

RESEARCH

Open Access



ScrumOntoBDD: Agile software development based on scrum, ontologies and behaviour-driven development

Pedro Lopes de Souza^{1*} , Wanderley Lopes de Souza¹  and Luís Ferreira Pires² 

* Correspondence: plsouza@estudante.ufscar.br

¹Department of Computing, Federal University of São Carlos, São Carlos, SP, Brazil

Full list of author information is available at the end of the article

Abstract

When developing a Learning Management System (LMS) using Scrum, we noticed that it was quite often necessary to redefine some system behaviour scenarios, due to ambiguities in the requirement specifications, or due to misinterpretations of stories reported by the Product Owners (POs). The definition of test suites was also cumbersome, resulting in test suites that were incomplete or did not at all comply with the system requirements. Based on this experience and to deal with these problems, in this paper, we propose the ScrumOntoBDD approach to agile software development, which combines Scrum, ontologies and Behaviour-Driven Development (BDD). This approach is centred on the concepts and techniques of Scrum and BDD and focuses on the planning and analysis phases of the software life cycle, since the BDD tools currently provide little support to these phases, while most of the problems during the LMS development were found exactly there. We claim that our approach improves the software development practices in this respect. Furthermore, ScrumOntoBDD employs ontologies in order to reduce ambiguities intrinsic to the use of a natural language as a BDD ubiquitous language. In this paper, we illustrate and systematically evaluate our approach, showing that it is beneficial since it improves the communication between members of an agile development team.

Keywords: Agile Software Development, Scrum, BDD, Ontologies, Learning Management Systems, Medical Education, Active Learning Methodologies, Problem-Based Learning

Highlights

- An approach to agile software development that minimises communication problems amongst members of a development team, by combining Scrum, ontologies and BDD
- An illustration of the proposed approach using a software product developed for the Education domain
- A systematic evaluation of the proposed approach using Action Research



© The Author(s). 2021 **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Introduction

In agile software development approaches, requirements and solutions evolve through collaboration between customers and developers. In this research, we focused on two of these approaches, namely *Scrum* [59] and *Behaviour-Driven Development* (BDD) [44]. Scrum prescribes *sprint reviews*, which are meetings involving the development team and the Product Owner (PO) to plan and evaluate sprints, while BDD prescribes the definition of usage scenarios (*behaviour specifications*) upfront as a way to better understand what the software is supposed to do, i.e., its ‘behaviour’.

Ontologies have been recognised as useful artefacts in the analysis and design of Information Systems (IS) in various application domains, such as Languages, Engineering, Health, and Education. According to [21], an ontology can impact an IS in a *temporal dimension*, depending on whether it is applied at design time or runtime, and in a *structural dimension*, depending on the IS components it affects, i.e., application programs, databases, and user interfaces.

This work has been motivated by our experience with the development of a Learning Management System (LMS) called EAMS-CBALM¹ [57] that, differently of current LMSs like Moodle [40], was developed to support active learning methodologies, such as Problem-Based learning [51], and Practice-Based Learning [7] at the Medicine Programme of the Federal University of São Carlos (UFSCar), Brazil. EAMS-CBALM was developed using Scrum, and during its development, it was quite often necessary to redefine some system behaviour scenarios, and consequently the Product Backlog (PB) items, due to misunderstanding of the stories reported by the PO. The definition of test suites was also cumbersome, resulting in test suites that were incomplete or did not comply at all with the system requirements.

In a Systematic Literature Review (SLR), we identified initiatives that used BDD to improve software development in general, like [34, 48, 61, 64]. These initiatives tend to be limited to the definition of a specific ubiquitous language for a given application domain, whereas we aimed at applying BDD in combination with Scrum. In another SLR, we identified initiatives that used ontologies to improve agile software development, like ([33, 35, 36, 62]). Most of these initiatives propose a process or approach to combine ontologies with some agile software methodology, while we aimed at employing ontologies in a broader context, starting from a reference ontology for a given domain, and gradually specialising it so that this ontology could be used in the agile software development for that domain. Finally, we aimed at exploiting the combination of Scrum, BDD and ontologies to improve software development. To the best of our knowledge, this has not been investigated before.

In order to devise solutions to the problems we experienced during the EAMS-CBALM development, in our work, we aimed at answering the following main research questions:

RQ₁: How can the communication between POs and developers be improved?

RQ₂: How can the ambiguities intrinsic to using natural languages to report user stories be reduced?

¹EAMS-CBALM stands for ‘Educational and Academic Management System for Courses Based on Active Learning Methodologies’.

Based on the results of our literature reviews discussed above, we realised that the combined application of BDD, Scrum and ontologies might produce appropriate answers to our research questions, which resulted in our two main hypotheses:

H_1 : Combining BDD with Scrum can improve communication between POs and developers.

H_2 : Employing ontologies can reduce the ambiguities intrinsic to using natural languages to report user stories.

In [66], we verified the first hypothesis (H1), by developing a case study in the context of the EAMS-CBALM, where we employed a ubiquitous language for the Education domain together with BDD scenarios and acceptance tests in order to allow the PO to follow and properly communicate with the developers throughout the development process. In [67], we verified the second hypothesis (H2) by developing another case study in the context of the EAMS-CBLAM, where we employed domain ontologies to describe the UFSCar Medicine Programme in order to reduce the ambiguities caused by using a natural language as a ubiquitous language. In this paper, we generalize and combine our findings from both studies, by defining a novel approach called ScrumOntoBDD that combines Scrum, Ontology and BDD, so that both hypotheses are covered. The detailed definition of the approach and its rigorous evaluation are the main contributions of this paper.

The remainder of this paper is organised as follow: The Background section is the third section of this work; the ScrumOntoBDD approach section presents the approach to agile software development; the Application example section illustrates ScrumOntoBDD with an application example related to the EAMS-CBALM development; the Evaluation section describes an experimental analysis of ScrumOntoBDD, aiming at showing the validity of our hypotheses; the Related work section discusses some related work, grouped according to the SLRs we carried out; finally, the Conclusions section gives our concluding remarks, and directions for future work.

Background

Since our approach combines Scrum, Behaviour-Driven Development (BDD) and ontologies, in this section, we briefly introduce these techniques and their main characteristics.

Scrum

Nowadays, it is difficult to predict how a software system should evolve, since market conditions and customer needs change rapidly, and new technologies constantly emerge. Agile development methods have been proposed to help software developers deal with these problems [2]. The well-known agile methods include Extreme Programming [4], Crystal [9], Adaptive Software Development [26], DSDM [70], Feature Driven Development [49], and Scrum [59] that is one of the most popular agile software development methods. Scrum emphasises the interactions between users and developers, mainly when the requirements are established and selected as *user stories*, in order to obtain a fast delivery and a satisfactory quality of the product. Most of the clarification

and details regarding the product under development are obtained during these interactions, allowing product adaptations to be done quickly, thus avoiding bottlenecks and delays and reducing the likelihood of unsatisfactory results.

In Scrum, the units of work are divided in *sprints*, which can last between a week and a month, the period in which the developers create an increment of the product to be delivered to the customer. According to [56], Scrum is a framework for organising and managing team work, and its practices involve specific roles, events and artefacts (see Fig. 1).

The development of a product by employing Scrum may involve one or more Development Teams (DT), each one composed of people that play at least the following three Scrum roles:

- Product Owner (PO)*, who can be designated by the customer, is responsible for what will be developed and in what order, and for tracking product development;
- Scrum Master (SM)* is responsible for promoting and supporting Scrum by helping the team understand its theory, practices, rules, and values; and
- Scrum Team (ST)* is composed by professionals responsible for delivering at each sprint a usable increment of the product requested by the customer.

During the Scrum development cycle, the following main artefacts are produced:

- Product Backlog (PB)* is a prioritized list of requirements stated as user stories, broken down (*grooming*) into a set of features, which reflects the PO vision of the product to be created;
- Sprint Backlog (SB)* is a list containing a subset of PB items, followed by the tasks to be performed for each item, which the ST believes and commits to complete on that sprint; and
- Product Increment (PI)* is a potentially shippable increment of the product, representing part of the PO's product vision, which is produced by ST at the end of sprint execution, and can be released to the customer.

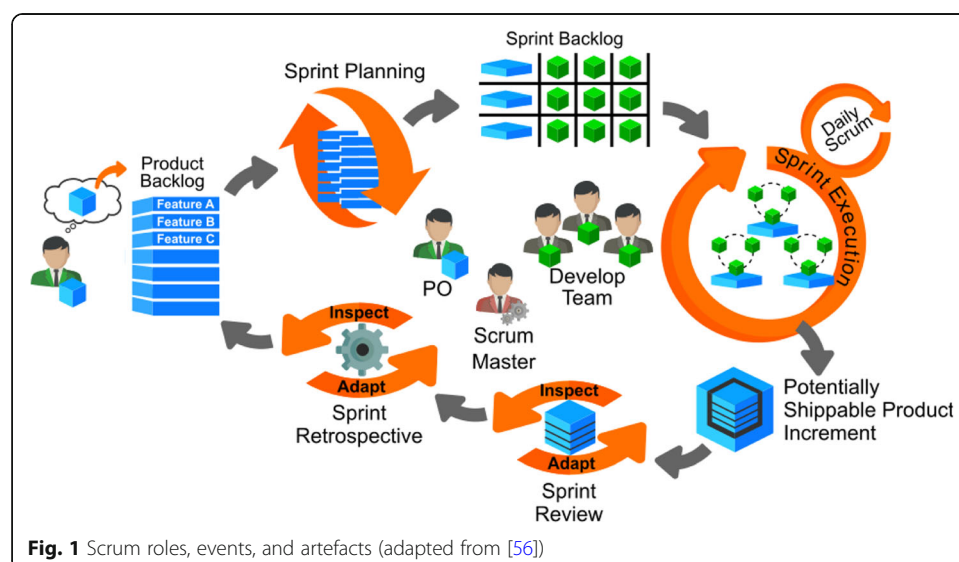


Fig. 1 Scrum roles, events, and artefacts (adapted from [56])

In order to produce these artefacts, the following main events take place:

- a) *Sprint Planning* is a meeting involving SM and ST for planning the work to be performed in that sprint, which can last a maximum of 8 h for a 1-month sprint or less for shorter sprints;
- b) *Sprint Execution*, where the ST guided by the SM performs the needed tasks to get the selected features done, and to produce a potentially shippable PI;
- c) *Daily Scrum* is a ST 15-min meeting held every day of the sprint for planning the work within the next 24 h, which allows for optimizing team collaboration and performance as the work produced since the last daily scrum is scrutinized, and sprint's next work is predicted;
- d) *Sprint Review* is a DT meeting held at the end of the sprint, which can last a maximum of 4 h for a 1-month sprint or less for shorter sprints, for inspecting the PI produced on that sprint, and for adapting the PB if needed; and
- e) *Sprint Retrospective* is a meeting held after the Sprint Review and prior to the next Sprint Planning, which can last a maximum of 3 h for a 1-month sprint or less for shorter sprints, where the DT inspect itself and create an improvement plan to be applied during the next sprint.

Scrum defines roles, artefacts and events that structure the development process from conception to implementation. Scrum can be also used as a container of other development techniques and practices that can be applied in each sprint to produce the PI [59]. We exploited this particular possibility in our work, by prescribing how BDD and ontologies can be applied to improve communication in Scrum-based development.

Behaviour-Driven Development (BDD)

Test-Driven Development (TDD) [3] is a software evolutionary development methodology, based on short development cycles, in which automated tests are described before to the production of code. Acceptance Test-Driven Development (ATDD) [31] is a TDD variation, where software development is driven by acceptance tests, which are used to represent stakeholder's requirements.

Behaviour-Driven Development (BDD) [44] is an agile software development methodology that is considered as an evolution of TDD and ATDD, whose fundamental principle is: "stakeholders and developers should refer to the same system in the same way". For achieving this goal, a ubiquitous language is required that is understandable by all those involved in system development [15], and enables teams to produce executable granular specifications of the system's behaviour and testing [13].

Six main characteristics of BDD were identified in [65]: ubiquitous language, iterative decomposition process, user story and scenario templates, automated acceptance testing with mapping rules, readable behaviour-oriented specification code, and behaviour-driven at different phases. Using these characteristics, [65] also analyses seven of the main BDD tools: NBehave [43] and JBehave [28]; MSpec [41] and RSpec [54]; StoryQ [71]; Cucumber [11]; and SpecFlow [69]. Based on this analysis, we decided to concentrate on JBehave, since it covers most of the BDD characteristics defined in [65] and has appropriate support and tooling.

However, BDD is a relatively novel and evolving methodology, and as such it still lacks a clear definition and common understanding. According to [65], the seven BDD tools analysed focus mainly on the implementation phase of a software project, provide limited support for the analysis phase, no support for the planning phase, and no support for the *Ubiquitous Language Definition*. Okolnychyi's study [46] analysed five BDD tools (some different from [65]) and concluded that only Concordion [10] supports the *Creation of a Ubiquitous Language*, but does not support a *Predefined Ubiquitous Language*.

Starting from [65], we did a SLR of BDD-related work, and an analysis of the current BDD toolkits. Figure 2 shows the BDD process depicted using the Structured Analysis and Design Technique (SADT) diagrammatic notation [55].

The BDD analysis phase starts by identifying the most expected system behaviours, which are derived from the business outcomes to be produced by the system. Based on these business outcomes, feature sets are defined, where each feature set can contain subsets, and indicates what should be accomplished to achieve a specific business outcome. This initial analysis may be sufficient for a first implementation, even if important system behaviours are yet undisclosed. However, as new behaviours are revealed, the whole process illustrated in Fig. 2 can be performed again, thus characterizing the iterative decomposition process described in [65].

Features and their acceptance criteria are expressed through user stories and describe the interactions between a user and the system. A user story should elucidate the user's role in this story, the feature desired by the user, and the benefit gained by the user if the system provides the desired feature. Due to different contexts, a user story may have different versions that will lead to different story instances, called scenarios, which in turn should describe specific contexts and outcomes of this user story. A scenario

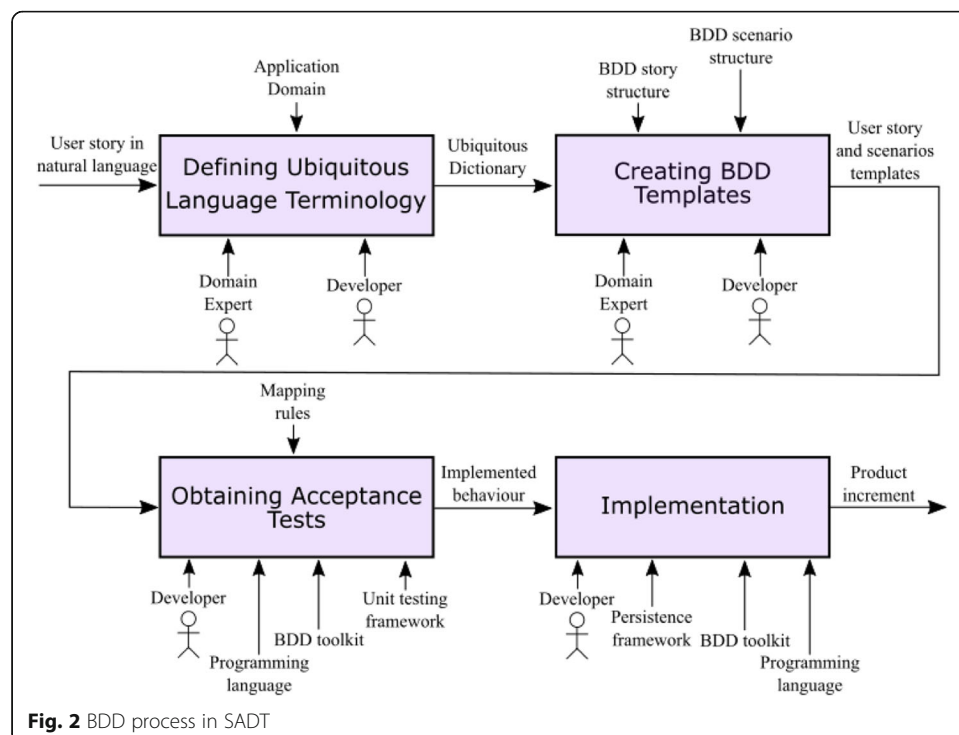


Fig. 2 BDD process in SADT

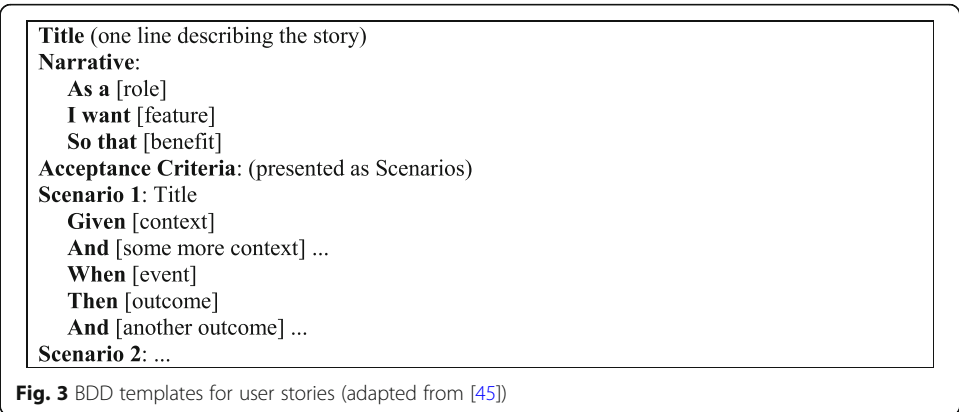
describes how the system that implements a given feature should behave when it is in a main context, with possible additional contexts that could be represented by additional conditions. When an event happens (e.g., a system entry), the scenario's result can be one or more actions that change the state of the system or produce some system output.

BDD user stories and scenarios follow the predefined templates described in [45] by employing a simple ubiquitous language. Figure 3 shows a BDD template.

According to these templates, a User Story is described with a **Title**, a **Narrative** and a set of **Scenarios** representing the **Acceptance Criteria**. The **Title** provides a general description of the story, making reference to a feature that this story represents. The **Narrative** is divided in three clauses: (i) **In order to** describe an activity to be performed by a user; (ii) **As a** correspond to the role played by the user on that story; and (iii) **I want** to show the benefit obtained by the user once the activity is performed. The **Acceptance Criteria** are defined through a set of **Scenarios**, each one with a **Title** and three main clauses: (i) **Given** to provide main context that could be represented by a system state; (ii) **When** to describe a specific event that will trigger the Scenario; and (iii) **Then** to present the main outcome that could be an action for changing the system state. Each one of these clauses can include an **And** statement to provide multiple contexts, events and/or outcomes. Each statement in this representation is called a step.

