Bridging User Story Sets with the Use Case Model

Yves Wautelet[1(✉)], Samedi Heng[2], Diana Hintea[3], Manuel Kolp[2], and Stephan Poelmans[1]

[1] KU Leuven, Leuven, Belgium
{yves.wautelet,stephan.poelmans}@kuleuven.be
[2] Université catholique de Louvain, Louvain-la-neuve, Belgium
{samedi.heng,manuel.kolp}@uclouvain.be
[3] Coventry University, Coventry, UK
diana.hintea@coventry.ac.uk

Abstract. User Stories (US) are mostly used as basis for representing requirements in agile development. Written in a direct manner, US fail in producing a visual representation of the main system-to-be functions. A Use-Case Diagram (UCD), on the other hand, intends to provide such a view. Approaches that map US sets to a UCD have been proposed; they however consider every US as a Use Case (UC). Nevertheless, a valid UC should not be an atomic task or a sub-process but enclose an entire scenario of the system use instead. A unified model of US templates to tag US sets was previously build. Within functional elements, it notably distinguishes granularity levels. In this paper, we propose to transform specific elements of a US set into a UCD using the granularity information obtained through tagging. In practice, such a transformation involves continuous round-tripping between the US and UC views; a CASE-tool supports this.

Keywords: User Story · UML · Agile development · XP · SCRUM

1 Introduction

Following [3], *User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.* [19] acknowledged that no unification is provided in *User Story* (*US*) templates. Indeed, the general pattern relates a WHO, a WHAT and possibly a WHY dimension (examples are provided in Sect. 4), but in practice different keywords are used to describe these dimensions (e.g. Mike Cohn's *As a* $<$ *type of user* $>$, *I want* $<$ *some goal* $>$ *so that* $<$ *some reason* $>$ [3]). Moreover, in the literature, no semantics are ever associated to these keywords. This is why, [19] conducted research to find the majority of templates used in practice, sort them and associate semantics to each keyword. These semantics were derived from several sources and frameworks (by order of importance [5,11,17,23]); some of these are derived from Goal-Oriented Requirements Engineering (GORE, see [8]).

S. Link and J.C. Trujillo (Eds.): ER 2016 Workshops, LNCS 9975, pp. 127–138, 2016.
DOI: 10.1007/978-3-319-47717-6_11

After performing a redundancy analysis, a first selection of keywords with associated semantics was performed. Then, applying them on large test sets led to a sub-selection of keywords forming a unified model. In the end, most of the semantics adopted for the remaining keywords were selected from the i* framework (i-star [4,22,24]); this is due to the research design that favored adopting elements of i* to higher the internal consistency of the unified US templates model (see [19]). Note that it is not the i* framework that is fully rebuilt for tagging US since concepts like the resource, the agent, etc. are not included in the unified model. The capability, a concept non-existent in i*, has been included in the unified model (see [19] and Sect. 3).

One may question the utility of such a model; why should US be "tagged" to a certain template and not simply be expressed using the WHO/WHAT and WHY pattern. The main advantage is that, if the tagging respects the semantics associated to the concepts, it provides information about both the nature and the granularity of the US element. Such information could possibly be used at a further stage for software analysis: structuring the problem and its solution, identifying missing requirements, etc. [9]. Visual GORE models were envisaged for graphical representation in [20]; the paper showed that GORE models are very well adapted for the purpose of US sets representation. Nevertheless, since GORE models are not an industry adopted practice, we explore in this paper an independent representation with the far more used *Unified Modeling Language* (*UML* [12]), *Use Case Model*. An instance of the latter produces a *Use Case Diagram* (*UCD*). A *Use Case* (*UC*) is a list of actions or event steps, typically defining the interactions between a role and a system in order to achieve a goal.

