Software and Systems Modeling (2022) 21:1469–1494
https://doi.org/10.1007/s10270-022-01015-y

THEME SECTION PAPER

Agile MERODE: a model-driven software engineering method for user-centric and value-based development

Monique Snoeck[1] · **Yves Wautelet**[1]

Received: 15 November 2021 / Revised: 5 May 2022 / Accepted: 16 May 2022 / Published online: 16 June 2022

Abstract
Agile is often associated with a lack of architectural thinking causing technical debt but has the advantage of user centricity and a strong focus on value. Model-driven software engineering (MDSE) strongly performs for building a quality architecture and code, but lacks focus on user requirements and tends to consider development as a monolithic whole. The combination of Agile and MDSE has been explored, but a convincing integrated method has not been proposed yet. This paper addresses this gap by exploring the specific combination of MERODE—as an example of a proven MDSE method—with Scrum, a reference agile method offering a concrete (sprint-based) life cycle management on the basis of user stories. The method resulting of this integration is called Agile MERODE; it is driven by user stories, themselves associated with behavior-driven development scenarios. It allows for domain-driven design and permits fast development from domain models by means of code generation. An illustrative example further clarifies the practical application of Agile MERODE, while a case study shows the planning game application in the case's context. While the approach, in its entirety, allows reducing technical debt by building the architecture in a logical, consistent and complete manner, introducing MDSE involves a trade-off with pure value-driven development. Agile MERODE contributes to the state of the art by showing how to increase user centricity in MDSE, how to align model-driven engineering with the Scrum cycle, and how to reduce the technical debt of agile developments yet remaining value-focused.

Keywords Model-driven engineering · Agile · MERODE · User story · User stories · BDD · Behavior-driven development

1 Problem statement

According to the "State of Agile Report 2020",[1] up to 90% of the companies are at least experimenting with Agile. The most important reasons to adopt Agile are *better responsiveness to change*, the *desire to accelerate software delivery*, and *increased productivity*. While 42% of the respondents also consider *better software quality* as a reason for going Agile, *software maintainability* is considered by no more than 20% of the respondents. According to the same survey, the success of Agile is prevalently evaluated on business value and customer satisfaction, while defects, their resolution and pass/fail tests are considered by less than 20% of the respondents as success measures. This focus on fast delivery to quickly generate business value and satisfy customers makes Agile projects highly vulnerable to technical debt, leading to low maintainability, rework and delays [32].

In *model-driven software engineering* (*MDSE*) models are used as central skeleton for software development by generating code from them via transformations. The drawbacks of agile listed in the previous paragraph are reported as being partially addressed by the use of MDSE. Indeed, according to Liebel et al. [26], *quality*, *reusability*, *reliability*, *traceability*, and *maintainability* are the five most positively rated elements by MDSE practitioners in the domain of embedded systems. Also, according to Wortmann et al. [51], in relation to Industry 4.0 where by nature we require flexible tools for studying how to swiftly integrate software in a rapidly evolving and heterogeneous technological environment, the

[1] https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494.

Communicated by K. Lano, S. Kolahdouz-Rahimi, J. Troya, and H. Alfraihi.

✉ Yves Wautelet
yves.wautelet@kuleuven.be

Monique Snoeck
monique.snoeck@kuleuven.be

[1] KU Leuven, Leuven, Belgium

main benefits of modeling languages like the Unified Modeling Language [30] are *reducing time & costs* and *improving sustainability & international competitiveness*. These benefits are attributed to the fact that modeling supports digital representation and integration.

This paper explores the specific combination of MERODE[2] as example of a proven MDSE approach for domain-driven design with the Agile and a value-driven way of working. The interest of combining *Agile Software Development* (*ASD*) with MDSE lies in the fact that domain-driven design has proven to be efficient in delivering a well-organized architecture in larger agile projects [45]. Alfraihi et al. [5] show some indicators that Agile MDD has improved the efficiency and the quality of the developed software. A comparative case study by Lano et al. [25] also provides indications of improved efficiency and quality when combining Agile with MDD. We believe that our study is an early step in understanding the impact of integrating Agile development with MDD.