A BDD acceptance test is an executable specification of the system behaviour, which verifies the interactions or behaviours of objects rather than their states. The produced scenarios are translated into tests that guide the implementation. A scenario is composed of several steps, where each step is an abstraction that represents the three elements of a scenario: context, event and action. The meaning of these elements is that in a particular case of a user story or context C , when the event X happens, the system response should be Z . Each step is mapped to one test method, and the scenario passes only if all its steps pass. Each step follows the TDD's process: red, green, and refactoring to make it pass. Since in BDD all scenarios must be executed automatically, the acceptance criteria must also be imported and analysed automatically. The classes that implement the scenarios read their specifications, which are written in the ubiquitous language, and execute them. Therefore, the mapping between scenarios and testing code needs to be explicitly defined.

In BDD, the code must be readable, contain the specification, describe the behaviour of the objects and be part of the system documentation. The classes and methods



```
Title (one line describing the story)
Narrative:
  As a [role]
  I want [feature]
  So that [benefit]
Acceptance Criteria: (presented as Scenarios)
Scenario 1: Title
  Given [context]
  And [some more context] ...
  When [event]
  Then [outcome]
  And [another outcome] ...
Scenario 2: ...
```

Fig. 3 BDD templates for user stories (adapted from [45])

names must be sentences, and the names of the methods must indicate their functionality. Mapping rules assist in the production of readable behaviour-oriented code and ensures that classes and methods names are the same as user story titles and scenarios, respectively. In addition, these names make use of the ubiquitous domain-specific language defined in the project.

Ontologies

The term ontology first appeared at a computer-related conference in 1967 [38]. Ontologies have been used in Computer Science as artefacts that formally capture conceptualizations, so that the objects and concepts that form these conceptualizations are properly represented and supported [19]. These conceptualizations correspond to some real-world phenomena of interest, which have to be observed or possibly acted upon. An ontology as an artefact needs to be represented with an ontology language [5], and it specifies the concepts and their relationships necessary for its purpose and domain, introducing in this way a vocabulary and the formal constraints that apply to the elements that make sense in the real world [20]. Ontologies have been used with benefits in areas like Databases, Information Systems, Artificial Intelligence and Software Engineering [63].

Ontology classification

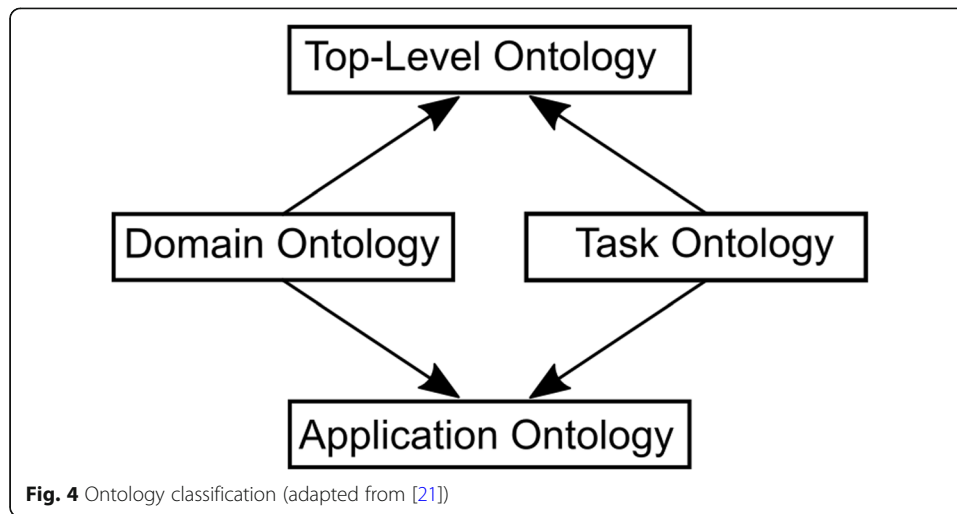
Guarino [21] classifies ontologies according to their generality and dependence levels as follows:

- a) *Top-level ontology*, also called upper or foundation ontology, describes general concepts (e.g., Object, Property) that are independent of a particular domain;
- b) *Domain ontology* describes the vocabulary related to a generic domain (e.g., Medicine, Course) by specializing terms introduced in the top-level ontology;
- c) *Task ontology* describes the vocabulary related to a generic task (e.g., Diagnosing, Lecturing), by also specializing terms introduced in the top-level ontology; and
- d) *Application ontology* describes concepts depending on a particular domain and task (e.g., the *roles* played by domain entities while performing the activity of *given a lecture*), which are specializations of the related ontologies.

A top-level ontology aims at supporting semantic interoperability among domain ontologies, by providing a common basis for formulating definitions. The reuse and maintenance of systems supported by domain ontologies usually requires a merge of these ontologies, and if they are derived from the same high-level ontology, this merging can be performed automatically. Figure 4 shows the classification proposed in [21].

Unfortunately, most of the available domain ontologies are not derived from the same top-level ontology. Moreover, different domain perceptions, different usage intentions, and different languages, give rise to specific application ontologies in the same domain that are often incompatible. In order to deal with these problems, application ontologies can be constructed from a domain reference ontology.

We use the Higher Education Reference Ontology (HERO) proposed in [79] to illustrate the concept of domain ontology. This ontology was defined to achieve consensual



knowledge of the university domain and to be shared and reused among several stakeholders. HERO describes several aspects of the university domain, such as organizational structure, staff, and income, which can be used to derive more elaborate domain ontologies for higher education. Figure 5 illustrates the key concepts of the HERO ontology.

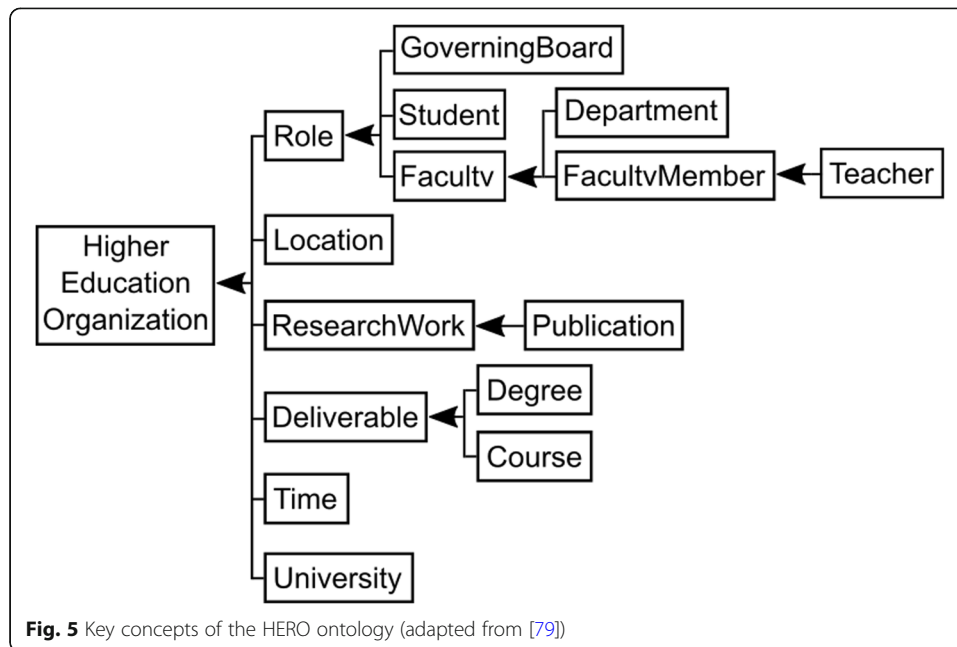
Ontology languages and tools

Several languages have been developed for ontology formal description, allowing the knowledge representation related to specific domains, and often including reasoning rules for processing this knowledge. Most of this effort has been spent to develop ontology languages for the web, and since the web is decentralized, such languages should enable the definition of several vocabularies and their evolution. As stated in [22], “a Semantic Web language must describe meaning in a machine-readable way. Therefore, an ontology language needs not only to include the ability to specify vocabulary, but also the means to formally define it in such a way that it will work for automated reasoning.”

Web Ontology Language (OWL) [74] was defined in 2001 as an extension of *Resource Description Framework (RDF)* [75], which evolved from other ontology languages that existed at that time. We decided to use OWL for specifying our ontologies in our work since OWL is the W3C standard ontology language for the Semantic Web, is used by a large community in many contexts and has appropriate tool support. Figure 6 depicts an excerpt of HERO ontology in OWL, showing some key concepts of this ontology.

When starting out the development of an ontology either from scratch or by reusing existing ontologies, one needs appropriate tools. Ontology tools are useful in the design, implementation and maintenance phases of the ontology life cycle, helping to acquire, organize and visualize domain knowledge before and during the ontology development. In particular, ontology editors allow users to manipulate, inspect, browse and code ontologies. Special attention has been given in recent years to the Semantic Web editors responsible for creating and manipulating ontologies [1].

Protégé [42] was developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine, and allows users to create,



visualize, manipulate and save ontologies in several formats, such as RDE, *RDF Schema (RDFS)* [76] and OWL. We used Protégé [50] in our work because it is a widespread platform recognized for its scalability and extensibility, which provides a tool set for building domain models and knowledge-based applications with ontologies. Furthermore, it is a free and open-source platform that has been continuously improved by Stanford University. According to [24], Protégé is a killer application, since it is a highly transformative technology that creates new markets and behaviour patterns. Figure 7 shows screenshot of the HERO ontology in Protégé, with the class hierarchy on the left-hand side and a graphical representation of this hierarchy on the right-hand side.

```

<!-- http://www.UniversityReferenceOntology.org/HERO#Role -->
<owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#Role">
  <rdfs:subClassOf rdf:resource="&owl;HigherEducationOrganization"/>
</owl:Class>
<!-- http://www.UniversityReferenceOntology.org/HERO#Location -->
<owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#Location">
  <rdfs:subClassOf rdf:resource="&owl;HigherEducationOrganization"/>
</owl:Class>
<!-- http://www.UniversityReferenceOntology.org/HERO#ResearchWork -->
<owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#ResearchWork">
  <rdfs:subClassOf rdf:resource="&owl;HigherEducationOrganization"/>
  <rdfs:isDefinedBy xml:lang="en">Faculty research work comprises a variety of related
  activities, such as plan and perform studies and experiments, [...]</rdfs:isDefinedBy>
</owl:Class>
  
```

Fig. 6 Excerpt of HERO ontology in OWL

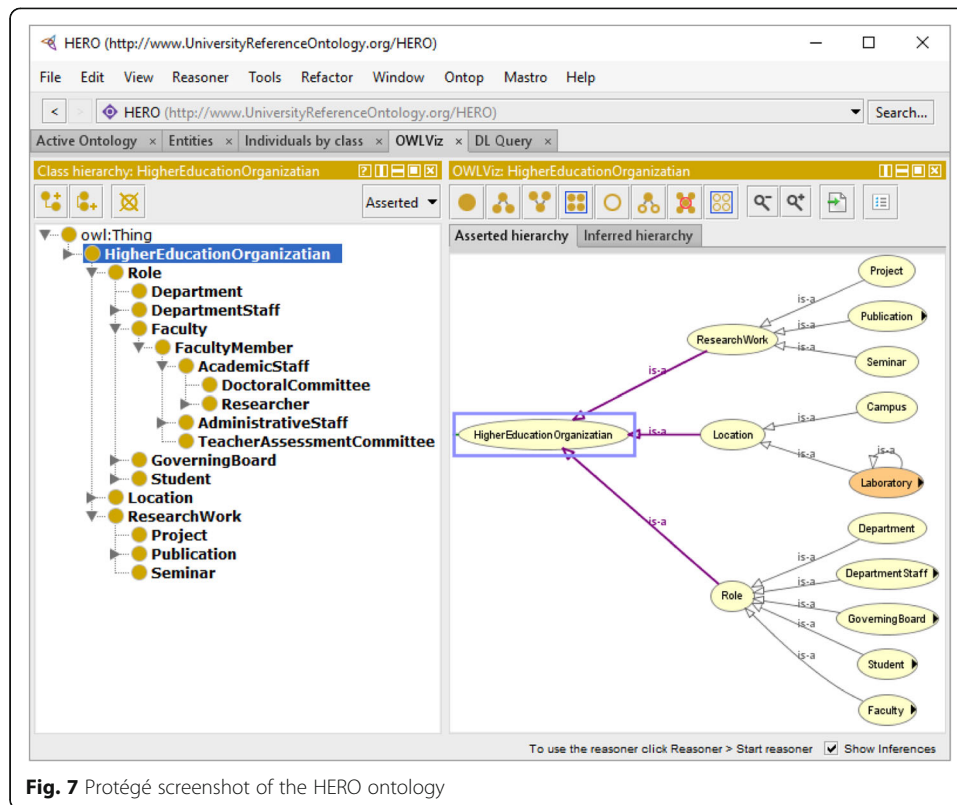


Fig. 7 Protégé screenshot of the HERO ontology

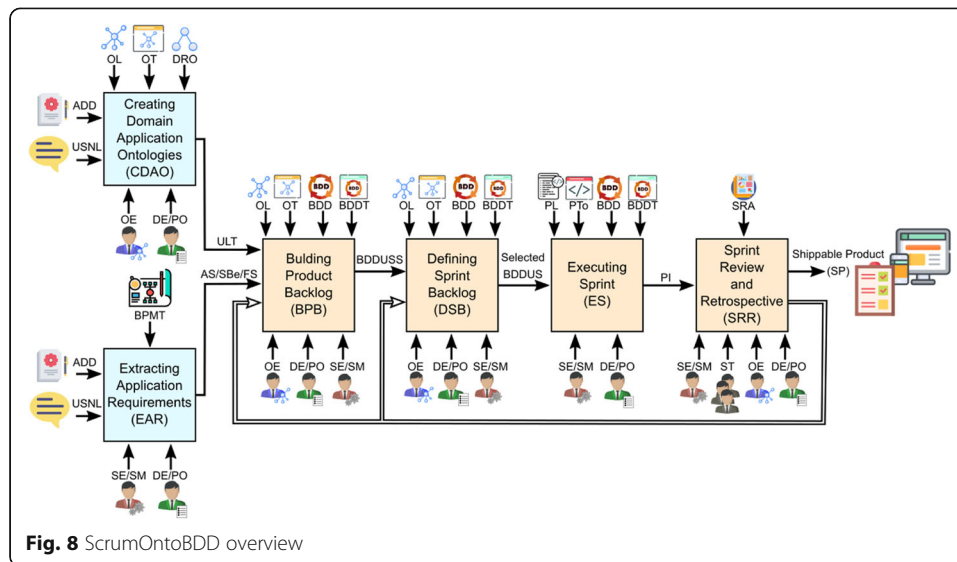
ScrumOntoBDD approach

Based on our experience with Scrum in a software development project, we developed the ScrumOntoBDD approach, which uses Scrum as container and applies BDD and ontologies to improve the communication between POs and developers and to reduce the ambiguities intrinsic to using natural languages to report user stories. This section presents this approach by discussing each of its main roles and activities.

Overview

Figure 8 gives an overview of the ScrumOntoBDD approach in a SADT diagram, where each rectangle represents a main activity of this approach, and an icon on the left-hand side of an activity represents an activity input, on the right-hand side of an activity represents an activity output, at the top of an activity represents a resource employed in this activity, and at the bottom of an activity represents a role of the Development Team (DT) in this activity.

The main ScrumOntoBDD activities are *Creating Domain Application Ontologies (CDAO)*, *Extracting Application Requirements (EAR)*, *Building Product Backlog (BPB)*, *Defining Sprint Backlog (DSB)*, *Executing Sprint (ES)*, and *Sprint Review and Retrospective (SRR)*. CDAO and EAR can be performed in parallel with some possible overlap. ES can be performed several times until a given PI is achieved, and after SRR is performed and the PI is accepted, a new sprint backlog is defined. The sprint review and sprint retrospective were combined in a single activity since they are performed in sequence



and involve all DT. The *Shippable Product (SP)* is complete after all PB items have been processed successfully.

The inputs and outputs of these activities are *Application Domain Documents (ADD)*, *User Stories in Natural Language (USNL)*, *Ubiquitous Language Terminology (ULT)*, *Application Scenarios (AS)*, *System Behaviours (SBe)*, *Feature Sets (FS)*, *BDD User Stories and Scenarios (BDDUSS)*, *Selected BDDUSS* and *Product Increment (PI)*. The main resources used to perform these activities are *Ontology Languages (OL)*, *Ontology Tools (OT)*, *Domain Reference Ontologies (DRO)*, *Business Process Modelling Techniques (BPMT)*, *BDD*, *BDD Tools (BDDT)*, *Programming Languages (PL)*, *Programming Tools (PTo)* and *Sprint Records and Annotations (SRA)*. DT should be multidisciplinary, and team members should play the following roles:

- Domain Expert (DE)* is responsible for sharing his/her knowledge of the application domain, and for transferring the necessary information to create the ontologies and to capture the application requirements;
- Ontology Engineer (OE)* is responsible for creating, manipulating and evaluating the domain application ontologies, and also for assisting the formalization of knowledge and of software requirements;
- Software Engineer (SE)* is responsible for tracking the entire software life cycle, ensuring the proper working according to the application requirements, and the generation of quality artefacts; and
- Product Owner (PO)*, *Scrum Master (SM)* and *Scrum Team (ST)* play the corresponding Scrum roles, but SM and ST must be able to apply BDD and BDD tools.

Creating Domain Application Ontologies (CDAO)

The CDAO activity can start from an existing reference domain ontology or build a domain application ontology from scratch. In the former case, the reference domain ontology must be gradually specialized to obtain an ontology that covers a terminology

Table 1 Inputs, outputs, resources, roles and tasks of CDAO activity

Inputs	<ul style="list-style-type: none"> • Application Domain Documents (ADD) • User Stories in Natural Language (USNL)
Outputs	<ul style="list-style-type: none"> • Ubiquitous Language Terminology (ULT)
Resources	<ul style="list-style-type: none"> • Ontology Languages (OL) • Ontology Tools (OT) • Domain Reference Ontologies (DRO)
Roles	<ul style="list-style-type: none"> • Ontology Engineer (OE) • Domain Expert (DE) • Product Owner (PO)
Tasks	<ul style="list-style-type: none"> (a) Search ontologies to select or create a DRO for the application domain; (b) Successively specialize DRO based on application domain characteristics; (c) Validate the created domain application ontologies; and (d) Extract the main ULT from these ontologies.