Facing the definition of US and UC, we can notice that there is a possible mismatch between the two concepts - the US can indeed include (very) fine-grained elements and the UC should be a coarse-grained element describing the software problem encompassing a set of fine-grained actions. When sorting is applied within US elements, a transformation process can generate a UCD. This is why, in this paper, **we envisage the representation of problem domain coarse-grained US elements as UC**. To this end, we map the elements defined in the UCM – i.e. the *Actor*, the *UC* and the relationships between use cases – with the elements defined in the unified US template model of [19].

In practice it is sometimes discouraged to use US and UCD concurrently because if not kept mutually up to date they can be inconsistent (see e.g. [6]). We therefore propose keeping US sets and the UCD consistent by auto-updating each change performed to one in the other through the use of a CASE-tool. They are considered here as two complementary views. We identify three primary benefits of our transformation approach: (1) a graphical high-level (coarse-grained) representation of requirements through the UCD; (2) multi-dimensional structuring of requirements and US dependencies at UCD level (not possible with US Maps); (3) identification of missing requirements (not expressed in the current US set) and the elimination of redundant US.

2 Related Work and Positioning

The CASE-Tool *Visual Paradigm* [13] (VP) already proposes to transform US into a UCD. In their approach, the actor expressed in the WHO dimension becomes an actor of the UCD and the element in the WHAT dimension becomes a UC without any selection based on granularity. The rule is simple - a US with a WHAT element becomes a UC in the UC diagram. Nevertheless, UML points to the use of UC as elements representing an entire process rather than fine-grained (or atomic) elements as US can be. This does not mean that a US cannot include elements adequately describing UC, but that a sorting process is required in order to only include relevant elements as UC. The approach of VP can thus be said to be "naive" because including elements not relevant as UC.

Structuring US is often performed using the *User Story Mapping* (*USM*) technique (see [14]); the latter uses *Story Maps* which are difficult to maintain and read. Building a UCD from a set of US could be compared to USM. In our approach, we split a US in 2 or 3 dimensions and we use the graphical representation of the UCD for relevant elements. Our aim is to obtain:

- an easy to read graphical representation of requirements. Story Maps remain limited to post-its on a board or even on the ground;
- an advanced structuring of requirements where we can overview the dependencies of coarse-grained elements to one another with the use of $<<include>>$ and $<<extend>>$ relationships. Fine-grained elements are not part of the UCD but can be represented under the scope of particular UC for example in workflows (UML activity diagrams or BPMN Diagrams, etc.). The latter is outside the present scope;
- an identification of gaps in requirements and elimination of redundant US elements from the graphical representation. By limiting the graphical UCD representation to solely coarse-grained elements we can obtain a high-level representation of the system-to-be. This allows to overview if, at an operational level, all aspects required to solve the software problem have been taken into account; if not they can be added to the US set. Similarly, elements appearing to be redundant because appearing in several US of the US set can lead to remove US from the set.

AgileModeling [2] emphasizes the usage of models for better understanding of the system-to-be and argues that it is useful to produce at least some models including a UML UCD and class diagram for the very first iteration called *iteration zero*. No transformation process is nevertheless provided.

3 Unified-Model of User Stories' Descriptive Concepts

As evoked, [19] propose to build a unified model for designing US templates. The overall approach is further structured in [20]; this structure is also valid for the present research. Figure 1 represents the meta-model of US templates in the form of a class diagram. The instance of one of these classes is a US element in

itself from a concrete US. A US template can be designed taking an element from the WHO, WHAT and possibly WHY dimensions. The link between the classes conceptually represents the link from one dimension to the other. Specifically, the unidirectional association from the *Role* to one of the *Capability*, *Task* or *Goal* classes implies that the target class instantiates an element of the WHAT dimension (always tagged as *wants/wants to/needs/can/would like* in the model). Then, the unidirectional association from one of these classes instantiating the WHAT dimension to one of the classes instantiating the WHY dimension (always tagged as *so that* into the model) implies that the target class can instantiate an element of the WHY dimension. The following is a US template supported by our model: *As a* $<Role>$, *I would like* $<Task>$ *so that* $<$*Hard-goal*$>$.

