modify the corresponding conceptual models, making the changes visible. We will delete a user story from the set of six without the participant knowing which user story is deleted. We will then ask the participant to find out which one of them was deleted. Next, we will use the tool to automatically generate the conceptual models from the modified set and show to the participant. Then we will ask the participant whether the models help identify the deleted user story.

Then we will make minor modification to a user story (from the original set), which can bring large effect of the conceptual models, without the participant knowing what is modified. Next, we will use the tool to automatically generate the conceptual models from the modified set and show to the participant. Then we will ask the participant whether the models help identify the modified user story. For example, we will change the word "closed" to "fixed" in user story 6. The revised user story is "As a customer, I want to cancel a service request so that the team can focus on other active requests. Given a service request is submitted or ***fixed***, when customer cancels it then it will be canceled." After the change in the user story, we will ask the practitioners to identify what was changed in the user story. We will then show the modified conceptual models. (e.g., Figure 3). In this Figure, we note that the sequence flow from "approve service request" to "cancel service request" is removed and the sequence flow from "resolve service request" to "cancel service request" is introduced from the previously generated process model (Figure 2). This change also identifies "approve service request" as an end event.

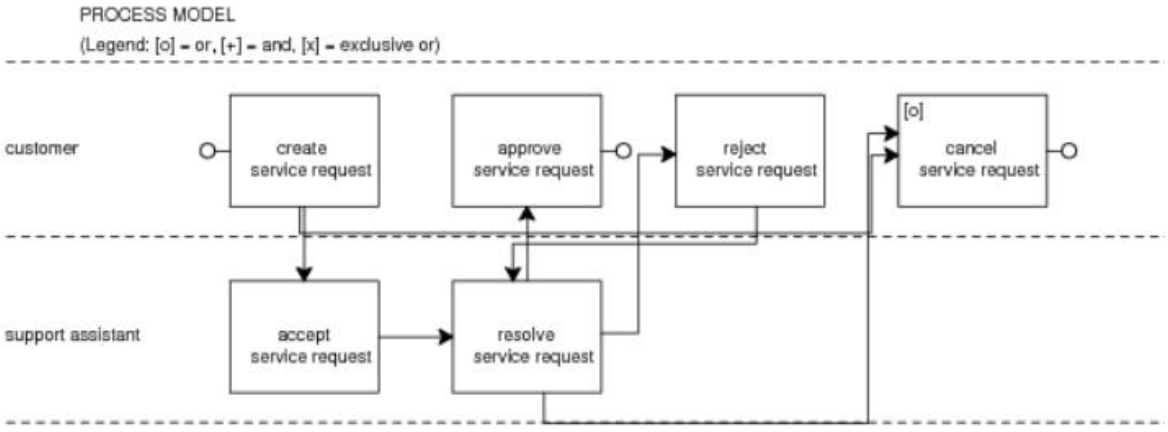

Figure 3: Revised Process Model after changing one word in the sixth user story

After engaging with the practitioners with the tool, we will repeat the questions asked in the previous two stages. We will also ask specific questions such as “how automatic generation of conceptual models can benefit management of user stories?” “What specific ways will you use the conceptual models in managing the process of software development?”, and “what would you do differently in managing user stories, if you had access to this tool?”

By demonstrating the functionality of the tool, we intend to engage the practitioners with the conceptual models. As the tool automatically recreates conceptual models with whatever input is provided, we indirectly demonstrate that users need not maintain the versioning of the conceptual models as these models are automatically updated after each iteration in the user stories. Conceptual models are used primarily for overall domain understanding and communication [5] but in this research we would like to identify what agile practitioners can do specifically with the conceptual models. For example, when the process model generated from the user stories is clearly incomplete (e.g., no path from the start event to the end event), we wish to find out whether practitioners can infer what this means for the user stories (e.g., based on the models, can they detect that the set of user stories is incomplete)?

We will record the interviews and create transcripts. At the end of all the interviews, we will identify ideas or concepts from the transcripts by tagging codes that summarizes those ideas or concepts. Then we will perform a qualitative analysis using those ideas and concepts to understand the practitioners’ opinions about how useful the auto generated conceptual models are. We will repeat the analysis in each stage of interview of practitioners. We anticipate that practitioners will demonstrate high engagement level in the last stage of interview and will come up with responses on creative use of conceptual models in agile methods.

5. Reflection

In practice, even if the use of conceptual models would have value in Agile software development, developing multiple conceptual models from user stories would require knowledge and skills that may not be present. Thus, a key advantage of our approach is that the developers need not know how to create conceptual models. If the user stories that are used as input to the tool are consistent and accurate, then the conceptual models that are automatically generated are also consistent and accurate. Similarly, users do not have to maintain the conceptual models when the user stories change. Version controlling of the set of the user stories will generate different versions of the conceptual models and therefore maintenance of the models will be easy. To identify the usefulness of the generated models in practice, we propose to do a study with practitioners where we will engage the practitioners to interact with the user stories and the conceptual models generated from them. The intention of this study is to understand the benefits of automatically generation of conceptual models from the user stories.

6. Acknowledgements

This work was supported by the Fund for Scientific Research – Flanders (FWO) (Research Project Grant G.0101.16N-39515). We also acknowledge the guidance of Dr. Palash Bera in this research.

7. References

[1] I. Inayat and S. S. Salim, "A framework to study requirements-driven collaboration among agile teams: Findings from two case studies," *Computers in Human Behavior*, Article vol. 51, no. Part B, pp. 1367-1379, 10/1/October 2015 2015.

[2] M. Cohn, *User Stories Applied: For Agile Software Development*. Boston: Addison-Wesley, 2004.

[3] D. Leffingwell, *Agile Software Requirements: lean requirements practices for teams, programs, and the enterprise* (Agile Software Development Series). Boston: Addision-Wesley, 2011.

[4] B. Ramesh, C. Lan, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Information Systems Journal*, Article vol. 20, no. 5, pp. 449-480, 2010.
[5] J. Mylopoulos, "Conceptual modeling and telos," in *Conceptual modeling, Databases and Cases*, P. a. Z. Locuopoulos, R., Ed. New York: John Wiley and Sons Inc, 1992.
[6] J. F. Smart, *BDD in action: Behavior-Driven development for the whole software lifecycle*. New York: Manning Publications Company, 2014.
[7] F. Bozyig, O. Aktas, and D. Kılınc, "Linking software requirements and conceptual models: A systematic literature review," *Engineering Science and Technology, an International Journal*, vol. 24, no. 1, pp. 71-82, 2021.
[8] M. Patton, *Qualitative research and evaluation methods*. Thousand Oaks, CA: 3rd Sage Publications, 2002.
[9] E.-M. Schön, J. Thomaschewski, and M. J. Escalona, "Agile Requirements Engineering: A systematic literature review," *Computer Standards & Interfaces*, Article vol. 49, pp. 79-91, 1/1/January 2017 2017.
[10] W. Helmy, A. Kamel, and O. Hegazy, "Requirements Engineering Methodology in Agile Environment," *International Journal of Computer Science*, vol. 9, no. 5, pp. 293-300, 2012.