(ubiquitous language) that is appropriate for the development of the intended application. Table 1 summarizes the inputs, outputs, resources, roles and the main tasks of this activity.

Extracting Application Requirements (EAR)

The EAR activity starts by identifying the customer needs, the objective, scope and scenarios of the application. These scenarios are then described using a Business Process Modelling Technique (BPMT) [72], such as UML activity diagram, Business Process Model Notation (BPMN) [47] or flowchart, to represent the key application requirements. Based on this description, the main software functions are defined, and the system behaviours and features are described. Table 2 summarizes the inputs, outputs, resources, roles and the main tasks of this activity.

Building Product Backlog (BPB)

The BPB activity starts by defining the user stories and scenarios that will be initially stored in the PB, using the ubiquitous language, the application scenarios, and the system behaviours and features. These items must be formally specified, by means of an ontology model that employs BDD templates, and they should be also listed in a priority order for implementation, building a formally described PB. Table 3 summarizes the inputs, outputs, resources, roles and the main tasks of this activity.

Table 2 Inputs, outputs, resources, roles and tasks of EAR activity

Inputs	<ul style="list-style-type: none"> • Application Domain Documents (ADD) • User Stories in Natural Language (USNL)
Outputs	<ul style="list-style-type: none"> • Application Scenarios (AS) • Feature Sets (FS) • System Behaviours (SBe)
Resources	<ul style="list-style-type: none"> • Business Process Modelling Techniques (BPMT)
Roles	<ul style="list-style-type: none"> • Software Engineer (SE) • Scrum Master (SM) • Domain Expert (DE) • Product Owner (PO)
Tasks	<ul style="list-style-type: none"> (a) Identify customer needs and application characteristics; (b) Draw up AS; (c) Define the main software functionalities; and (d) Draw up FS and SBe.

Table 3 Inputs, outputs, resources, roles and tasks of BPB activity

Inputs	<ul style="list-style-type: none"> • Ubiquitous Language Terminology (ULT) • Application Scenarios (AS) • Feature Sets (FS) • System Behaviours (SBe)
Outputs	<ul style="list-style-type: none"> • BDD User Stories and Scenarios (BDDUSS)
Resources	<ul style="list-style-type: none"> • Ontology Languages (OL) • Ontology Tools (OT) • BDD • BDD Tools (BDDT)
Roles	<ul style="list-style-type: none"> • Ontology Engineer (OE) • Domain Expert (DE) • Product Owner (PO) • Software Engineer (SE) • Scrum Master (SM)
Tasks	<ul style="list-style-type: none"> (a) Define User Stories and Scenarios (USS) using ULT, AS, FS, and SBe; (b) Formally specify the defined USS by means of a User Story and Scenario ontology that employs BDD templates; and (c) List the specified BDDUSS in a priority order thus building the Product Backlog.

Defining Sprint Backlog (DSB)

The DSB activity starts with a sprint planning meeting for defining which PB items will be implemented in that sprint. After that, the tasks to be performed for each one of these items must be described, and the work to accomplish them must be planned. Table 4 summarizes the inputs, outputs, resources, roles and the main tasks of this activity.

Executing Sprint (ES)

The ES activity starts with a daily meeting for analysing the work done until then and planning the work to be done on that day in order to implement the selected user stories and scenarios. During this activity, these scenarios must be translated into BDD acceptance tests to guide the implementation. ES can last several days until a given PI is obtained. Table 5 summarizes the inputs, outputs, resources, roles and the main tasks of this activity.

Sprint Review and Retrospective (SRR)

The SRR activity starts with a DT meeting for inspecting the PI produced on the current sprint. This meeting is followed by another one for retrospection, and for creating an improvement plan to be applied on the next sprint based on the SRA gathered during the current sprint. The

Table 4 Inputs, outputs, resources, roles and tasks of DSB activity

Inputs	<ul style="list-style-type: none"> • BDD User Stories and Scenarios (BDDUSS)
Outputs	<ul style="list-style-type: none"> • Selected BDD User Stories and Scenarios (Selected BDDUSS)
Resources	<ul style="list-style-type: none"> • Ontology Languages (OL) • Ontology Tools (OT) • BDD • BDD Tools (BDDT)
Roles	<ul style="list-style-type: none"> • Ontology Engineer (OE) • Domain Expert (DE) • Product Owner (PO) • Software Engineer (SE) • Scrum Master (SM)
Tasks	<ul style="list-style-type: none"> (a) Select the BDDUSS of the Backlog to be implemented; and (b) Define the tasks and work for implementing the Selected BDDUSS.

Table 5 Inputs, outputs, resources, roles and tasks of ES activity

Inputs	• Selected BDD User Stories and Scenarios (Selected BDDUSS)
Outputs	• Product Increment (PI)
Resources	• Programming Languages (PL) • Programming Tools (PTo) • BDD • BDD Tools (BDDT)
Roles	• Software Engineer (SE) • Scrum Master (SM) • Scrum Team (ST)
Tasks	(a) Obtain BDD acceptance tests for the Selected BDDUSS; and (b) Perform the tasks to implement the Selected BDDUSS thus producing a Product Increment (PI).

only resource needed for the SRR activity is the SRA, but the participation of the entire DT is required, including the OE and the DE, since it may be necessary to adapt the PB. Table 6 summarizes the inputs, outputs, resources, roles and the main tasks of this activity.

Application example

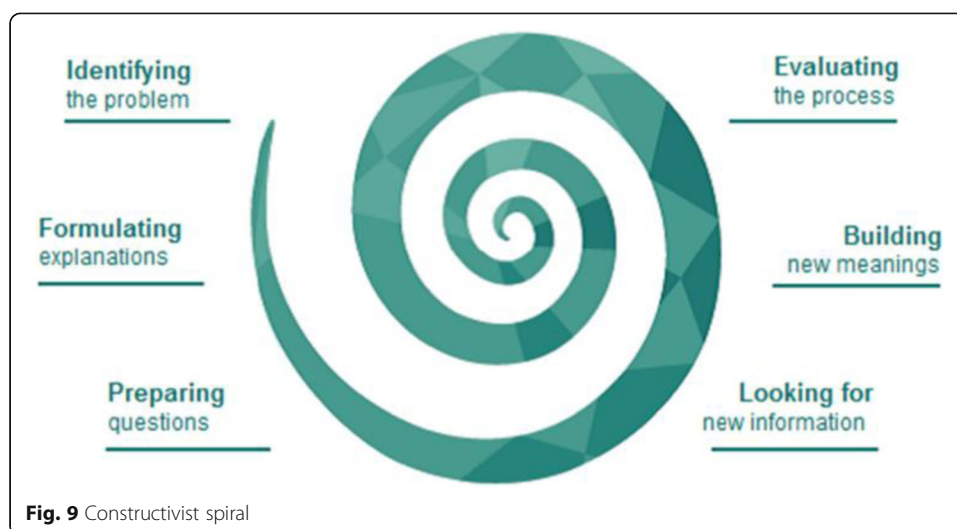
In this section, we illustrate our approach with an example taken from the EAMS-CBALM development. We start with a short introduction of the application context in which EAMS-CBALM is expected to operate, and we show how each activity prescribed in our approach has been performed to obtain a PI that supports one specific EAMS-CBALM requirement, namely the *Instrument Types Registration* requirement of the EAMS-CBALM *Evaluation Management* module.

Application context

Different teaching-learning processes may be involved in education, and they can occur anytime and anywhere [8]. Most Brazilian universities still employ traditional teaching-learning methods based on frontal lectures in their education programmes. The Medicine Programme of the Federal University of São Carlos (UFSCar) was established in 2006 and has been developed to brake with this tradition, by following a socio-constructivist educational approach, with a competence-oriented pedagogical programme, and employing active learning methodologies, such as Problem-Based Learning [51] and Practice-Based Learning [7]. Educational activities in the UFSCar Medicine Programme have been designed based on the constructivist spiral depicted in Fig. 9. These activities have *learning triggers*, which are problems simulating or

Table 6 Inputs, outputs, resources, roles and tasks of SRR activity

Inputs	• Product Increment (PI)
Outputs	• Shippable Product (SP)
Resources	• Sprint Records and Annotations (SRA)
Roles	• Software Engineer (SE) • Scrum Master (SM) • Scrum Team (ST) • Ontology Engineer (OE) • Domain Expert (DE) • Product Owner (PO)
Tasks	(a) Inspect the produced PI and adapt the PB if needed; and (b) Inspect the DT and create an improvement plan for the next sprint.



portraying the daily activities to be exercised by the graduates of this programme. These triggers activate the movements of the constructivist spiral by encouraging students to reflect and stimulating them to develop their capacities. From each learning trigger, the students traverse all these movements, starting by *identifying the problem*, then *formulating explanations*, *preparing learning questions*, *looking for new information*, *building new meanings* and finally *evaluating the process*.

The *Reflective Portfolio* (RP) is a tool for recording and monitoring student activities that is often used in courses that employ active learning methodologies, since it is a flexible and evidence-based instrument that motivates the students to perform continuous reflection, and collaborative learning analysis [80]. In the UFSCar Medicine Programme, a reflective portfolio is used in all student activities, both internal and external (outside the university campus).

In the early years of the programme, RPs were not at all automated, i.e., they were recorded 100% on paper. This has triggered the Ubiquitous Computing Group (UCG) of UFSCar to launch the electronic reflective portfolio project [17]. EAMS-CBALM has been developed as a spin-off of this effort, in partnership with the UFSCar Department of Medicine (DMed), the Teaching and Research Institute (TRI) of the Sírio-Libanês Hospital (SLH), and a software house.

EAMS-CBALM has been developed using Scrum in a project that took 1 year, during which weekly meetings were held at the software house in São Carlos-SP for the planning and analysis of its different phases. These meetings involved the UCG/UFSCar coordinator, the developer team coordinator of the software house, two POs designated by TRI/SLH, and a PhD student. Monthly meetings were held at TRI/SLH to present the PIs coming out of the sprints, involving the participants of the software house meetings and TRI/SLH members directly or indirectly related to the project. In addition, the PhD student participated in the courses offered by DMed /UFSCar and TRI/SLH to observe the teaching-learning process of these courses.

During the software house meetings, the POs reported stories, informally in Portuguese, describing what the user activities to be supported by EAMS-CBALM. From these stories, system requirements were then captured and specified also informally in Portuguese. Using these requirement specifications, the software house development

team defined system behaviour scenarios and implemented the screen prototypes for these scenarios, which were presented and discussed at the next meeting. These meetings totalled 109 h, through which the CBALM teaching–learning process was understood, and the functional and non-functional system requirements were defined, as well as the system architecture.

The EAMS-CBALM has two main functional modules: the *academic module* and the *pedagogical module*. The academic module allows its users to register the programme structure, the CBALM community and the curriculum. Since this module is less relevant to our illustration, we refrain from discussing it in more detail.

Figure 10 illustrates the pedagogical module, which stores the curricular structure of each class (school year) of each course previously recorded in the academic module. The educational actions are programmed in the pedagogical module, giving rise to the educational environment in which students and teachers perform and record their activities. The planning of an educational action starts with preparation of meetings, where each meeting corresponds to a movement of the constructivist spiral, and ends with the preparation of evaluations.

EAMS-CBALM allows students and teachers to discuss through forums, exchange text documents, images, and video and audio fragments, create individual or collaborative documents, store and retrieve document versions, and share the knowledge produced by other students and teachers. All these resources contribute to the development of student capabilities, leading to more knowledge production. EAMS-CBALM has some other interesting features (e.g., data visualization), but we refrain from discussing them in more detail here since they are not relevant to our illustration.

During the EAMS-CBALM development, the *Evaluation Management module* (sub-module of the pedagogical module) generated the most controversies between PO and developers. This module supports 14 functional requirements, and especially the scenarios of the *Instrument Types Registration* requirement have been redefined several times, but the final implementation of this requirement did not fully satisfy the PO.

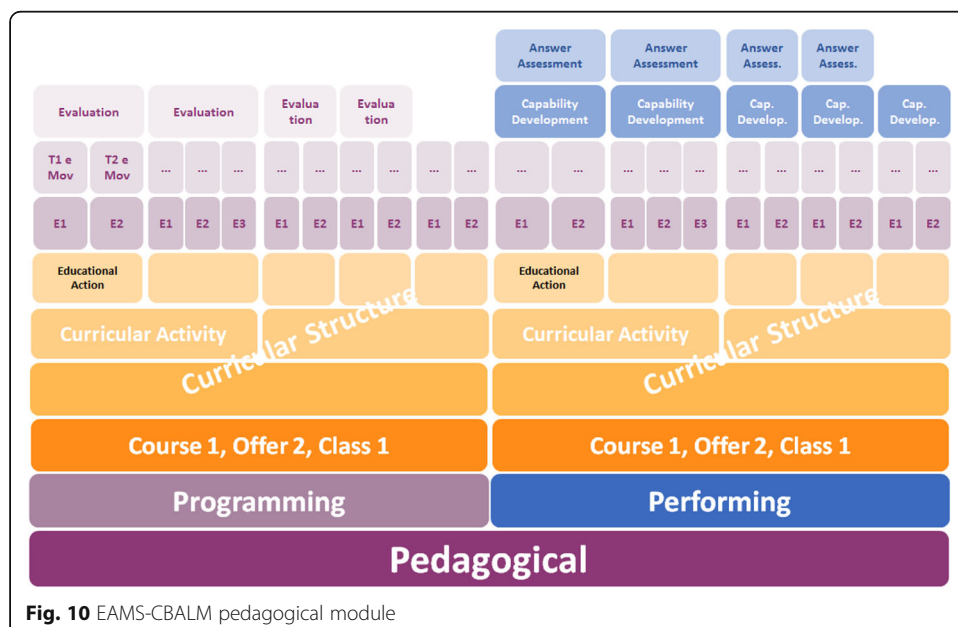


Fig. 10 EAMS-CBALM pedagogical module

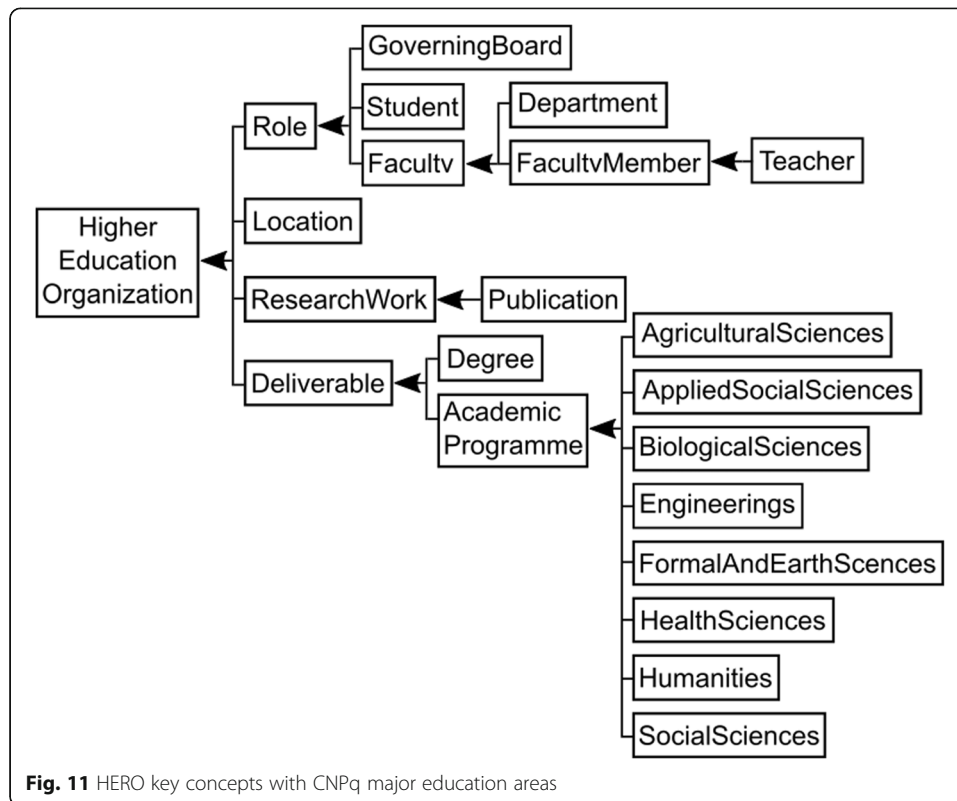
Therefore, we selected this module and this specific requirement as the example to illustrate ScrumOntoBDD.

Problem description

The student evaluation process of the UFSCar Medicine Programme [73] is more complex than the traditional evaluation processes based on a final exam and/or a project/assignment. The evaluations in this programme are performed by all people involved in the educational activities, expressing their perceptions, indicating the relevant aspects, and aspects that need to be improved, reworked or replaced. Two evaluation types are defined in this process, namely *formative* and *summative* evaluations, and their possible results are *satisfactory*, *needs improvement* and *unsatisfactory*. A student who does not reach a satisfactory result must undertake an improvement plan proposed by the teacher, and after that be re-evaluated. Evaluations are consolidated by applying the following instrument types:

- a) *Performance Assessment of the Teaching-Learning Process* (PATLP): Formative evaluation, where the teacher (respondent) evaluates student. A PATLP has three steps: teacher evaluation, classmate evaluation and improvement plan (if necessary);
- b) *Reflective Portfolio* (RP): Formative and summative evaluation. The monitoring of each student's portfolio by the teacher is part of the formative assessment. The summative evaluation refers to its preparation and presentation according to pre-established delivery dates that must be recorded into the system;
- c) *Written Examination* (WE): Summative evaluation. Each WE consists of questions defined by the teacher, answered by the student, and then assigned by the teacher. Each question is individually analysed in order to determine the student progress. All questions must have a satisfactory result for the student to pass the WE. Questions that failed to get a satisfactory result are considered as progress deficit and are worked out in the next WE;
- d) *Progress Test* (PT): Formative and summative evaluation. A PT consists of multiple-choice questions. The teacher monitors the performance of each student in a PT as part of the formative assessment. A PT is also summative because the students' presence at a PT gives them already a satisfactory result;
- e) *Objective and Structured Evaluation of Professional Performance* (OSEPP): Summative evaluation. Instead of answering questions like in a WE, the students act in clinical cases and are evaluated by the teacher. The OSEPP evaluation is similar to the WE evaluation;
- f) *Problems-Based Exercise* (PBE): Formative evaluation. A PBE assesses the student's individual ability to study and identify health needs, formulate patient and family problems, and propose a healthcare plan for a particular context and problem situation."