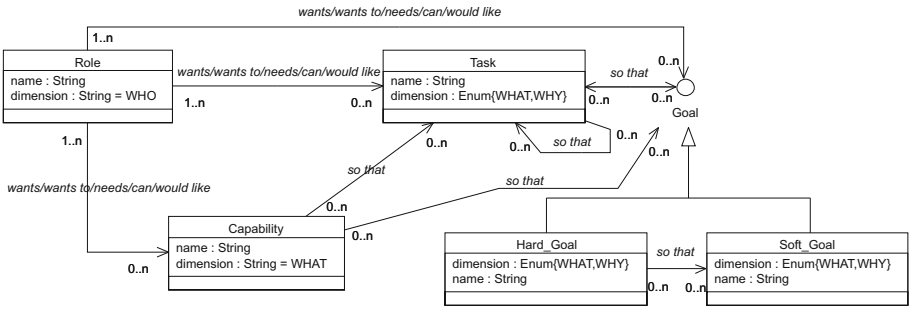


Fig. 1. Unified model for user story descriptive concepts

Each concept is associated with a particular syntax (identical to the name of the class in Fig. 1) and a semantic. Due to a lack of space we do not depict the semantic associated to each of the concepts here; it can be found in [19,20].

For granularity evaluation for functional elements, a few more explanations are required to distinguish the *Hard-goal*, *Task* and *Capability* elements.

The *Hard-goal* is the most abstract element; there is no defined way to attain it and several ways could be followed in practice. It is indeed part of the problem domain. The *Task* represents an operational way to attain a *Hard-goal*. It is thus part of the solution domain. An example of a *Hard-goal* could be to *Be transported from Brussels to Paris*; it can be the *Hard-goal* of a traveler but there are several ways to attain this *Hard-goal* (by train, by car, etc.).

The *Task* and the *Capability* represent more concrete and operational elements but these two need to be distinguished. The *Capability* does in fact represent a *Task* but the *Capability* has more properties than the former since it is expressed as a direct intention from a role. In order to avoid ambiguities in interpretation, we point to the use of the *Capability* element only for *an atomic Task* (i.e., a task that is not refined into other elements but is located at the lowest level of hierarchy). A *Task* could then be *Move from Brussels to Paris by car* and a *Capability* would be *Sit in the car*.

4 Running Example

Our proposal will be illustrated using a running example about carpooling. Carpooling deals with sharing car journeys so that more than one person travels within a car. In this context, it becomes increasingly important to save gas, reduce traffic, save driving time and control pollution. *ClubCar* [15] is a multichannel application available as an Android application, SMS service and IVR system. Users of ClubCar are riders and/or drivers that can register by SMS, voice or through an Android app. Roughly speaking, the software allows drivers to propose rides and submit their details with *dates*, *times*, *sources* and *destinations* while riders can search for available rides.

As shown in Table 1, we have taken a sample of the US of the ClubCar application to illustrate the research developed in this paper. Some of these US contain elements to be transformed in UC and elements that are not relevant as UC (see Sect. 5). The first column depicts the *Dimension* of US *Descriptive_Concept* (*D_C*), the second column describes the element itself and the last column provides the type of the *D_C*[1].

Table 1. US sample of the ClubCar application development

Dimension	Element	D_C Type
WHO	*As a DRIVER*	Role
WHAT	*I want to register to the service*	Task
WHY	*So that I can propose a ride to go from A to B*	Hard-goal
WHO	*As a DRIVER*	Role
WHAT	*I want to propose a ride from A to B with the price location and time of departure, and number of seats available*	Task
WHO	*As a DRIVER*	Role
WHAT	*I want to log in to the platform*	Capability
WHY	*So that I can register to the service*	Task
WHO	*As a RIDER*	Role
WHAT	*I want to be transported from A to B*	Hard-goal
WHO	*As a DRIVER*	Role
WHAT	*I want to confirm the proposal*	Capability
WHO	*As a DRIVER*	Role
WHAT	*I want the RIDER to be satisfied of my service*	Soft-goal

[1] Note that, when there were several possibilities, a choice ensuring the consistency of the entire set has been made.

