Information and Software Technology 71 (2016) 58–76

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Using business process models to better understand the dependencies among user stories

Marina Trkman[a,*], Jan Mendling[b], Marjan Krisper[a]

[a] University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113 SI-1001 Ljubljana, Slovenia
[b] WU (Vienna University of Economics and Business), Department of Information Systems and Operations, Institute for Information Business, Welthandelsplatz 1, 1020 Vienna, Austria

ARTICLE INFO

Article history:
Received 1 June 2015
Revised 16 October 2015
Accepted 19 October 2015
Available online 27 November 2015

Keywords:
User story
Execution order dependency
Integration dependency
Context
Experiment

ABSTRACT

Context: Agile software development projects often manage user requirements with models that are called user stories. Every good user story has to be independent, negotiable, valuable, estimable, small, and testable. A proper understanding of a user story also requires an understanding of its dependencies. The lack of explicit representation of such dependencies presumably leads to missing information regarding the context of a user story.

Objective: We propose a method that facilitates better understanding of execution order and integration dependencies of user stories by making use of business process models. The method associates user stories with the corresponding business process model activity element.

Method: We adopted a situational method engineering approach to define our proposed method. In order to provide understanding of proposed method's constructs we used ontological concepts. Our method associates a user story to an activity element. In this way, the business process model can be used to infer information about the execution order and integration dependencies of the user story. We defined three levels of association granularity: a user story can be more abstract, approximately equal to, or more detailed than its associated business process model activity element. In our experiment we evaluate each of these three levels.

Results: Our experiment uses a between-subject design. We applied comprehension, problem-solving and recall tasks to evaluate the hypotheses. The statistical results provide support for all of the hypotheses. Accordingly, there appears to be significantly greater understanding of the execution order and integration dependencies of user stories when associated business process models are available.

Conclusions: We addressed a problem which arises from managing user stories in software development projects and focuses on the missing context of a user story. Our method contributes to the discipline of conceptual modeling in agile development. Our experiment provides empirical insight into requirement dependencies.

1. Introduction

User stories are the most commonly used requirement model in agile development projects [1–3]. Every good user story has to be independent, negotiable, valuable, estimable, small, and testable (INVEST) [4–7]. How to specify a good user story is well covered by the professional literature [5,6,8]. However, a single user story does not convey the entire business, but only a part of it [10]. This means that, while a user story is independent from the perspective of a developer, it can still depend on other user stories from a business perspective [10]. In this context, insights into requirement dependencies are crucial for project success [9–15] and more specifically for requirements interaction management [16]. Being unaware of requirement dependencies can lead to missing information regarding the context of a project, namely, its domain [5,17]. Martakis and Daneva [10] have emphasized the importance of integration and execution order dependencies. *Integration dependencies* present information about how certain user stories require other user stories to be previously developed. While *execution order dependencies* present information about how the completion of a user story directly impacts another. Strode [34] refers to these two types of requirements as technical dependencies and activity dependencies.

In practice, it is very important to understand if there are any dependencies among user stories and why they exist. Tracing execution

* Corresponding author.
E-mail addresses: marina.trkman@gmail.com (M. Trkman), jan.mendling@wu.ac.at (J. Mendling), marjan.krisper@fri.uni-lj.si (M. Krisper).

http://dx.doi.org/10.1016/j.infsof.2015.10.006
0950-5849/

order and integration dependencies is addressed in the Scrum agile method with the definition of the role of a Product Owner who is in charge of overviewing them. Recent literature suggests several solutions for building up an explicit knowledge base for a project's integration dependencies. Lin et al. [9] proposed a method for decomposing complex processes into phased goals, and grouping low-level user stories with high-level goals. Clarke and Kautz [18] developed a method for factoring epics into user stories. Leffingwell [5] proposed a model which decomposes project's requirements in a tree view. Liskin [19] concludes that managing the granularity levels of user stories needs to be investigated more. Recent literature also discusses how to create an explicit knowledge base capturing the execution order dependencies among user stories. Milicic et al. [20] proposes a user-centric method which organizes user stories along scenarios and users. Leffingwell [5] suggests that the sequence of execution should be represented by use case diagrams and use case specification documents. Patton [21] proposes a story mapping technique for breaking big stories down while still maintaining the big picture of a project. The overall purpose of the mentioned solutions is to create a conceptual model that complements the list of user stories with information about the integration and execution order dependencies.