In the sequel, we consider the goal of obtaining a shippable PI for the *Instrument Types Registration* requirement of the EAMS-CBALM Evaluation Management module



to illustrate our approach. For this purpose, we show how the activities prescribed in the ScrumOntoBDD approach can be applied to achieve this goal.

CDAO for the UFSCar Medicine Programme

Based on HERO key concepts shown in Fig. 5 and according to the higher education areas described by the Brazilian National Council for Scientific and Technological Development (CNPq), we extended this reference ontology in order to describe the general organisation of the Brazilian universities. Figure 11 shows this extension with the CNPq major educational areas.

We then applied the concepts shown in Fig. 11 to define an ontology that represents all the programmes offered by UFSCar. Figure 12 depicts an excerpt of this ontology in OWLViz (a Protégé plugin) that shows the 8 CNPq large educational areas and focuses on the 10 UFSCar programmes in the Health Sciences area, including the Medicine Programme.

Based on the UFSCar Medicine Programme curriculum description [73], we specialised the UFSCar ontology and obtained the UFSCar Medicine Programme ontology. Figure 13 shows an excerpt of this ontology, focusing on the programme structure, and Fig. 14 shows another excerpt of the same ontology, focusing on the learning methodology, some of the educational activities and the steps of the constructivist spiral.

Evaluation Process ontology

Based on the student evaluation process summarised in the Problem description section, and in accordance with [18], we defined the terminology (names, adjectives and

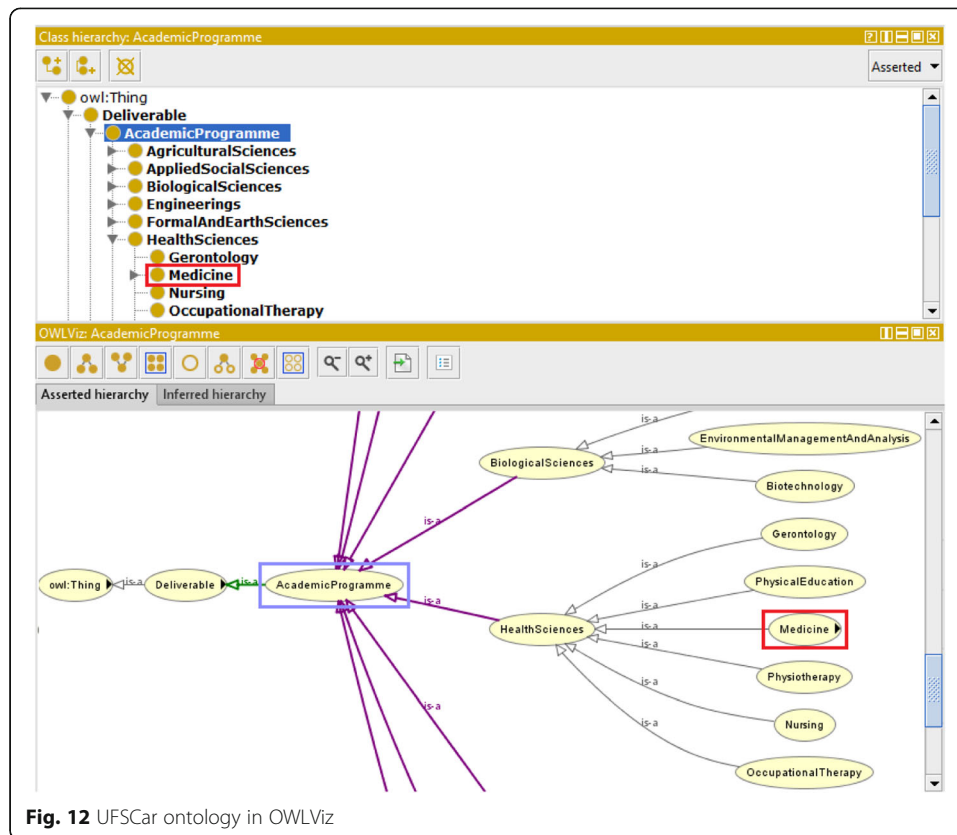


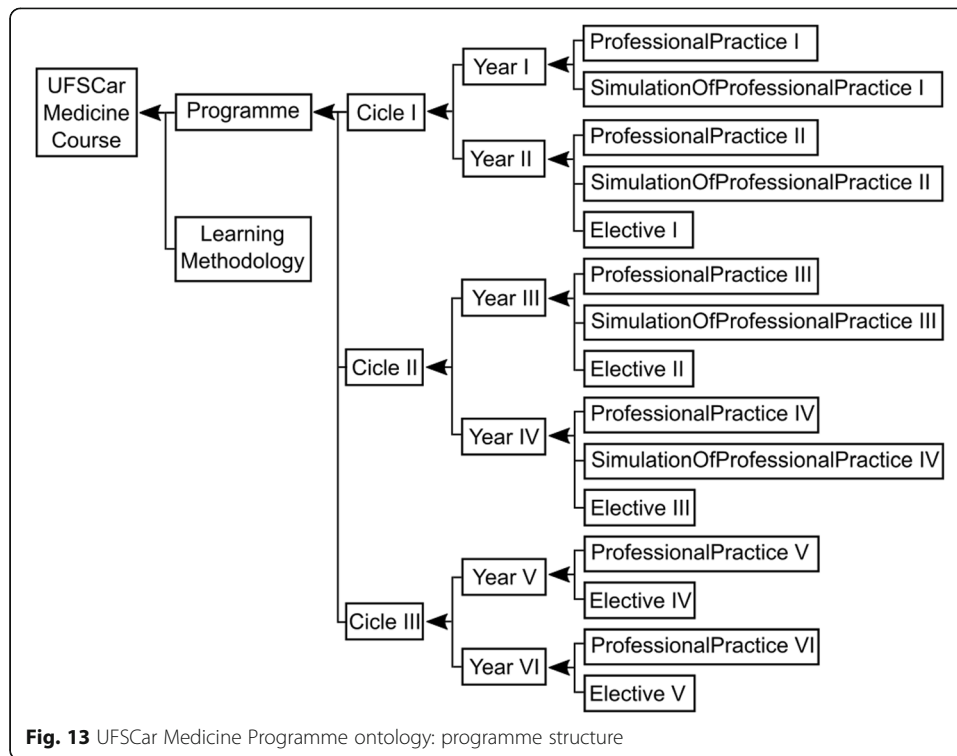
Fig. 12 UFSCar ontology in OWLViz

verbs) to be formally represented in the Evaluation Process ontology of the UFSCar Medicine Programme in terms of concepts, attributes and relations. This ontology is the main Ubiquitous Language Terminology (ULT) used in the development of the EAMS-CBALM Evaluation Management module. Figure 15 shows an excerpt of this ontology, where (i) classes *EducationalActivity* and *EvaluationProcess* model the concept of *CurricularActivity* in the Programme; (ii) each *EducationalActivity* starts with one or more meetings. A *Meeting* has the participation of students and teachers with specific *Roles* and triggers a *LearningTrigger*; (iii) each *LearningTrigger* transverse the *ConstructivistSpiralSteps* and ends with an *EvaluationProcess*; and (iv) an *EvaluationProcess* is consolidated by applying *EvaluationInstruments*.

Figure 16 shows an excerpt of the Evaluation Process ontology of the UFSCar Medicine Programme, focusing on the evaluation instrument types, and showing the PATLP instrument and the relationships between its actors. All instrument types of the programme have been defined in this ontology in a similar way.

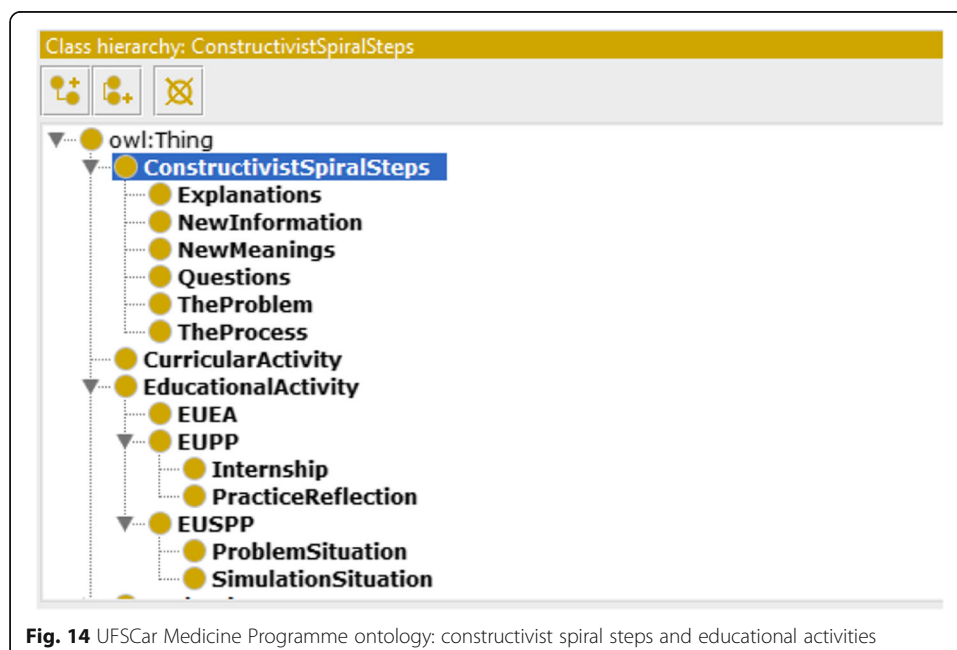
Ontology validation

We assessed the quality of our ontologies according to the structural and functional dimensions [18]. Structural validation considers the ontology logical structure, focusing on its syntax and formal semantics. Protégé offers several ontology verification tools for detecting inconsistencies and redundancies in ontologies. We used the reasoners



FaCT++, HermiT and Pellet to verify our ontologies, and we discovered no inconsistencies nor redundancies.

Figure 17 shows an example of structural validation, where FaCT++, Hermit and Pellet have successfully processed the Evaluation Process ontology in 890, 2125, and 568 ms, respectively. The current version of this ontology bears an amount of 358 axioms,



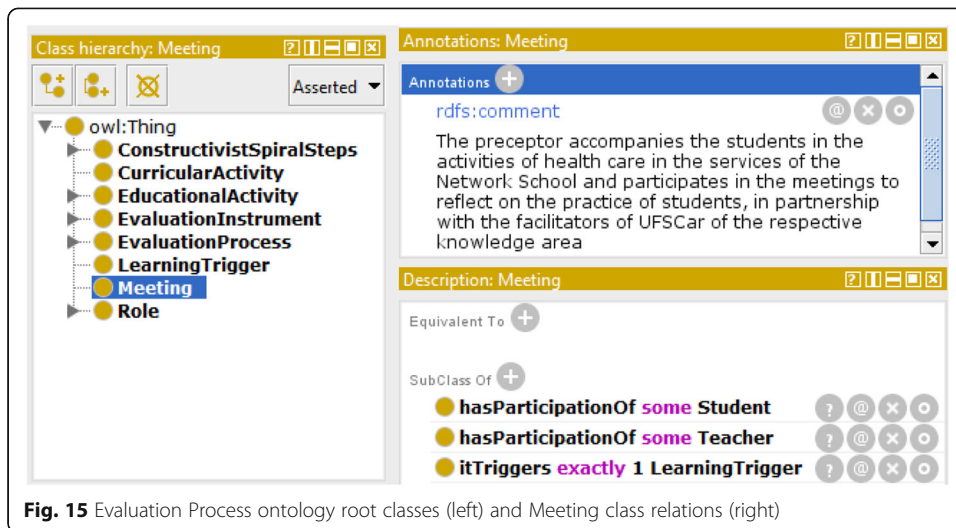


Fig. 15 Evaluation Process ontology root classes (left) and Meeting class relations (right)

182 of them being logical axioms, 42 classes, 24 object properties, 9 data properties and 0 individuals.

Our functional assessment focused on usage evaluation by domain experts, user satisfaction, task assessment and topic assessment. During development, our ontologies were assessed in collaboration with the PO to verify their structure, their restrictions, relations between concepts and the attributes of their concepts. The PO also assessed their compliance with some predefined criteria, such as completeness, reliability and recognition level of the objects, and concepts supported by the ontology. These criteria have been updated according to the PO suggestions.

EAR for EAMS-CBALM

The following user story was reported by the PO during the EAMS-CBALM development concerning the *Instrument Types Registration* requirement:

“In order for the programme coordinator to carry out a student evaluation, the six instrument types have to be previously registered. This requirement is needed because the

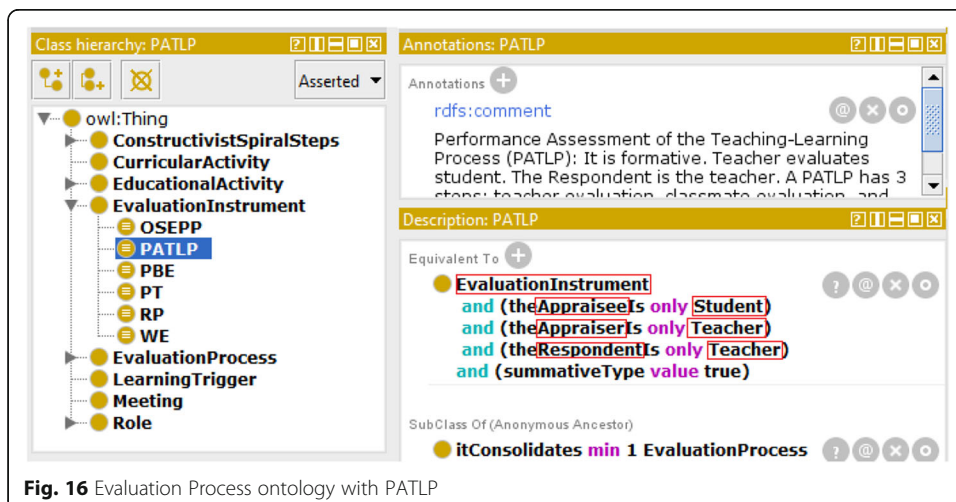


Fig. 16 Evaluation Process ontology with PATLP

```
INFO 15:43:29 ----- Running Reasoner -----
INFO 15:43:29 Pre-computing inferences:
INFO 15:43:29   - class hierarchy
INFO 15:43:29   - object property hierarchy
INFO 15:43:29   - data property hierarchy
INFO 15:43:29   - class assertions
INFO 15:43:29   - object property assertions
INFO 15:43:29   - data property assertions
INFO 15:43:29   - same individuals
INFO 15:43:30 Ontologies processed in 890 ms by FaCT++
INFO 15:43:59 ----- Running Reasoner -----
INFO 15:43:59 Pre-computing inferences:
INFO 15:43:59   - class hierarchy
INFO 15:43:59   - object property hierarchy
INFO 15:43:59   - data property hierarchy
INFO 15:43:59   - class assertions
INFO 15:43:59   - object property assertions
INFO 15:43:59   - data property assertions
INFO 15:43:59   - same individuals
INFO 15:44:02 Ontologies processed in 2125 ms by null
INFO 15:44:14 ----- Running Reasoner -----
INFO 15:44:14 Pre-computing inferences:
INFO 15:44:14   - class hierarchy
INFO 15:44:14   - object property hierarchy
INFO 15:44:14   - data property hierarchy
INFO 15:44:14   - class assertions
INFO 15:44:14   - object property assertions
INFO 15:44:14   - data property assertions
INFO 15:44:14   - same individuals
INFO 15:44:14 Ontologies processed in 568 ms by Pellet
```

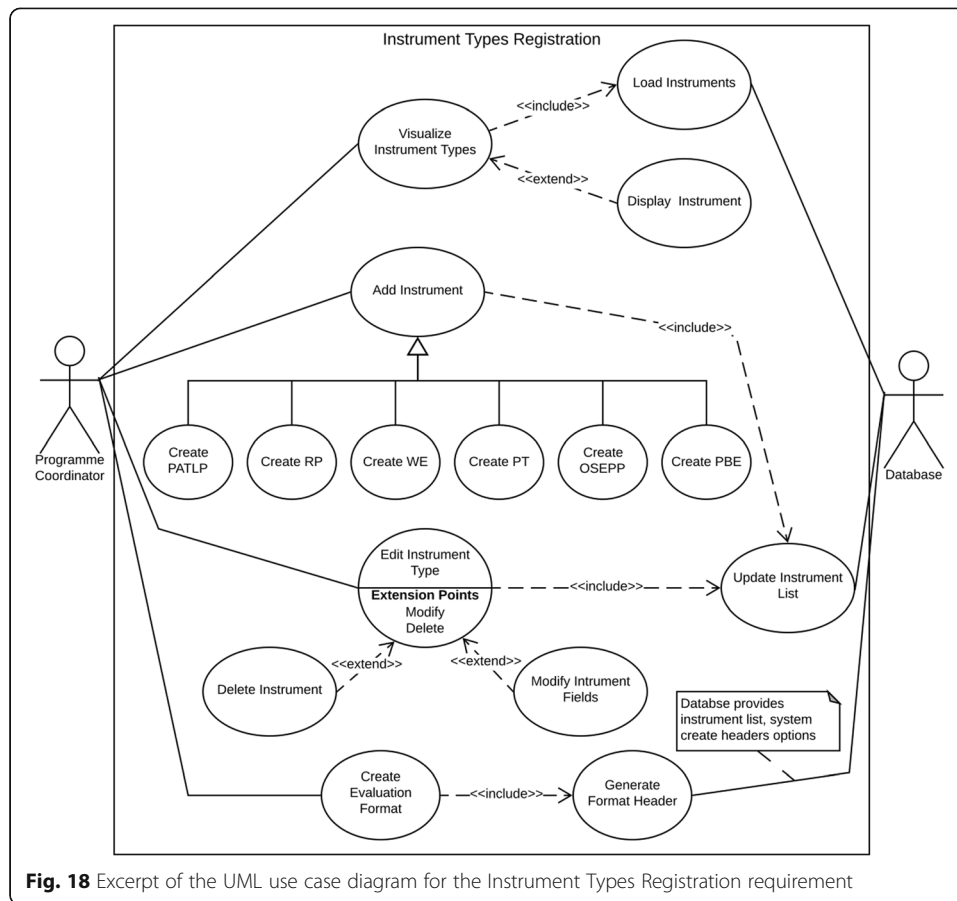
Fig. 17 Results of processing the Evaluation Process ontology: FaCT++ (top), HermiT (middle) and Pellet (bottom)

evaluation form heading changes according to the employed instrument. When registering an instrument, the system must keep the following information: name, acronym and the relationship between who responds to the evaluation, who evaluates and who is evaluated. This last information is crucial since in conjunction with the curricular activity it defines which form type is applied when registering an evaluation.”