5 User Stories Integration Through a Use-Case Diagram

5.1 The Role

A Role within a US can be forwarded to an Actor in the UCD. The UCD Actor is indeed the only structure defined to represent the WHO dimension (see Fig. 2).

5.2 Hard-Goal, Task and Capability

Three functional elements – the *Hard-goal*, the *Task* and the *Capability* – can be found in the unified model. The *Capability* is located on a fine level of granularity and thus non relevant for inclusion in the UCD. The Hard-goal and the Task elements are aimed to contain coarse-grained elements thus relevant for inclusion as UC.

Hard-goal and Task elements can be distinguished by their nature (i.e. their formulation) rather than by their grain level. Both are indeed intended to represent coarse-grained elements. The Hard-goal is the most abstract element and the Task is its counterpart expressed in an operational manner. This means that the Task represents a way of fulfilling the Hard-goal (entirely or parts of it); the Task is thus the counterpart of the Hard-goal in the solution domain while the Hard-goal belongs to the problem domain.

In line with UML, the use case does not necessarily explicit HOW the problem should be solved (this can be documented within on a workflow containing fine-grained elements) but rather WHAT should be achieved. **US elements tagged as Hard-goals should thus necessarily be represented as use cases** since they depict WHAT problem should be solved. The tricky question is then to determine if Task elements must also be represented as UC. Since Tasks are part of the solution domain, if they are represented as UC they should be linked to the Hard-goals they furnish part of a solution to; this means that we can make their relationship explicit. This is something useful when different US elements represented as Hard-goals make use of the same US elements represented as Tasks. It indeed allows to show that some behavior can be reused in several situations. This is what $<<include>>$ and $<<extend>>$ dependency relationships are intended to model in a UCD. We can then link the UC corresponding to a Hard-goal element with the one corresponding to a Task element using an $<<include>>$ and $<<extend>>$ dependency in function of the particular situation. If no behavior is recycled, we do not point to the representation of the US element tagged as Task in the UCD because this would lead to several UC that do not need to be externalized (leading to lots of $<<include>>$ and $<<extend>>$ dependencies) and are only sub-processes of the UC. Moreover, we want to make clear that we do not point to use these dependencies to depict means-end relationships between Hard-goals and Tasks. We point to keep the UCD as simple as possible and leave the description of solutions to the Hard-goal in other views (e.g. workflows). In a phrase, only decompositions of Hard-goals in Tasks where the Tasks are used in multiple Hard-goals are represented as UC in the UCD.

There is thus no universal answer to the representation of US elements tagged as Tasks in the UCD; as evoked it may be interesting to highlight the reuse of more special behavior but some Tasks are just subprocesses of the Hard-goal elements and should then not be represented as UC. These need to be documented for example in workflows depicting the realization scenario(s) of the Hard-goal (thus UC).

Let's finally note that UC transformed from Hard-goal elements can also be linked with other UC transformed from Hard-goal elements through an $<<$ *include* $>>$ relationship. In our context this shows that some Hard-goals are possibly needed for the realization of other Hard-goals. We do not consider $<<$*extend*$>>$ relationships among Hard-goals because such elements do have a possible stand-alone realization.