Conceptual models are key artifacts for understanding an application domain and its requirements [22]. Since most agile development projects aim to support business needs [23], it is essential to understand the processes of the business domain that the user stories are intended to support [24,25]. A process is a sequence (or a flow) of activities in an organization with the objective of carrying out work [26]. Processes can be represented as business process models. System development often makes use of process models [27,28]. These models bear the potential to explicate the integration and execution order dependencies among previously defined user stories; however, there is no prior research that investigates this potential.

This paper proposes and experimentally validates a novel method called BuPUS (Business Process User Story method), which associates user stories with business process model activities in such a way that execution order and integration dependencies can be easily traced. Similar to [28–31], we build on a theoretical framework for empirically evaluating the conceptual modeling techniques. We measure the understanding of a user story with problem-solving questions [29,32]. The results show that the understanding of the execution order and integration dependencies of a user story is greater when having the output of BuPUS available in comparison to only having a list of user stories.

We proceed as follows. We first discuss the importance of the knowledge about the execution order and integration dependencies of a user story for domain understanding. We review existing solutions for recognizing these dependencies. Next, we propose our BuPUS method that integrates user stories with BPMN models in order to enhance an individual's domain understanding. We then formalize our hypotheses with Mayer's theory of multi-media learning. For evaluation, we define an experiment design, show statistical results, and discuss threats to validity. Upon this basis, we highlight implications for research and practice. Finally, we conclude the paper and present directions for future work.

2. Background

2.1. Requirement dependencies

A requirement dependency refers to a situation in which the progress of the action of one user story assumes the timely outcome of the action of another user story or the fulfillment of a specific condition [10]. Dependencies are ubiquitous when developing an application and occur between people, groups, tasks, and artifacts, including the software components under construction [33]. They can cause problems such as bottlenecks, blockages in the flow of work, and waiting, which potentially increase the likelihood of project delays, schedule overruns, and the need for task switching [34].

It is essential to understand requirement dependencies before carrying out the design of an application in order to avoid unnecessary implementation effort [35]. Focusing on dependencies as early as possible in a project is thus beneficial due to its potential to save rework and redesign [35]. On the other hand, ignoring the dependencies increases the risk attached to cost-effective project execution and, consequently, to project success [15,36–38]. Indeed, undetected dependencies can delay a project as people wait for resources, for the activities of others to be completed, or for necessary information [34].

Further, understanding requirement dependencies is of paramount importance for the successful deployment of agile development projects [35]. About 80% of all agile projects are supported by the agile development methods Scrum or XP [5]. Both of these popular methods suggest managing project requirements with user stories. This emphasizes the need to study dependencies among user stories (see [35,39,40]).

2.2. Dependencies among user stories

A user story is a brief statement that describes something the system needs to do for the user [5] and as such is business-oriented [41]. Its goal is to contain a few short notices about a desired function of the system [42] and provide a high-level overview of the requirements for a system [3]. Typically, a user story is created by following a general template [3,5,6]: *I as a <user role> can <function >*. As such, it is a non-visual requirement model. A user story model is typically used in agile software development projects [3]. A specific user story is just one of many user stories that represent a cohesive software product, and as such it cannot be treated as a stand-alone independent requirement. The dependencies among user stories need to be understood so that we can fully understand the domain of the software project.

The dependencies among user stories are a subject of speculation when previous knowledge about the domain is insufficient. We argue that the list of user stories alone hardly depicts the execution order and integration dependencies because of its three characteristics which we illustrate with user story examples in Table 1:

1) The naming of the user story functions can refer to different levels of abstraction (e.g. in Table 1 the function of user story US_299 "notify the customer about the credit committee's

Table 1
A list of user stories for Case 1.

List of user stories:	
US_189	I as a clerk in DEPT1 can modify the account's limit.
US_244	I as a clerk in DEPT1 can send a notification about a change in the account's attribute.
US_11	I as a clerk in the front office can create a request to modify the account.
US_299	I as a clerk in the front office can notify the customer about the credit committee's decision.
US_43	I as a clerk in the front office can give information about a rejection.
US_165	I as a clerk in the front office can read about the account's new limit.
US_99	I as a clerk in the front office can read about the decision.
US_999	I as a clerk in the front office can send a letter about the approval.
US_765	I as a credit committee (member) at a business unit can evaluate a non-standard account limit.
US_199	I as a credit committee (member) at a central unit can make an assessment report for a non-standard limit.
US_300	I as a clerk in the front office can notify the customer.
US_100	I as a credit committee (member) can evaluate the proposal.