Based on this user story, we identified application scenarios and specified them in a UML use case diagram. Figure 18 shows an excerpt of this use case diagram. Based on these application scenarios, we identified the following feature set: *Visualize Instrument Type*, *Add Instrument*, *Edit Instrument Type* and *Create Evaluation Format*. Figure 19 shows the *Add Instrument* feature, where one of its expected system behaviours is specified using a BDD notation.

BPB for EAMS-CBALM

The BDD scenario template is similar to an extended finite state machine [14] and was formally defined through Protégé OWL ontology in [62]. An excerpt of this ontology is shown in Fig. 20. This ontology describes concepts used by platforms, models and artefacts of interactive systems, allowing User Interface (UI) elements and its behaviours to be described in order to specify scenarios and to support

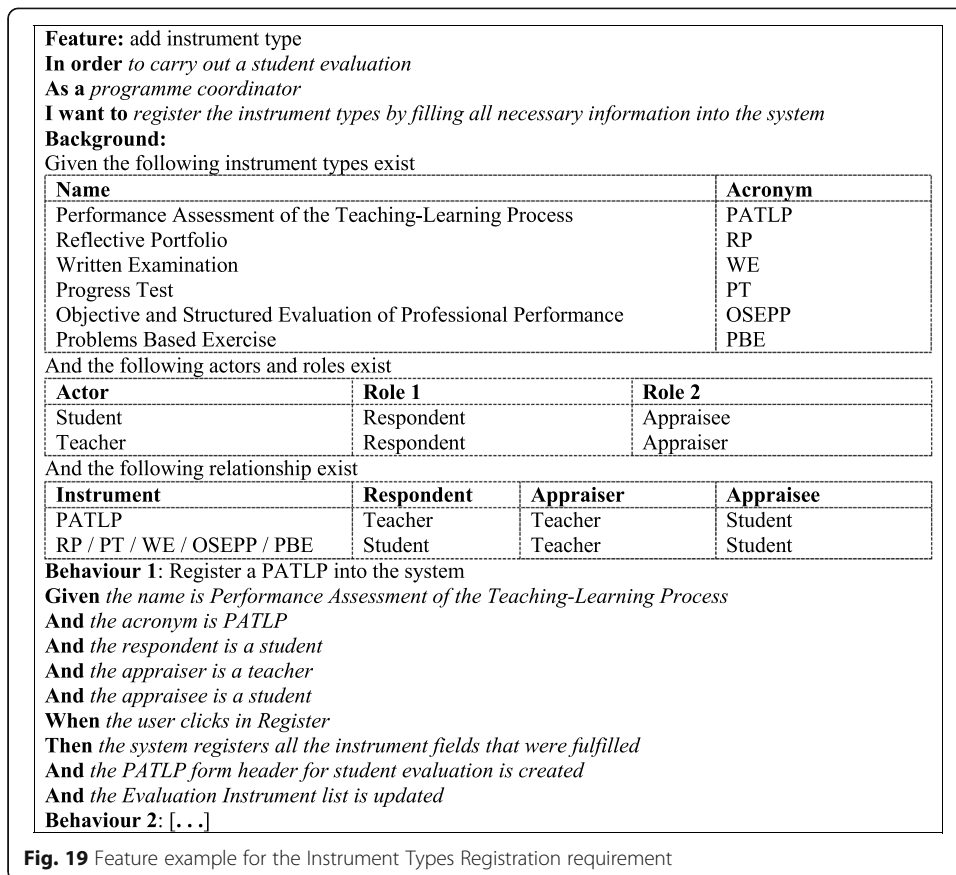


acceptance testing. A scenario that runs in a given UI is represented as a transition, and states represent the original and resulting UIs after a transition occurs, respectively. Scenarios in the transition have at least one or more *conditions*, one or more *events*, and one or more *actions*, which are represented by the *given*, *when* and *then* clauses, respectively. In order to formally specify the user stories of the PB using ontologies and BDD templates, a User Story and Scenario ontology should be defined using a similar state-based model.

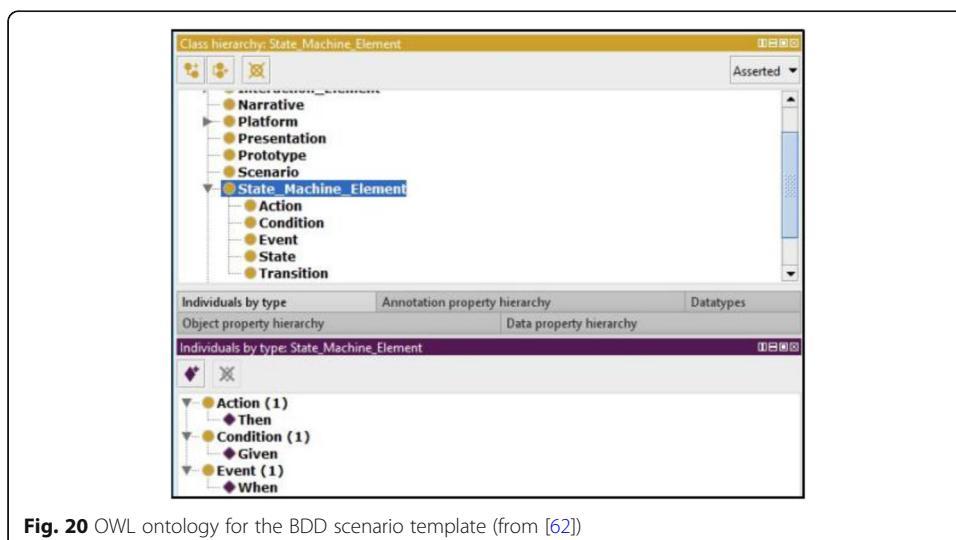
Although BDD user stories and scenarios follow the templates described in [45] and illustrated in Fig. 3, BDD tools generally do not strictly follow these models. For example, JBehave [28] supports a slightly different user story template and the same scenario template, which are shown in Fig. 21.

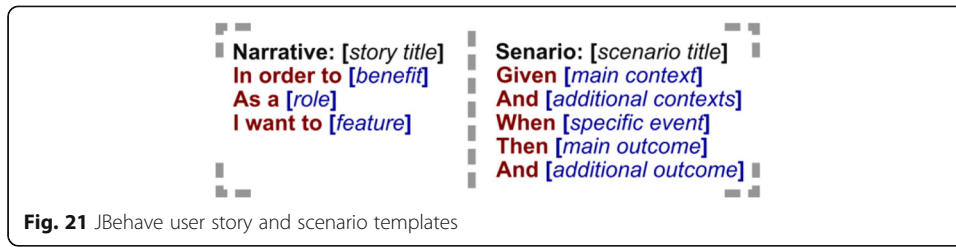
Six scenarios were defined in our user story example, one for each instrument type. Figure 22 illustrates the state-based model we defined for the EAMS-CBALM User Story and Scenario ontology. This example deals with the registering of the PATLP instrument type.

The ontology presented in [62] is domain-independent, since it allows behaviours to be described in terms of actions on UIs through interaction elements in a scenario that is not bound to a specific domain. Therefore, the specific business behaviours of our user story example still had to be specified, since most of them



employ the domain terminology derived from the Evaluation Process ontology. Hence, we had to map terms from this ontology to a user interaction (e.g., click, selection), and to define steps for each one of these interactions. Figure 23 shows an excerpt of the EAMS-CBALM User Story and Scenario ontology that deals with the registration of the PATLP instrument type.





DSB for EAMS-CBALM

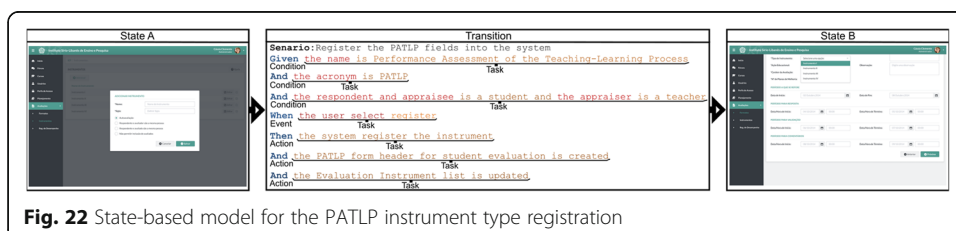
BDD prescribes that acceptance tests have to be defined before the selected BDD User Story and Scenario (BDDUSS) is implemented. Since it is possible to reuse the steps of the User Story and Scenario ontology in multiple testing scenarios, in the DSB activity scenarios for acceptance testing can be defined, which are applied later in the ES activity, mitigating in this way the misinterpretation of feature sets in this activity. However, the user stories and scenarios described using the ontology language (OL) first have to be translated to the language supported by the BDD tools (BDDT). Depending on the used ontology tools (OT) and BDDT, tools that support to this mapping may be available. In this application example, we defined a script that reads the OWL file that describes the User Story and Scenario ontology, and selects all BDD scenarios and its corresponding steps. Then, a JBehave textual story file is created and filled in with the appropriate information using the JBehave scenario template. This textual story file has been used later to structure all acceptance tests related to the scenarios defined in this file. Figure 24 shows an excerpt of the User Story and Scenarios OWL file and its mapping to a JBehave textual story file.

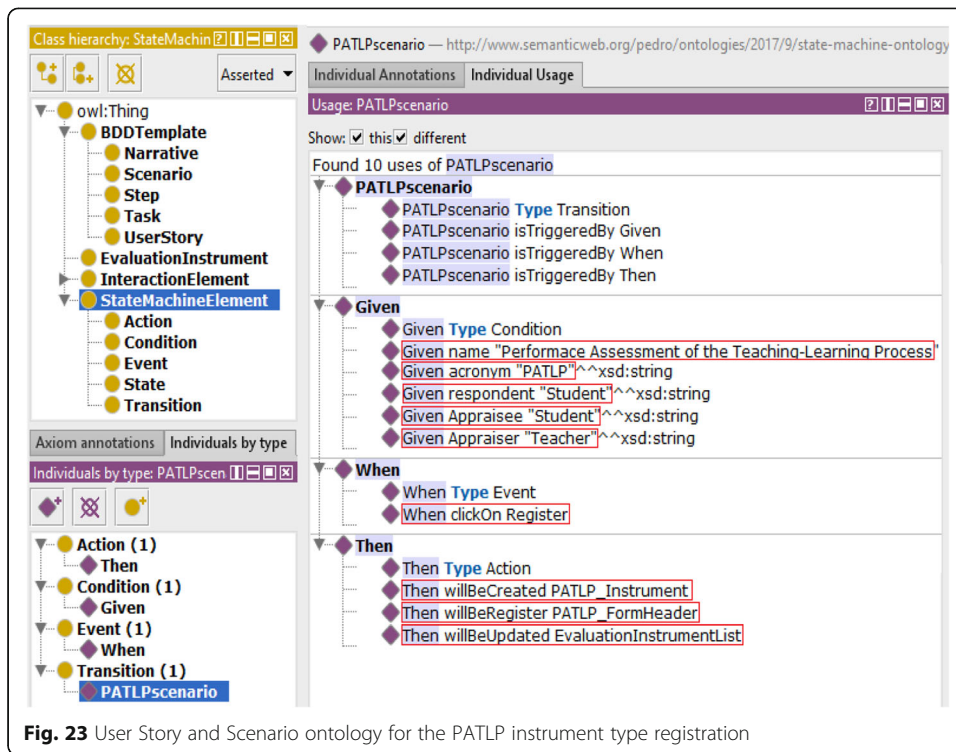
ES for the Selected BDDUSS of EAMS-CBALM

Two main tasks must be performed in this activity: (1) define BDD acceptance tests for the selected BDDUSS of EAMS-CBALM; and (2) implement the selected BDDUSS.

Acceptance tests for the selected BDDUSS

JBehave allows acceptance tests to be structured according to its textual story file. This file was created in the DSB activity and contains the main behaviours of the feature set to be tested. Additional scenarios may be inserted if the SE believes it is necessary to complement the acceptance test suite. There are six sets of acceptance tests in our user story example, one set for each instrument type to be registered, and the most relevant excerpts of the JBehave acceptance test code for the PATLP instrument type registration are shown in Fig. 25.





```
Public class InstrumentRegisterSteps extends Steps{
...
// setting variables and methods to support the scenarios
@Given("the name is Performance Assessment of the Teaching-Learning Process ")
public void setInstrumentName(String instrumentName){
... //set and check instrument name
}@Given("the acronym is PATLP ")
public void setInstrumentAcronym(String acronym){
... //set and check acronym role
}@Given("the respondent is student")
public void setRespondent(String respondent){
... //set and check respondent role
}@Given("the appraisee is student")
public void setAppraisee(String appraisee){
... //set and check appraisee role
}@Given("the appraiser is teacher")
public void setAppraiser(String appraiser){
... //set and check appraiser role
}@When("clickOn Register")
public void addInstrument(Button InstrumentRegisterAction) {
... //perform action
}@Then("willBeCreated PATLP_Instrument")
public void saveInstrument() {
... //data persistence
}@Then("willBeCreated PATLP_FormHeader")
public void createInstrumentFormatPHeader() {
... //creating, persisting format header for the instrument
}@Then("loads the people performance evaluation format header page")
public void updateInstrumentList() {
... //display instrument list updated
}
// next scenario
...
}
```

Fig. 25 Excerpt of the acceptance tests generated for the PATLP scenario

JBehave uses the *@Given*, *@When* and *@Then* annotations to relate scenario specification clauses to Java methods, and the Java class that implements these methods should extend the Steps class. JBehave allows the scenario to be executed as a JUnit test [29]. The link between the JBehave executor framework and the textual scenarios is provided by the Embeddable class definitions. This class extends class JUnitStory, and its name can be mapped to the textual story filename.

Implementation of the selected BDDUSS of EAMS-CBALM

This task corresponds to the second phase of ES activity, in which the BDD acceptance tests guides the implementation. After the implementation passes the acceptance tests of the selected BDDUSS, we can assert that all requirements have been fulfilled. A readable behaviour-oriented code can be obtained since the ubiquitous language terminology of the application has been used in all ScrumOntoBDD activities. Figure 26 depicts this with excerpts of the JBehave implementation code for the Instrument Types Registration requirement. In this code, the names of the class and methods employ the EAMS-CBALM ubiquitous language terminology.

```

1 package br.com.ufscar.dc.ges.pemaap
2
3 class Instrument {
4
5   String name
6   String acronym
7   Boolean dontHaveAppraisee
8   Boolean selfAppraisal
9   Boolean sameAppraiserAndRespondent
10  Boolean sameAppraiseeAndRespondent
11
12  static hasMany = {formats: Format}
13
14  static constraints = {
15    name blank: false, nullable: false
16    acronym blank: false, nullable: false
17    dontHaveAppraisee blank: false, nullable: false
18    selfAppraisal blank: false, nullable: false
19    sameAppraiserAndRespondent blank: false, nullable: false
20    sameAppraiseeAndRespondent blank: false, nullable: false
21
22    formats blank: true, nullable: true
23  }
24
25  static mapping = {
26    sort name: 'name'
27  }
28 }
29
30 class InstrumentService {
31
32  def messageSource
33
34  def create(data) {
35    Instrument instrument = new Instrument(
36      name: data.name,
37      acronym: data.acronym,
38      dontHaveAppraisee: data.dontHaveAppraisee,
39      selfAppraisal: data.selfAppraisal,
40      sameAppraiserAndRespondent: data.sameAppraiserAndRespondent,
41      sameAppraiseeAndRespondent: data.sameAppraiseeAndRespondent)
42    return instrument
43  }
44
45  def update(data) {
46    Instrument instrument = Instrument.findById(data.id as Long)
47
48    if (instrument) {
49      instrument.name = data.name
50      instrument.acronym = data.acronym
51      instrument.dontHaveAppraisee = data.dontHaveAppraisee
52      instrument.selfAppraisal = data.selfAppraisal
53      instrument.sameAppraiserAndRespondent = data.sameAppraiserAndRespondent
54      instrument.sameAppraiseeAndRespondent = data.sameAppraiseeAndRespondent
55    }
56
57    return instrument
58  }
59 }

```

Fig. 26 Excerpt of the Instrument Types Registration requirement implementation

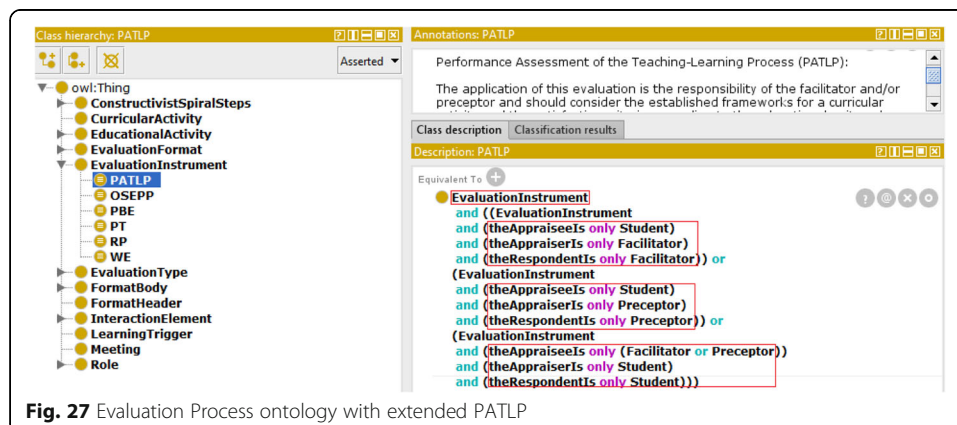
The complete JBehave implementation code corresponds to the PI that implements the *Instrument Types Registration* requirement. Nevertheless, this code must be inspected by the DT before being delivered to the client.