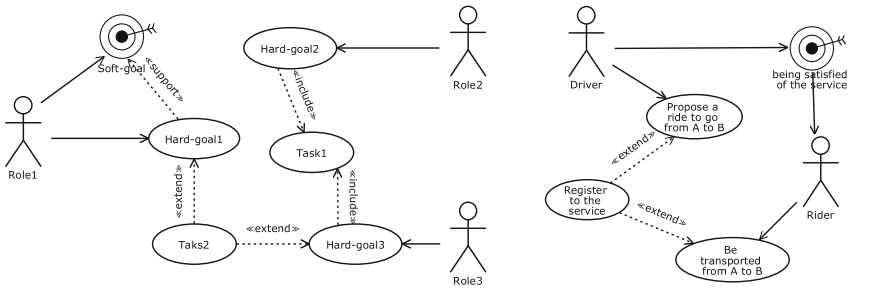
Concretely, in Fig. 2, the Task is linked with the UC representing the Hard-goal with an $<<$ *include* $>>$ dependency relationship from the Hard-goal to the Task. This is illustrated in the left side of Fig. 2 in a canonical form and instantiated on the Carpooling example in the right side of the same figure.

Representing both elements in the UCD is thus in some cases a way of explicitly linking the problem and solution domains where system behavior can be recycled in multiple use cases (thus Hard-goals).

5.3 The Soft-Goal

A Soft-goal is a condition or state of affairs in the world that the actor would like to achieve [22]. For a Soft-goal *there are no clear-cut criteria for whether the condition is achieved*; it cannot be represented as such into an element of a standard UCD. In a standard UML UCD there is no element for the representation of softgoals but a refinement of the UCD is included in the *Rational Unified Process* (*RUP*, see [7]) and known as the RUP/UML Business Use Case Model (see [16]). A representation in the UCD would allow us to trace which functional requirement (in the form of a Hard-goal or a Task) *supports* the realization of a Soft-goal. [21] suggests to map the Soft-goal with the RUP/UML Goal because a semantic analysis of both definitions concludes that those represent the same type (or at least closely related) elements. This solution is relevant for us since it allows a graphical representation of Soft-goals in the UCD as well as a potential support analysis (by highlighting which UC contributes to the satisfaction of the represented Soft-goal). Consequently and even though it is not standard UML, we map the Soft-goal elements from the US set to the graphical representation of the business Goal element. As shown in Fig. 2, we can have:

- The Soft-goal in the WHAT dimension. Then, in the UCD, we immediately relate the Actor (Role in the US WHO dimension) to a Business Goal (Soft-goal in the US WHAT dimension) and a simple link is used;
- The Soft-goal in the WHY dimension. Then, in the UCD, if the element in the WHAT dimension leads to a UC (Hard-goal or a Task in the US), it can be linked to the Business Goal (Soft-goal in the US WHAT dimension) using a $<<$ *support* $>>$ dependency relationship. This link visually expresses that



(a) Use-Case Model Forwarded on the Basis of Stereotyped User Stories

(b) Partial Use-Case Diagram for the Carpooling Example

Fig. 2. Use case diagram: Canonical Form and Carpooling Example

the functional element contributes to the realization of the Soft-goal within the software implementation.

6 Automating the Approach and Round-Tripping Between Views

In order to support the approach, we have built an add-on to the cloud version of the Descartes Architect CASE-Tool [1] that, for the present purpose, allows multiple views:

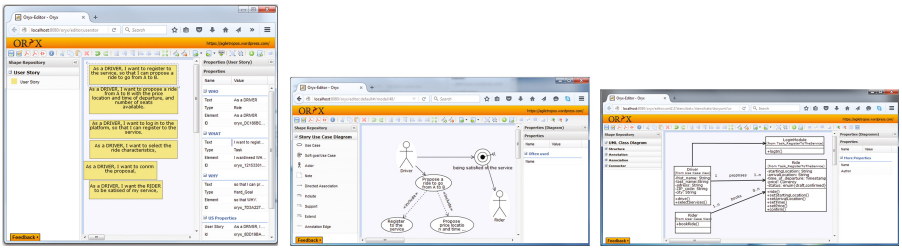
- the *User Story View* (*USV*) to edit US through virtual US cards. Each US element in a dimension must be tagged with a concept of the unified model;
- the *Use-Case View* (*UCV*) to edit a UCD. The UCD is automatically transformed from the US set defined in the USV. When changes are made to UC or Actors in the UCV, the corresponding elements are automatically updated in the USV and vice-versa. These indeed are the same logical element represented in multiple views;
- the *Class, Sequence and Activity Diagram Views* (outside the scope of this paper).