SRR for EAMS-CBALM

An SRR activity that led to a PB adaptation was related to the *Instrument Types Registration* requirement. During the DT meeting of this SRR activity, the PO reported:

“In addition to the student evaluation, the UFSCar Medicine Programme also contemplates the teacher evaluation, mainly when the latter is playing the facilitator or preceptor role in a curricular activity. In this case, similar instruments to those employed to evaluate students are also employed to evaluate teachers. For example, the PATLP instrument is also employed for teacher evaluation, but with different relationships between the involved actors: the respondent and appraiser is the student, and the appraiser is the teacher.”

Based on this report, the Evaluation Process ontology illustrated in Fig. 16 was extended to include teacher evaluation. The PB item related to the *Instrument Types Registration* requirement then was adapted for encompassing both student and teacher evaluations. Figure 27 shows an excerpt of the Evaluation Process ontology with the



extended PATLP instrument type representing student, facilitator and preceptor evaluations.

Evaluation

In our evaluation of ScrumOntoBDD, we aimed at confirming the two hypotheses posed in the Introduction section, concerning communication improvement and reduction of ambiguities, respectively. In this evaluation, we applied Action Research [27], which is a form of empirical research targeted to practice improvement. This section justifies our choice of using Action Research (AR) and describes the experimental analysis we performed to confirm our hypotheses, demonstrating in this way the benefits of ScrumOntoBDD for practice.

Action research

Haneef [23] points out that research can be theoretical (conceptual) or practical, and two major categories of practical research are empirical research and developmental or problem-solving research. The former is evidence-oriented and focuses on data collection, where the researcher mainly tries to describe and theorise about some phenomena rather than develop or prescribe solutions to a problem, while the latter is utility-oriented or solution-oriented, where the researcher focuses on the development or prescription of artefacts that should become an integral part of the acquired knowledge. Design Research (or Design Science) and Action Research (AR) [27] are two representatives of developmental (problem-solving) research, and they are both suitable for evaluating a development approach like ScrumOntoBDD. Besides that, we have decided to apply AR in this evaluation because it stresses practice improvement and the participation of practitioners in the experimentation.

According to [32], in AR the work is partitioned into stages and can involve several cycles. Figure 28 shows the canonical AR cyclic process [27], which consists of five phases:

1. *Diagnosing*, in which a problem is identified or defined;

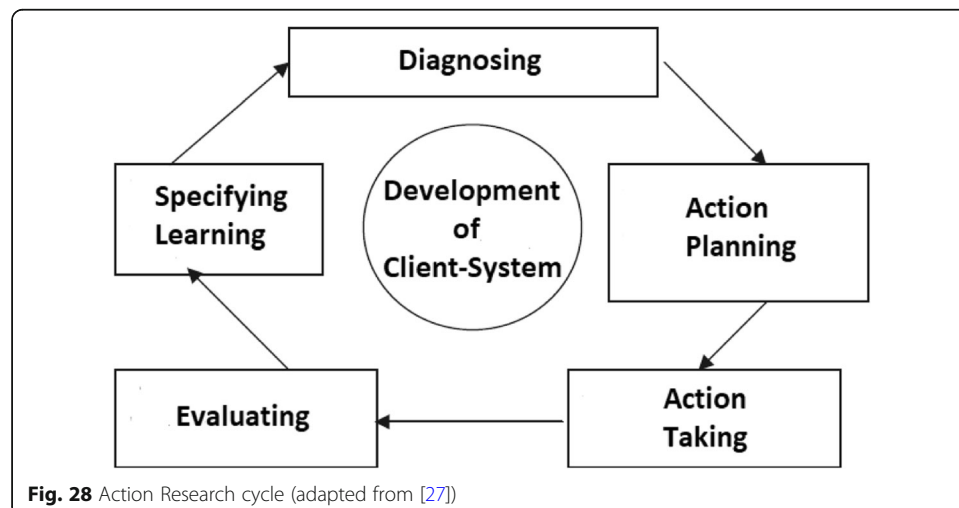


Fig. 28 Action Research cycle (adapted from [27])

2. *Action Planning*, in which alternative courses of action to solve a problem are considered;
3. *Action Taking*, in which a course of action is selected;
4. *Evaluating*, in which consequences of an action are studied;
5. *Specifying Learning*, in which general findings are identified.

Although [27] acknowledges that AR projects may differ in the number of phases that is carried out for solving a problem, in our evaluation, we applied a cyclic approach that follows the canonical AR cyclic process shown in Fig. 28. Furthermore, it is quite common in AR to perform the AR cycle multiple times until a problem solution is reached. In our validation, we used AR to carry out an experimental analysis of ScrumOntoBDD because the AR cyclic process is guided by the collaborative and continuous learning principles and requires a cooperative and participative involvement of the researcher(s) and the actors in the evaluation experiment.

Objective, setting and participants

The main objective of this experiment has been to evaluate if ScrumOntoBDD both improves communication between PO and developers (hypothesis H_1) and reduces the ambiguities intrinsic in using natural languages to report user stories (hypothesis H_2). For this experiment, we choose a usage scenario of the UFSCar Medicine Programme to be supported by EAM-CBALM that is similar to the scenario used to illustrate ScrumOntoBDD in the Application example section, so that we could reuse most of the ontologies already defined and validated. This usage scenario is supported by the EAMS-CBALM Evaluation Management module and is related to a more comprehensive functional requirement, namely the *Create People Performance Evaluation Format* requirement.

The following people participated in this experiment:

- *Participant 1*: PO involved in the EAMS-CBALM development, who also played the DE role since she was a teacher of the UFSCar Medicine Programme;
- *Participant 2*: PhD student involved in the EAMS-CBALM development, who played the SE role;
- *Participant 3*: MSc student involved in the EAMS-CBALM development as an observer, who is the first author of this paper and was the AR researcher who played all the other roles defined in ScrumOntoBDD.

Besides these participants, an AR expert contributed to the preparation and analysis of a semi-structured interview, which was employed in the Evaluating and Specifying Learning phases of the AR cycles. All these participants signed *The Written Informed Consent Form*.

Experimental analysis

Since we wanted to compare the development of a same usage scenario with two different approaches (using Scrum and using ScrumOntoBDD), our experimental analysis

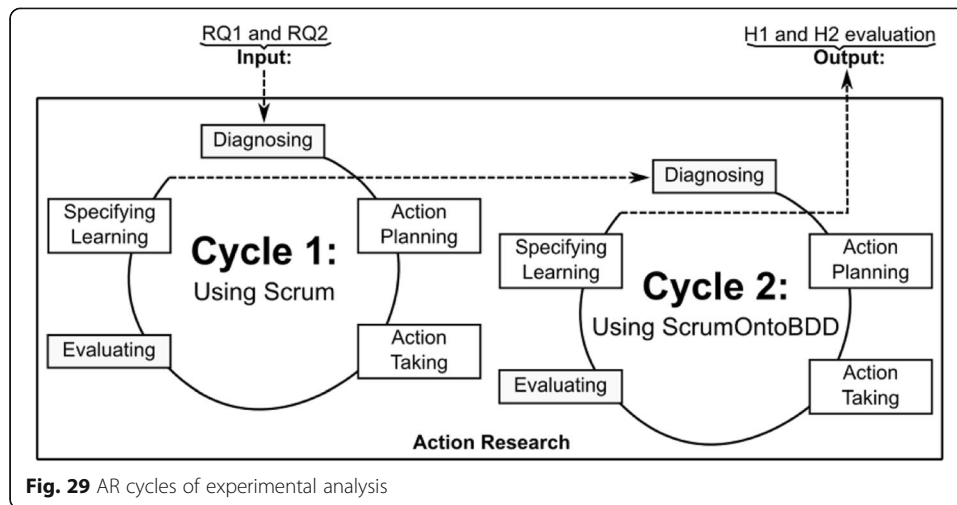


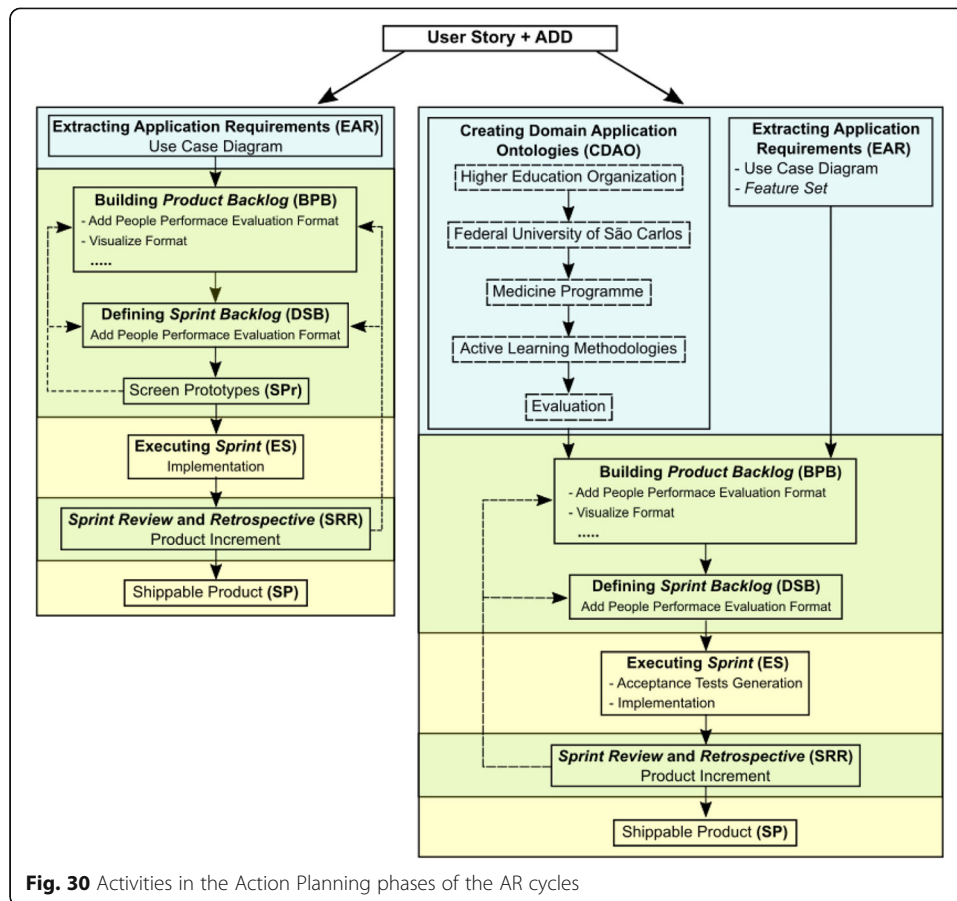
Fig. 29 AR cycles of experimental analysis

consisted of two AR cycles: a first AR cycle using Scrum, and a second AR cycle using ScrumOntoBDD. This is similar to the approach reported in [39].

Our research questions RQ_1 and RQ_2 have been directly derived from the problems we faced in the EAMS-CBALM development, which have been discussed in the Introduction section. In order to simplify the presentation of our experimental analysis approach, we refer to these problems through the acronyms RQ_1 and RQ_2 (their corresponding research questions), using them to denote the input of the experimental analysis, and consequently of the Diagnosing phase of the first AR cycle. The evaluation of hypotheses H_1 and H_2 is a direct consequence of the analysis of the data collected in the semi-structured interview. Therefore, we use 'H₁ and H₂ evaluation' to denote the output of the experimental analysis, and consequently of the Specifying Learning phase of the second AR cycle. Figure 29 illustrates these two AR cycles with their phases.

In order to have PO and SE-independent views of this experiment, we performed it twice: first involving *Participant 1* and *Participant 3*; and then involving *Participant 2* and *Participant 3*. The AR phases in each cycle were performed as follows:

- Diagnosing* defines the problem to be investigated by the experiment, which is represented by the research questions RQ_1 and RQ_2 for the first cycle;
- Action Planning* starts by choosing the EAMS-CBALM usage scenario and the Application Domain Documents (ADD) for this experiment, which are the same for both cycles, followed by planning Scrum activities for the first cycle, and ScrumOntoBDD activities for the second cycle;
- Action Taking*, for the first cycle, is the execution of the Scrum activities for achieving an implementation of the EAMS-CBALM usage scenario, which was performed by the software house, and for the second cycle is the execution of the ScrumOntoBDD activities for achieving another implementation of the same EAMS-CBALM usage scenario, which was performed by the AR researcher;



- d) *Evaluating* a semi-structured interview performed in both cycles and conducted by *Participant 3*, where the interviewee was *Participant 1* in the first experiment and was *Participant 2* in the second experiment; and
- e) *Specifying Learning* is the analysis of the data collected in the interview, which in the first cycle was a feed for the *Diagnosing* phase of second cycle, and in the second cycle was the output of the whole experiment, i.e., the evaluation of hypotheses H_1 and H_2 .

Action planning

The EAMS-CBALM usage scenario for the *Create People Performance Evaluation Format* requirement was extracted from a user story reported by the PO during the EAMS-CBALM development. Based on this user story and the ADD [73], the activities for both AR cycles were planned. Figure 30 gives an overview of this planning, and it should be highlighted that the *Screen Prototypes (SPr)* activity was included in Scrum by the software house.

Action Taking

Except for the SPr activity in cycle 1 and for the CDAO activity in cycle 2, all activities were performed both cycles, and the only difference between them is in the way they

performed. In the following paragraphs, we discuss how each of these activities was performed in each cycle.

The first task of the EAR activity was the same for both cycles, i.e., identification of the application scenarios for the *Create People Performance Evaluation Format* requirement by means of a UML use case diagram. The EAR activity of cycle 2 had a second task since the approach prescribes the use of BDD, namely the definition of a feature set for the *Create People Performance Evaluation Format* requirement based on the application scenarios described in the UML use case diagram.

Most of the ontologies created for illustrating ScrumOntoBDD have been reused, adapted or extended in the CDAO activity of cycle 2 of this experiment. For instance, the Evaluation Process ontology from the Evaluation Process ontology section was extended to support the description of the *Create People Performance Evaluation Format* requirement.

In the BPB activities of cycle 1, the PB items were informally specified using tables, while in cycle 2, they were formally specified using ontologies. SPr activity was included in cycle 1 because it was widely used by the software house throughout the EAMS-CBALM development, and was not included in cycle 2 because we wanted to assess its usefulness since we were considering to include it in ScrumOntoBDD.

The ultimate goal of ES activity in both cycles is to obtain an implementation of the selected PB item. However, in cycle 2, the Acceptance Tests Generation task prescribed by the use of BDD was performed before the actual Implementation task. Figure 31 (left side) shows an excerpt of the implementation obtained in cycle 1 by the software house during the EAMS-CBALM development, while on the right side shows an excerpt of

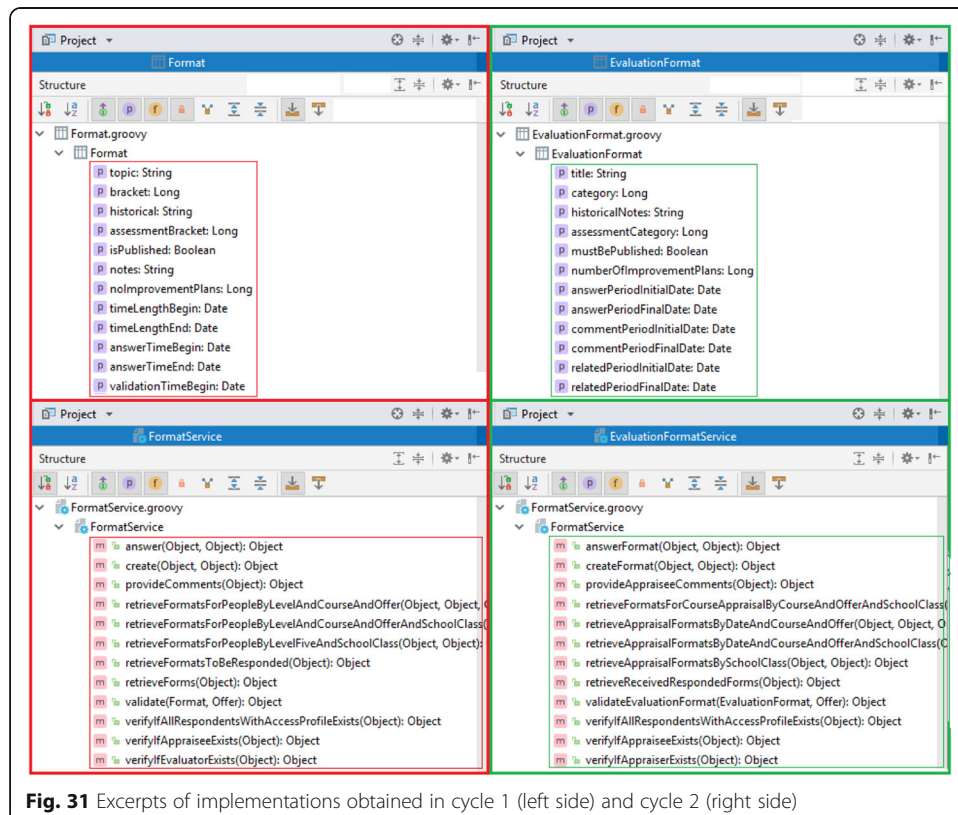


Fig. 31 Excerpts of implementations obtained in cycle 1 (left side) and cycle 2 (right side)

the implementation obtained in cycle 2 by the AR researcher. These implementations correspond to the PIs to be analysed in the SRR activities of these cycles. Although both implementations have similar structural code, they have different names for classes, attributes and methods. Since cycle 2 uses ScrumOntoBDD, its code terminology is closer to the application domain. Therefore, this implementation is expected to be easier to understand for the participants. Figure 31 shows implementation excerpts of the same PB item from both cycles, illustrating similar structural code. However, the excerpts from cycle 2 (right side) have the application domain terminology in the classes, attributes and methods names.

Evaluating

We applied semi-structured interviews, also known as flexibly structured interviews [6], to evaluate each cycle since these interviews allow the interviewee to more freely report on her experiences and more freely articulate her thoughts about them. These interviews follow a predefined script of subjects to be addressed with the interviewee by means of open questions. For example, to assess the interviewee's participation in the implementation of the Create People Performance Evaluation Format requirement, in the Evaluating phase of the first AR cycle, the following main question and sub-questions were asked: "What was your participation in the development of the Create People Performance Evaluation Format software requirement?"; "How would you describe the steps in which you participated more actively?"; and "Would you point out steps that were most critical in terms of defining system requirements? (If so, which ones?)". In total, 7 main questions and 15 sub-questions have been asked in each interview, but these questions unfolded into more sub-questions.

The interview with *Participant 1* (PO) was held on 6 August 2018 in two sections with a total duration of 1 h 45 min 50 s, and with *Participant 2* (SE) the day after in a single section of 1 h 45 min 14 s. These interviews were audio-taped with the participants' consent. Each interview consisted of two parts, corresponding to each AR cycle (cycles 1 and 2) of the experiment, respectively. In the first part (cycle 1), the AR researcher presented and illustrated the Scrum activities employed by the software house for developing this system with the implementation of the *Create People Performance Evaluation Format* requirement of EAMS-CBALM, which resulted in the product currently being tested in the UFSCar Medicine Programme. The interviewee then provided comments on the AR presentation, regarding her participation in the EAMS-CBALM development and the Scrum method employed in this development. Furthermore, the AR researcher encouraged the interviewee to provide additional remarks, clarifications, and criticism about these matters. In the second part (cycle 2), the AR researcher presented and illustrated the ScrumOntoBDD activities with the implementation of the same EAMS-CBALM requirement. Similarly, the interviewee commented on the presentation, and the AR researcher encouraged the interviewee to provide more comments, evaluate the ScrumOntoBDD activities and compare them with the activities in cycle 1, also taking into account the interviewee's professional experience.

These audio-taped interviews were fully transcribed in Portuguese using conventional writing. The interview transcripts were inductively reduced by seeking what emerges as important and of interest, then analysed and interpreted in the following sequential

Table 7 Main results of the cycle 1 Evaluating phase

Product Owner (PO)		Software Engineer (SE)	
Focus on the application development		Focus on the development methodology	
Advantages	Disadvantages	Advantages	Disadvantages
Agility due to weekly meetings	Lack of more systematic "records" that could give more visibility to the different development phases	Using agile software development methodologies like Scrum	Scrum distributes roles and activities to team members, but does not explain how to implement these activities Lack of a formal methodology that facilitates the communication with the user and with the developer
Using screen prototypes	Lack of development team experience	Using screen prototypes	Division into non-testable sub-tasks throughout the process Need for more frequent meetings during the requirements gathering phase Not testable with a real data set
Software that "handles the Medicine Programme needs"	Lack of openness for more effective collaboration, not taking advantage of PO meaningful prior experience Discontinuity and very delayed testing		Lack of development team experience Lack of openness for more effective collaboration, not taking advantage of SE meaningful prior experience Discontinuity and very delayed testing

Table 8 Main results of the cycle 2 Evaluating phase

Product Owner (PO)		Software Engineer (SE)	
Focus on the application development		Focus on the development methodology	
Advantages	Disadvantages	Advantages	Disadvantages
Phase for defining a common language between user and developer	Does not use screen prototypes Probably the usual POs will have understanding difficulties	Use of formal methodology to complement Scrum	Longer development process
Use of "records" that give visibility to the development phases and enable the PO collaboration on those phases	Need for an additional actor, the Ontology Engineer, and likely an increase of the system cost	Use of "records" that facilitate the communication with the user	Does not use screen prototypes Probably the usual POs will have understanding difficulties
Possibility of better result with less code repair	Likely increase in system development time	Better communication with the developer, and consequent increase in productivity Likely reduction of code repair Probably better cost/benefit ratio	

process [60]: seeking connections and repetitions among excerpts; noting them; labeling them by descriptive words or expressions; and grouping them by building interpretative categories. Finally, the selected excerpts were arranged according to their AR cycles and to the main results of this analysis. This entire process was performed by hand without employing any software tools. The complete English translation of each one of these excerpts, followed by its original transcription, are presented in [68].

Specifying Learning

We present our conclusions here based on the main results obtained from the analysis done in the Evaluating phases, which are summarized in Tables 7 and 8.

Tables 7 and 8 show that the PO comments and evaluations mainly focused on aspects and issues related to the development process of a product that fulfils the user demands. In contrast, the SE comments and evaluations mainly focused on aspects and issues related to the software development methodologies employed in cycles 1 and 2.

Table 7 shows that the difficulties identified in cycle 1 by both interviewees converge, although their focus was different. These results corroborate with our assumption that the exclusive use of Scrum method in cycle 1 would reveal communication issues between the PO and DT, insofar as information is lost in EAR and BPB activities. Firstly, the interaction between PO and DT was almost always in natural language and through texts, and all information extracted from the user stories are described exclusively using BPMT. Secondly, informally specified requirements were then mapped to PB items, which are also usually described informally or semi-formally. Table 7 also shows issues concerning long application development time and the lack of openness of the DT to collaborate with the PO and SE, although both had significant prior experience in their respective roles. Even though the interviewees agree that the main cause of significant delays in the PI releases was due to most tests being insufficient and noncontinuous throughout the software development, they pointed out other factors that should be taken into account. The PO emphasizes the complexity of the requirements related to the UFSCar Medicine Programme evaluation system as a factor to be considered, but for the SE, the adopted method and the DT lack of experience are also relevant factors to be considered.

From the results in Table 7 and the considerations above, we can conclude that in cycle 1, *there is enough evidence that the use of plain Scrum neither improves communication between POs and developers nor reduces ambiguities intrinsic in using natural languages to report user stories*. With the exception of the screen prototypes, which both interviewees pointed out to be an effective resource in the communication between PO and DT, all remaining comments confirmed the problems related to RQ₁ and RQ₂.

Table 8 shows that interviewees' opinions converge concerning the possibilities created by ScrumOntoBDD to overcome the problems identified in cycle 1. The ScrumOntoBDD characteristics that are considered to be the most advantageous are the ability to define a common language between users and developers by employing ontologies and BDD, the generation of artefacts ("records") that give visibility to each phase of software development (which can also improve the communication and PO collaboration with the DT); and reduction of the total time spent repairing software bugs. The

lack of screen prototypes was indicated as a drawback in ScrumOntoBDD, which is understandable since it was considered a major benefit in cycle 1 by both interviewees. Table 8 also shows that the interviewees converge also when it comes to the ScrumOntoBDD drawbacks. The first drawback concerns possible difficulties of less experienced or less available POs to understand the formalism adopted in this approach, while the second one concerns the additional time required to define the ontologies, and the higher cost of having an additional actor, i.e., the Ontology Engineer (OE). Nevertheless, both interviewees indicate the potential of ScrumOntoBDD to obtain better results in terms of cost–benefit. The SE insists on the potential significant reduction of time spent in the software development process as a whole, due to the smaller number of errors she believes would be made when using this approach.

From the results in Table 8 and the considerations above, we conclude that in cycle 2, *there are strong evidences that our research hypotheses H_1 and H_2 both hold for ScrumOntoBDD, i.e., that combining BDD with Scrum can improve communication between PO and developers, and employing ontologies can reduce the ambiguities intrinsic in using natural languages to report on user stories.*

Table 9 Systematic Literature Reviews summary

	Systematic Literature Review 1 (SLR ₁)	Systematic Literature Review 2 (SLR ₂)
Main question	How BDD or its ubiquitous language can be used to improve agile software development?	How ontologies can be combined with agile methods to improve software development?
Data sources	ACM Digital Library: http://portal.acm.org/ IEEEExplore: http://ieeexplore.ieee.org/ Scopus: https://www.scopus.com/ CAPES portal: http://www-periodicos-capes-gov-br.ez1.periodicos.capes.gov.br/	
Canonical search query	((BDD) OR (Behaviour-Driven Development) OR (ubiquitous language)) AND ((agile development) OR (Scrum) OR (eXtreme Programming))	((ontology) OR (ontologies) OR (taxonomy) OR (terminology)) AND ((agile development) OR (Scrum) OR (eXtreme Programming) OR (BDD) OR (Behaviour-Driven Development))
Inclusion criteria	Abstract refers to BDD or ubiquitous language in agile software development Title, keywords or abstract refers to a study to classify other papers or is a SLR on BDD or ubiquitous language in agile software development Title or keywords match the terms from the canonical search query Paper was published not longer than 6 years ago (from the SLR starting date)	Abstract refers to ontologies or taxonomies in agile software development Title, keywords or abstract refers to a study to classify other papers or is a SLR on ontologies in agile software development
Exclusion criteria	Paper is not about improving agile software development with BDD tools or ubiquitous language (primary studies) Paper lacks details on the tools and methods used or developed (primary studies) Repeated or duplicated studies. The most complete and comprehensive study has been considered (primary and secondary studies). Papers focuses on other domains than Education, Software Engineering and Health (primary and secondary studies).	Paper does not discuss ontologies and/or how they were developed (primary studies). Paper lacks a method for software development (primary studies).
Statistics	279 Studies found 83 Studies filtered with inclusion criteria 6 Studies selected with exclusion criteria	163 Studies found 79 Studies filtered with inclusion criteria 7 Studies selected with exclusion criteria

Related work

To perform both SLRs, we used the State of the Art through Systematic Review (StArt) tool, which supports the planning protocol steps proposed by [16]. The first SLR aimed at investigating how BDD and its ubiquitous language have been used to improve agile software development, particularly Scrum. The second SLR aimed at investigating how ontologies have been combined with agile software methodology such as BDD and Scrum in order to improve software development. Table 9 gives the main question, the data sources, the queries, inclusion and exclusion criteria and the statistics of the results for each SLR.

BDD in software development

The rising popularity of BDD has spurred research to apply it in different domains [13]. Since currently available BDD tools give little or no support to the planning phase, most of the BDD-related developments we found in our SLR define a specific ubiquitous language for a given application domain.

Rocha [52] reports on an experiment in which BDD and TDD have been applied to Software Engineering teaching. This experiment was carried out during the Software Engineering Laboratory course of a Bachelor programme in Computer Science and Information Systems at a Brazilian university. Based on the results of this experiment, the authors argue that BDD helps in the integration of different contents of this course, enabling a better understanding of the problems to be solved and bringing benefits for student learning.

Lubke [34] presents a platform for integrating systems responsible for land registration in Switzerland. The goal is to reduce process execution time between systems and also the communication time between POs. In order to model executable integration processes between various systems and to develop test cases to validate these processes, the authors used BDD and Business Process Model Notation (BPMN) as the ubiquitous language for defining test case models (scenarios). BPMN was chosen because this language was known by the POs. By combining BDD with BPMN scenarios, the authors found improvements in communication between developers, users and investors, which contributed significantly to the more agile development of the platform.

Oruç [48] proposes a tool to facilitate the creation of scenarios for testing web services. This tool uses BDD in conjunction with the ubiquitous language Gherkin [78] to dynamically generate test scripts. These scripts are run in JMeter, which is a test tool to analyse and measure the performance of web applications [30]. The authors claim to achieve two benefits with this tool: because Gherkin is a domain-specific language, it allows any domain expert to create and run web service tests even without software knowledge; and developers do not need to write unit tests manually, since JMeter automates the execution of the automatically generated tests.

Silva [61] proposes an approach based on BDD to support the automated assessment of artefacts along the development process of interactive systems. A formal ontology model is defined for describing concepts used by platforms, models and artefacts that compose the design of interactive systems, allowing in this way a wide description of UI elements and their behaviour to support testing activities. In addition, the approach proposes improvements to how teams should write requirements for testing purposes. Once described

in the ontology, the behaviours can be reused freely to write new scenarios in natural language, providing test automation in BDD and decreasing manual coding.

Soeken [64] presents a methodology to assist developers carrying out the BDD steps. This methodology proposes a design flow where the developer engages in a dialog with a computer program in an interactive way. This dialog contains the user story, and this program processes each spoken sentence and generates the step definitions and code blocks (classes, attributes and methods) of each user story scenario. Some natural language processing tools are explored, and a case study illustrates the application. Rather than going manually through the established BDD steps, this methodology suggests some scenario skeletons to facilitate test refinement and implementation.

The main difference between our work and the work mentioned above is that our work not only applies BDD to software development, but also exposes the benefits of using BDD in combination with Scrum. Moreover, our case studies were developed for the education domain, more specifically for courses based on active learning methodologies. In our SLR, we have not been able to find this specific combination of techniques (BDD and Scrum) in software development in the education domain.

Ontologies in software development

Ontologies have been used in Computer Science and Software Engineering. Most of the work related to ontologies we found in our SLR propose a process or an approach to combine ontologies with some agile software methodology, in order to improve software development.

Machado [36] proposes an agile process that associates practices of Software Engineering, Ontology Engineering and Scrum to improve the collaboration between software and ontology engineers. This process provides a set of guidelines for defining activities, tasks, roles and artefacts to develop ontology-based software. OntoSoft was applied for developing an ontology-based application to map and recommend real estates. This paper synthesizes the main results obtained in the development of a PhD project, and its complete description can be found in [37].

Lin [33] presents an approach to help team members perform more efficiently their daily tasks according to a specific process. This approach is based on the K-CRIO ontology for business processes modelling and on a multi-agent system for providing intelligent assistance to workers.

Lucassen [35] proposes the Quality User Story (QUS) framework for ensuring the quality of agile requirements expressed as user stories. QUS contains 13 criteria that determine the quality of user stories in terms of syntax, semantics and pragmatics. Based on QUS, the Automatic Quality User Story Artisan (AQUSA) tool was built to detect QUS quality criteria violations and to improve user stories.

Silva [62] introduces an ontological model to support scenario description and to test functional requirements of interactive systems. This model was developed based on BDD principles, describing user behaviours when interacting with UI elements in a scenario-based approach. Once described in the ontology, behaviours can be freely reused to write new scenarios in natural language, providing test automation. A case study is presented for the flight tickets e-commerce domain, where ontology-based tools were used to support the assessment of evolutionary prototypes and final UIs.

The main difference between the work discussed above and ours is that we use ontologies in a broader context, starting from a reference ontology for a given domain, and gradually specialising this ontology so that it can be used in the agile software development for that domain. The other developments concentrate on the collaboration among software engineers and ontology engineers, modelling the Scrum development process, and/or extracting semantic information to improve user stories and to automate testing.

Finally, our work proposes an integrated approach that combines three techniques, namely Scrum, ontologies and BDD, to improve agile software development. To the best of our knowledge, this combination has not been investigated before.

Conclusions

This section presents our concluding remarks, discussing the main contributions and limitations of our work and giving directions for future work.

Contributions and limitations

The main motivation for doing this research came from the problems we observed during the EAMS-CBALM development using Scrum, namely that quite often it was necessary to redefine some system behaviour scenarios and their corresponding PB items due to misunderstanding of the stories reported by the PO, and that the definition of test suites was cumbersome, resulting in test suites that were incomplete or did not comply with the system requirements. These problems triggered two research questions, which we answered in the MSc project of the first author, namely (RQ₁) “How can the communication between POs and developers be improved?” and (RQ₂) “How can the ambiguities intrinsic to using natural languages to report user stories be reduced?”.

The main contribution of our work was the development of the ScrumOntoBDD approach, which combines Scrum, Ontology and BDD to address the problems we identified and consequently to answer our research questions. Based on these research questions and using our approach, we have defined two main hypotheses: combining BDD with Scrum can improve communication between PO and developers (H₁); and employing ontologies can reduce the ambiguities intrinsic in using natural languages to report user stories (H₂). An experimental analysis of ScrumOntoBDD was carried out using Action Research, in order to get evidence that the two hypotheses hold when employing this approach.

We verified H₁ with a case study in the EAMS-CBALM context, in which a ubiquitous language for the Education domain was defined together with BDD scenarios and acceptance tests, allowing the PO to follow and properly communicate with the developers throughout the development process. This work was reported in [66]. We verified H₂ with another case study also in the EAMS-CBALM context, in which domain ontologies were used to describe the UFSCar Medicine Programme, in order to reduce the ambiguities caused by using a natural language as a ubiquitous language. This work was reported in [67].

The main challenge we found during this work concerns the ScrumOntoBDD validation. Our hypotheses deal with the communication between POs and developers, and we based our experiments on the development of EAMS-CBALM, which is a product

that has been developed by a software house. Unfortunately, we could not obtain any statistical evidence for corroborating these hypotheses, due to the limited number of participants in these experiments. This also forced the AR researcher to play several ScrumOntoBDD roles on those experiments, which can be considered to be another threat to the validity of the ScrumOntoBDD evaluation results. Therefore, for a more systematic verification of our hypotheses, it would be necessary to define an experiment involving the complete EAMS-CBALM development team, but that was not possible because of their limited availability and the costs involved. This means that our ScrumOntoBDD evaluation aimed at finding evidence that our hypotheses hold when employing this approach with a limited number of participants and with a single requirement, and it is not as conclusive as it could have been if the whole development team could participate and more requirements could be implemented.

In ScrumOntoBDD, an ontology is defined to serve as a ubiquitous language for the whole project, but more specifically for the User Stories and Scenarios. In the application reported in this paper, we used HERO as domain ontology, but when applying ScrumOntoBDD in another application domain, an appropriate ontology should be found or developed in activity CDAO (see Fig. 8). This means that ScrumOntoBDD is general enough to be applied to other domains. For example, [53] describes a system for monitoring patients with Non-Communicable Diseases (NCD), such as hypertension and diabetes, which was developed according to a bottom-up approach and employing TDD. A similar system could be developed employing ScrumOntoBDD, a top-down approach, starting from a reference ontology for the Health domain such as the Disease Ontology [58] available at <http://www.disease-ontology.org>.

Future work

Firstly, we should consider extending the ScrumOntoBDD approach with the suggestions of the experiment participants. For example, we can consider including the Screen Prototype Generation task related to the SelectedBDDUSS item of the Backlog in the Defining Sprint Backlog (DSB) activity, so that these screen prototypes can be validated by the PO. If needed, this item could be adapted before being used as input in the next activity, i.e., Executing Sprint (ES).

Secondly, a quantitative study of the ScrumOntoBDD approach could be performed using an experimental methodology as the one presented in [77]. This study could involve two development teams that would develop the same application separately in two rounds: in the first round, a team would employ Scrum, and the other would employ ScrumOntoBDD; and in the second round, these approaches would be swapped. Metrics like time spent on each development of the application and/or lines of code could be then collected for further analysis and comparison. In addition, a qualitative study involving the same development teams could be carried out to assess the users' acceptance of the ScrumOntoBDD approach. In this case, assessment models such as the Technology Acceptance Model (TAM) [12] and the Self-Assessment Manikin (SAM) [25] can be used.