The CASE-Tool immediately build elements in the UCV, when elements are built in the USV following the rules given in this paper and summarized in Table 2.

The editing process is continuous over the requirements analysis stage and over the entire project life cycle. In practice, US elements are re-tagged several times when they analyzed and structured. Consistency among views is ensured by separating the conceptual element in the CASE-tool memory from its representation in a view (Fig. 3).

Table 2. Mapping a US set with the UCD

US Set Element	UCD Element
Role	Actor
Hard-goal	Use Case; several Use Cases transformed from Hard-goals can be linked through $<<include>>$ dependencies
Task	(Possible) Use Case; the Use Case transformed form a Task should be linked through $<<include>>$ or $<<extend>>$ dependencies with Use Cases transformed from Hard-goals
Capability	No possible transformation
Soft-goal	RUP/UML Business Goal



(a) User Story View (b) Use-Case View (c) Class Diagram View

Fig. 3. The supporting CASE-Tool.

7 Impact on Produced Software: Future Work

Two types of impacts will be evaluated in future work:

– *What is the impact on the software design of using our approach versus using another one?* Since we aim to transform coarse-grained US elements in UC, the produced UC are likely to become the scope elements for which realization scenarios will be build. Also, sets of US are expressed in a fined-grained way only or parts of the requirements are not expressed. In order to fill the gap, one or more UC can then be added; US are then automatically generated accordingly. Fine-grained elements can also be omitted from the US set and be identified through the approach. Finally, identifying and representing soft-goals in the UCD may lead to better take them into account in the design. These aspects need further investigation;
– *What will be the variability in the UCD produced from the same US set by different modelers?* Various modelers applying the transformation will not produce exactly the same model. They are indeed likely to interpret elements differently and consequently tag them differently. Analysis activities occurring after the initial transformation often lead to reconsider some of the associated tags (granularity of US elements is thus not set once and for all but refined

through the requirements elicitation). This variability needs to be studied and evaluated.

8 Validity and Threats to the Validity: Future Work

As already evoked, the prerequisite to the use of our approach is to tag the US when setting them up. In terms of time, the investment for disposing of first input models is limited to the tagging and encoding in the CASE-tool. A few threats to the validity will also be highlighted; these should be clarified in later validation of the work:

- *The accuracy in US tags.* [18] studies the perception of US elements' granularity using the unified model. The study distinguished different groups of users from students to software development professionals. The models produced by experienced modelers where more accurate, but identifying granularity did not lead to major issues in any group with the condition that the set of US was taken as a consistent whole. A new experiment with the UML UCD will be performed;
- *The accuracy of the UCD with respect to the system-to-be.* In order to asses the validity of the UCD, we will proceed to the following experience. At first, subjects (part of 3 groups: researchers, students and business analysts) will be informed about a case and asked to carefully read and tag a set of US. Secondly, these same subjects will be asked to rank their perceived relevancy of 3 UCD: (UCD1) generated from their own US set tagging (from the first part); (UCD2) generated from an internally validated solution; and (UCD3) randomly generated out of the US set. The perceived relevancy of the UCD will then be evaluated by the subjects.

Future work also includes the application of the full validation of the technique on real-life projects. We will notably proceed to a statistical analysis of the stakeholders perception of the relevancy of the UC model for a project they have worked on in the past. Moreover, they will be interviewed about the value of a consistent UCD complimentary to the US sets. Finally, the application of the technique on large sets of US will allow us to precisely determine the contribution of the method in terms of scalability. Other metrics for the evaluation of UCD will also be envisaged in line with quality elements for US defined by [10].