Finally, our description of the ScrumOntoBDD approach could be refined in order to transform it into a process, by defining the activities and involved tasks in more detail, such that they provide more concrete guidelines for the development teams. This work

would have to be performed in close collaboration with a development team in order to yield results that are really useful for practitioners.

Abbreviations

ADD: Application Domain Documents; AR: Action Research; AS: Application Scenarios; ATDD: Acceptance Test Driven Development; BDD: Behaviour-Driven Development; BDDT: BDD Tools; BDDUSS: BDD User Stories and Scenarios; BPB: Building Product Backlog; BPMN: Business Process Model Notation; BPMT: Business Process Modelling Techniques; CBALM: Courses Based on Active Learning Methodologies; CDAO: Creating Domain Application Ontologies; DE: Domain Expert; DMed: Department of Medicine; DRO: Domain Reference Ontologies; DSB: Defining Sprint Backlog; DT: Development Team; EAMS-CBALM: Educational and Academic Management System for Courses Based on Active Learning Methodologies; EAR: Extracting Application Requirements; ES: Executing Sprint; FS: Feature Sets; H1: Hypothesis 1; H2: Hypothesis 2; HERO: Higher Education Reference Ontology; IS: Information Systems; NCD: Non-Communicable Diseases; OE: Ontology Engineer; OL: Ontology Languages; OSEPP: Objective and Structured Evaluation of Professional Performance; OT: Ontology Tools; OWL: Web Ontology Language; PATLP: Performance Assessment of the Teaching-Learning Process; PB: Product Backlog; PBE: Problems-Based Exercise; PI: Product Increment; PL: Programming Languages; PO: Product Owner; PT: Progress Test; PTO: Programming Tools; RDF: Resource Description Framework; RDFS: RDF Schema; RP: Reflective Portfolio; RQ1: Research Question 1; RQ2: Research Question 2; SADT: Structured Analysis and Design Technique; SAM: Self-Assessment Manikin; SB: Sprint Backlog; SBe: System Behaviours; ScrumOntoBDD: Approach based on Scrum, Ontology and BDD for agile software development; SE: Software Engineer; SLH: S rio-Liban s Hospital; SLR: Systematic Literature Review; SM: Scrum Master; SP: Shippable Product; SPR: Screen Prototypes; SRA: Sprint Records and Annotations; SRR: Sprint Review and Retrospective; ST: Scrum Team; TAM: Technology Acceptance Model; TDD: Test-Driven Development; TRI: Teaching and Research Institute; UCG: Ubiquitous Computing Group; UFSCar: Federal University of S o Carlos; UI: User Interface; ULT: Ubiquitous Language Terminology; UML: Unified Modelling Language; USNL: User Stories in Natural Language; WE: Written Examination

Acknowledgments

This study was financed in part by the Coordena o de Aperfei amento de Pessoal de N vel Superior—Brasil (CAPE S)—Finance Code 001.

Authors' contributions

PLS did all SLRs to support this work development. Furthermore, PLS designed, developed and implemented the ScrumOntoBDD approach, all ontologies involved the approach and the evaluation model. Finally, PLS participated on the preparation, creation and presentation of the published work. WLS supervised and aided on the SLRs, the design, development and implementation of the published work. WLS also participated on the preparation, creation and presentation of the published work. LFP aided on the design of the published work. LFP also supervised and revised the preparation, creation and presentation of the published work. The authors read and approved the final manuscript.

Funding

The authors disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Coordena o de Aperfei amento de Pessoal de N vel Superior—Brasil (CAPE S)—Finance Code 001.

Availability of data and materials

Not applicable

Declarations

Competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Author details

¹Department of Computing, Federal University of S o Carlos, S o Carlos, SP, Brazil. ²Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, The Netherlands.

Received: 25 November 2020 Accepted: 22 April 2021

Published online: 13 June 2021

References

- Alatrish E (2013) Comparison some of ontology editors. *Int Sci J Manage Inform Syst* 8(2):18–24
- Beck K et al (2001) Manifesto for agile software development. www.agilemanifesto.org/. Accessed 5 Sept 2017
- Beck K (2002) *Test Driven Development: by example*. Addison-Wesley, 240 pgs
- Beck K (2012) *Extreme programming explained: embrace change*. Second edition, Addison-Wesley, 189 pgs
- Berners-Lee T (2009) The Semantic Web as a language of logic. Available at <https://www.w3.org/DesignIssues/Logic.html#Crawf90>. Accessed 14 May 2018
- Bogdan RC, Biklen SK (2007) *Qualitative research for education: an introduction to theory and methods*. Fifth edition, Pearson Education Inc, USA
- Carlisle C, Calman L, Ibbotson T (2009) Practice-based learning: the role of practice education facilitators in supporting mentors. *Nurs Educ Today* 29(7):715–721

8. Chen GD, Chang CK, Wang CY (2008) Ubiquitous learning website: scaffold learners by mobile devices with information-aware techniques. *Comput Educ* 50(1):77–90
9. Cockburn A (2004) *Crystal Clear: a human-powered methodology for small teams*. Addison-Wesley, p. 336
10. Concordion (2015). Available at <https://concordion.org/>. Accessed 26 Jan 2021
11. Cucumber (2014) Cucumber. Available at <http://cukes.info/>. Accessed 10 Oct 2016
12. Davis FD (1989) Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q* 13(3):318–341
13. Diepenbeek M et al (2015) Behavior Driven Development for tests and verification. *Tests and Proofs, Lectures Notes in Computer Science (LNCS)*, Springer International Publishing AG, USA, Vol. 8570, pp. 61–77
14. El-Fakih K et al (2016) Distinguishing extended finite state machine configurations using predicate abstraction. *J Softw Eng Res Dev* 4(1):26
15. Evans E (2003) *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, USA, p. 529
16. Fabbri S et al (2016) Improvements in the Start tool to better support the systematic review process. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE'16)*, Association for Computing Machinery, USA, paper 21, p. 05
17. Forte M et al (2013) A ubiquitous reflective E-portfolio architecture. *Int J Med Inform* 82(11):1111–1122
18. Gangemi A et al (2006) Modelling ontology evaluation and validation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 4011, pp. 140–154
19. Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int J Hum Comput Stud* 43(Issues 5-6):907–928 Available at <http://tomgruber.org/writing/onto-design.pdf>. Accessed 20 Apr 2018
20. Gruber TR (2008) *Ontology*. *Encyclopedia of Database Systems*, Springer-Verlag, p. 04. Available <http://tomgruber.org/writing/ontology-in-encyclopedia-of-dbs.pdf>. Accessed 20 Apr 2018
21. Guarino N (1998) Formal ontology and information systems. In *Proceedings of 1st International Conference on Formal Ontology in Information Systems (FOIS'98)*, N. Guarino (ed.), *Frontiers in Artificial Intelligence and Applications*, IOS Press, Vol. 46, pp. 3–15
22. Gutierrez-Pulido JR et al (2006) *Ontology languages for the semantic web: a never completely updated review*. *Knowledge-Based Syst* 19(7):489–497
23. Haneef N (2011) Empirical research consolidation: a generic overview and a classification scheme for methods. *Qual Quantity Int J Method* 47(1):383–410
24. Harith A, Kieron O, Nigel S (2005) Common features of killer apps: a comparison with Protégé. In *Proceedings of 8th International Protégé Conference*, p. 04. Available at <https://eprints.soton.ac.uk/260989/1/protége05-Alani.pdf>. Accessed 29 May 2018
25. Hayashi ECS et al (2008) "Avaliando a Qualidade Afetiva de Sistemas Computacionais Interativos no Cenário Brasileiro". *Resultados do Workshop Usabilidade, Acessibilidade e Inteligibilidade Aplicadas em Interfaces para Analfabetos, Idosos e Pessoas com Deficiência Usabilidade (in Portuguese)*, VIII Brazilian Symposium on Human Factors in Computing Systems (IHC 2008), Brazil, pp. 55–62
26. Highsmith JA (2000) *Adaptive software development: a collaborative approach to managing complex systems*. Dorset House Publishing, USA, p. 358
27. Järvinen P (2007) Action research is similar to design science. *Qual Quantity* 41(1):37–54
28. JBehave (2015) JBehave. Available at <http://jbehave.org/>. Accessed 19 Oct 2016
29. JUnit (2016) JUnit. Available at <http://junit.org/junit4/>. Accessed 14 Oct 2016
30. JMeter (2016) JMeter graphical server performance testing tool. Available at <http://jmeterapache.org/>. Accessed 19 Dec 2016
31. Koskela L (2008) *Test Driven: TDD and acceptance TDD for Java Developers*. Manning Publications Co., p. 513
32. Lewin K (1988) *Group decision and social change*. *The Action Research Reader*, S. Kemmis (ed.), Deakin University Press, Australia, pp. 47–56
33. Lin Y et al (2015) Multi-agent system for intelligent Scrum project management. *Integrated Comput Aided Eng* 22(3):281–296
34. Lubke D, Van Lessen T (2016) Modelling test cases in bpmn for behavior-driven development. *IEEE Softw* 33(5):15–21
35. Lucassen G et al (2016) Improving agile requirements: the Quality User Story framework and tool. *Requirements Eng* 21(3):283–403
36. Machado JB et al (2016) *OntoSoft Process: towards an agile process for ontology-based software*. In: *Proceedings of 49th Hawaii International Conference on System Sciences*, IEEE Computer Society, pp 5813–5822
37. Machado JB (2017) "OntoSoft: um processo de desenvolvimento ágil para software baseado em ontologia". PhD's thesis (in Portuguese). Graduate Program in Computer Science and Computational Mathematics (PPG-CCMC) of University of São Paulo, Brazil, p. 195
38. Mealy GH (1967) Another look at data. In *Proceedings of Fall Joint Computer Conference*, pp. 525–534. Available at <https://www.computer.org/csdl/proceedings/afips/1967/5070/00/50700525.pdf>. Accessed 20 Apr 2018
39. Mejía-Gutiérrez R, Carvajal-Arango R (2017) Design Verification through virtual prototyping techniques based on Systems Engineering. *Res Eng Design* 28(4):477–494
40. Moodle (2018) Moodle: community driven globally supported. Moodle Partners. Available at <https://moodle.org/?lang=en>. Accessed 04 June 2018
41. MSpec (2008) MSpec. Available at https://github.com/machine/machine_specifications. Accessed 19 Oct 2016
42. Musen MA (2015) The Protégé project: a look back and a look forward. *AI Matters*, Association of Computing Machinery Specific Interest Group in Artificial Intelligence 1(4):4–12, <https://doi.org/10.1145/2557001.25757003>. Available at <https://protege.stanford.edu/>. Accessed 25 May 2018
43. NBehave (2011) NBehave. Available at <https://github.com/nbehave/>. Accessed 29 Jan 2017
44. North D (2006) *Introducing BDD*. Dan North & Associates. Available at <http://dannorth.net/introducing-bdd>. Accessed 5 Sept 2017
45. North D (2017) *What's in a Story?*. Dan North & Associates. Available at <https://dannorth.net/whats-in-a-story>. Accessed 5 Sept 2017
46. Okolnychyi A, Fögen K (2016) A study of tools for behavior-driven development. *Full-scale Software Engineering/Current Trends in Release Engineering*, Seminar Winter Term 2015/2016, Research Group Software Construction, RWTH Aachen University, pp. 7–12

47. OMG (2011) About the Business Process Model and Notation Specification 2.0. Object Management Group. Available at: <https://www.omg.org/spec/BPMN/2.0>. Accessed 04 June 2018
48. Oruç AF, Ovatman T (2016) Testing of web services using behavior-driven development. In: CLOSER 2016 – In Proceedings of 6th International Conference on Cloud Computing and Services Science, Vol. 2, pp. 85–92
49. Palmer SR, Felsing JM (2002) A practical guide to Feature-Driven Development. Prentice Hall, USA, p. 271
50. Protégé (2017) A free, open-source ontology editor and framework for building intelligent systems. Available at <https://protege.stanford.edu/>. Accessed 20 Sept 2017
51. Rhem J (1998) Problem Based Learning an Introduction. Natl Teach Learn Forum, Vol. 8, No 1, p. 07. Available at <http://www1.udel.edu/pbl/deu-june2006/supplemental/NTLF-PBL-introduction.pdf>. Accessed 5 Sept 2017
52. Rocha FG et al (2019) Agile Teaching Practices: Using TDD and BDD in Software Development Teaching. In Proceedings of XXXIII Brazilian Symposium on Software Engineering (SBES 2019), Brazil, p. 10. Available at <https://doi.org/10.1145/3350768.3351799>. Accessed 19 Jan 2021
53. Rodrigues RJS et al (2020) MyHealth: a system for monitoring non-communicable diseases. *Advances in Intelligent Systems and Computing*, Vol. 1134, Chap. 58, pp. 439–444. Springer International Publishing. Available at https://link.springer.com/chapter/10.1007%2F978-3-030-43020-7_58. Accessed 20 Jan 2021
54. RSpec (2016) RSpec. Available at <http://rspec.info/>. Accessed 19 Oct 2016
55. Ross DT (1977) Structured analysis (sa): a language for communicating ideas. *IEEE Transact Softw Eng* 3:16–34
56. Rubin KS (2012) *Essential Scrum: a practical guide to the most popular agile process*. Addison-Wesley, USA, p. 482
57. Santos HF et al (2016) Augmented Reality Approach for Knowledge Visualization and Production (ARAKVP) in Educational and Academic Management System for Courses Based on Active Learning Methodologies (EAMS–CBALM). In Proceedings of 13th International Conference on Information Technology: New Generations (ITNG 2016), *Advances in Intelligent Systems and Computing*, Springer International Publishing AG, Vol. 448, pp. 1113–1123
58. Schriml LM et al (2018) Human Disease Ontology 2018 update: classification, content and workflow expansion. *Nucleic acids research* 47(D1):D955–D962 Available at <https://academic.oup.com/nar/article/47/D1/D955/5165342>. Accessed 20 Jan 2021
59. Schwaber K, Sutherland J (2017) *The Scrum Guide™ - the definitive guide to Scrum: the rules of the game*. Scrum.Org and ScrumInc, p. 19. Available at <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100>. Accessed 12 Mar 2018
60. Seidman I (2006) *Interviewing as Qualitative Research: a guide for researchers in education and the social sciences*. Third edition, Teachers College Press, USA.
61. Silva T, Hak J-L, Winckler M (2016) Testing prototypes and final user interfaces through an ontological perspective for behavior-driven development. *Lecture Notes Comput Sci* 9856:86–107
62. Silva T et al (2017) A behavior-based ontology for supporting automated assessment of interactive systems. In: Proceedings of 11th International Conference on Semantic Computing, IEEE Computer Society, pp 250–257
63. Smith B, Welty C (2001) Ontology: towards a new synthesis. *Second International Conference on Formal Ontology and Information Systems*, p. 07. Available at <http://mba.eci.ufmg.br/downloads/recol/piii-foreword.pdf>. Accessed 20 Apr 2018
64. Soeken M, Wille R, Drechsler R (2012) Assisted behavior driven development using natural language processing. *Lecture Notes Comput Sci* 7304:269–287
65. Solis C, Wang X (2011) A study of the characteristics of Behaviour Driven Development. In Proceedings of SEAA 2011: 37th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE Computer Society, ISBN 978-0769544885, pp. 383–387
66. Souza PL et al (2017) Combining Behaviour-Driven Development with Scrum for software development in the education domain. In: Proceedings of 19th International Conference on Enterprise Information Systems (ICEIS 2017), SCITEPRESS – Science and Technology Publications Lda, Vol. 2, pp 449–458
67. Souza PL et al (2018) Improving Agile Software Development with Domain Ontologies. In: Proceedings of 15th International Conference on Information Technology: New Generations (ITNG 2018), *Advances in Intelligent Systems and Computing*, Springer International Publishing AG, Vol. 738, Chapter 37, pp 267–274
68. Souza PL (2018) *ScrumOntoBDD: an approach based on Scrum, Ontology and BDD for agile software development*. MSc's dissertation, Graduate Program in Computer Science (PPG-CC) of Federal University of São Carlos (UFSCar), Brazil, p. 179
69. SpecFlow (2016) SpecFlow. Available at <http://www.specflow.org/>. Accessed 19 Oct 2016
70. Stapleton J (2003) "DSDM: business focused development". Addison-Wesley, USA, p. 239.
71. StoryQ (2010) StoryQ. Available at <http://storyq.codeplex.com/>. Accessed 19 Oct 2016
72. TilyFy (2018) 9 Best business process modeling techniques (with examples). TilyFy. Available at <https://tallyfy.com/business-process-modeling-techniques/>. Accessed 04 June 2018
73. UFSCar (2007) "Curso de Medicina - CCBS Projeto Político Pedagógico". *Medicina UFSCar* (in Portuguese), p. 139. Available at: <http://www.prograd.ufscar.br/cursos/cursos-oferecidos-1/medicina/medicina-projeto-pedagogico.pdf>. Accessed 12 Oct 2016
74. W3C (2012) Web Ontology Language (OWL). W3C Semantic Web. Available at <https://www.w3.org/2001/sw/wiki/OWL>. Accessed 18 May 2018
75. W3C (2014) Resource Description Framework (RDF). RDF Working Group. Available at <https://www.w3.org/2001/sw/wiki/RDF>. Accessed 22 Jan 2021
76. W3C (2014) RDF Schema 1.1. W3C Recommendation. Available at <https://www.w3.org/TR/rdf-schema/>. Accessed 22 Jan 2021
77. Wohlin C et al (2000) Experimentation in software engineering: an introduction. Kluwer Academic Publishers, USA
78. Wynne M, Hellesoy A (2012) *The Cucumber Book: Behaviour-Driven Development for testers and developers*. Pragmatic Programmers LLC, Pragmatic Bookshelf, USA, p. 309
79. Zemmouchi-Ghomari L et al (2013) Process of Building Reference Ontology for Higher Education. In Proceedings of World Congress on Engineering, Vol. III, p. 06
80. Zubizarreta J (2009) *The learning portfolio: reflective practice for improving student learning*. Second edition, John Wiley & Sons Inc, USA, p. 354

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.