9 Conclusion

Agile methods use very simple requirements descriptions in the form of US. These are easy to read but difficult to structure leading to the need of visual requirements representations to sort them, understand the system-to-be, dialogue with stakeholders, etc. We have consequently suggested to structure coarse-grained elements found in US sets in a UML UCD. The UCD view is aimed to remain consistent with the set of US, encompassing changing requirements to furnish

the possibility of UC driven development in methods where US sets are the firstly expressed requirement artifact. This work is complementary to previous work focusing on the representation of US sets with GORE models; further work includes the evaluation of the benefits of the integrated use of the UCD and GORE views.

References

1. The descartes architect case-tool (2016). http://www.isys.ucl.ac.be/descartes/
2. Ambler, S.: Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley, New York (2002)
3. Cohn, M.: Succeeding with Agile: Software Development Using Scrum, vol. 1. Addison-Wesley Professional, Boston (2009)
4. Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 language guide. CoRR abs/1605.07767 (2016)
5. Glinz, M.: A glossary of requirements engineering terminology, version 1.4 (2012)
6. Hastie, S., Wick, A.: User stories and use case - don't use both! (2014). http://www.batimes.com/articles/user-stories-and-use-cases-dont-use-both.html
7. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley, Boston (2003)
8. van Lamsweerde, A.: Goal-oriented requirements enginering: a roundtrip from research to practice. In: 12th IEEE International Conference on Requirements Engineering (RE 2004), 6–10 September 2004, Kyoto, Japan, pp. 4–7. IEEE Computer Society (2004)
9. Liskin, O., Pham, R., Kiesling, S., Schneider, K.: Why we need a granularity concept for user stories. In: Cantone, G., Marchesi, M. (eds.) XP 2014. LNBIP, vol. 179, pp. 110–125. Springer, Heidelberg (2014). doi:10.1007/978-3-319-06862-6_8
10. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requir. Eng. **21**(3), 383–403 (2016)
11. OMG: Business process model and notation (bpmn). version 2.0.1. Technical report, Object Management Group (2013)
12. OMG: Omg unified modeling language^{TM}(omg uml). version 2.5. Technical report, Object Management Group (2015)
13. Oscar, S.: Visual Paradigm for UML. International Book Market Service Limited (2013)
14. Patton, J., Economy, P.: User Story Mapping: Discover the Whole Story, Build the Right Product. 1st edn. O'Reilly Media Inc. (2014)
15. Shergill, M.P.K., Scharff, C.: Developing multi-channel mobile solutions for a global audience: the case of a smarter energy solution. In: SARNOFF 2012 (2012)
16. Shuja, A., Krebs, J.: IBM; Rational Unified Process; Reference and Certification Guide: Solution Designer, 1st edn. IBM Press, Upper Saddle River (2007)
17. Van Lamsweerde, A.: Requirements engineering: From System Goals to UML Models to Software Specifications. Wiley, Hoboken (2009)
18. Velghe, M.: Requirements engineering in agile methods: contributions on user story models. Master's thesis, KU Leuven, Belgium (2015)
19. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I.: Unifying and extending user story models. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 211–225. Springer, Heidelberg (2014). doi:10.1007/978-3-319-07881-6_15

20. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I., Poelmans, S.: Building a rationale diagram for evaluating user story sets. In: 10th IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, 1–3 June 2016, pp. 477–488 (2016)
21. Wautelet, Y., Kolp, M.: Mapping i* within UML for business modeling. In: Doerr, J., Opdahl, A.L. (eds.) REFSQ 2013. LNCS, vol. 7830, pp. 237–252. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37422-7_17
22. Yu, E.: Modeling Strategic Relationships for Process Reengineering (Chap. 1–2), pp. 1–153. MIT Press, Cambridge (2011)
23. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering. MIT Press, Cambridge (2011)
24. Yu, E.S.: Social Modeling and i*. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos. LNCS, vol. 5600, pp. 99–121. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02463-4_7