The paper mainly addresses the Research Question (RQ): "*How can a MDSE method like MERODE be anchored in an ASD structure for user-centric and value-driven (i.e., agile) development to minimize the technical debt?*" To answer the RQ, we build an overall approach based on domain-driven design on the basis of *Behavior-Driven Development* (*BDD*) scenarios (associated with specific *User Stories* (*US*)) with MERODE as *Domain-Specific Language*[3] (*DSL*) while achieving fast release by generating code from the domain model. Agility principles are sustained by the user-centricity inherent to the artifacts[4] used in the model-driven design (the BDD scenarios) as well as (partially) prioritizing the rapid development of specific (software) features based on US value. The framework we propose is called Agile MERODE; through the use of the method, with respect to ASD and MDSE, the strengths of one approach are used to address the limitations of the other and vice versa.

To intertwine ASD and MDSE, we anchor MERODE into the Scrum development life cycle. Within the classical MERODE method, the requirements analysis is achieved partially through a *Unified Modeling Language* (*UML*) use case model [30]; in Agile MERODE, the latter is **replaced by USs and BDD scenarios** that are user-centric and industry-adopted (thoroughly used in Scrum) agile requirements engineering artifacts. Also, no concrete iterative (or sprint-based) **life cycle support** is furnished by the traditional MERODE method that is essentially meant to be used in a plan-driven fashion. The use of USs within the requirements analysis of an Agile MERODE project supports the (sprint) planning game similarly as for traditional Scrum-based developments. The planning game [10] can thus be executed in a comparable fashion as for non-MDSE agile projects. The MDSE approach makes the precedence constraints on object construction that govern the architecture more explicit, as a consequence of which the prioritization of feature development will be less purely value-driven compared to non-MDSE projects. US are thus pivot elements in the Agile MERODE method, they are (i) associated with a few BDD scenarios that are the input for the model creation process and (ii) the scope elements for the planning game. The adoption of US in MERODE allows its swift anchoring in the Scrum life cycle. An illustrative running example shows the application of the method, and a more elaborated case study further clarifies the practical application of the planning game into an Agile MERODE-based development project. The approach, in its entirety, allows reducing technical debt by building the architecture in a logical, consistent and complete manner but involves a trade-off with pure value-driven because of architectural design precedence constraints. Agile MERODE contributes to the state of the art by showing how to increase user-centricity in model-driven development, how to align model-driven engineering with the Scrum cycle, and how to reduce the technical debt of agile developments yet remaining focused on value.

The remainder of this paper is structured as follows: Section 2 presents the essentials of the MERODE approach, including the relevant elements of its meta-model. Section 3 presents the research paradigm (design sciences) and the way the developed framework instantiates such a research cycle. Section 4 presents the combined approach of ASD and MDSE into the Agile MERODE approach starting with a presentation of the US and BDD meta-model and its integration with the MERODE meta-model and then turning to the Agile MERODE lifecycle and planning game. Throughout these sections, a small illustrative case study is used. Section 5 depicts Agile MERODE as a software process through a dynamic high-level view and a detailed static one. Section 6 applies Agile MERODE to a larger real-life case study and focuses on the planning game application for sprint content assignment. Section 7 discusses the specifics of the approach in terms of planning tradeoff, heaviness in method application and scalability, as well as the threats to validity. Furthermore, it discusses the approach in the light of related work. Finally, Sect. 8 concludes the paper.

[2] The name 'MERODE' originally stood for 'Model-driven Entity Relationship Object-oriented DEvelopment', referring in this way to the roots of the method.

[3] MERODE can itself be considered as a DSL for the domain-driven engineering of enterprise information systems. The language itself does not have mechanisms (e.g., such as stereotypes) to further adjust the language to specific domains.

[4] In this paper, the word *artifact* is used according to Scacchi's definition [35]: *a by-product of software development that helps describe the architecture, design and function of software*.

2 Background

2.1 Research gap and novelty of the approach

MDSE is not prescribed as an agile practice. While modeling is a well-established discipline, MDSE still faces a number of challenges. Among these we find current MDSE shortcomings in addressing complex software problems and MDSE tool usability issues [29], as well as the need for making MDSE valuable in the context of ASD [8]. Despite these challenges, combined ASD and MDSE approaches have proven to be successful in several domains [22]. Related work until 2016 has been described in a review of 15 studies published between 2001 and 2016 [3]. Many Agile MDSE approaches report positive impacts of a combined Agile-MDSE approach, such as better quality and productivity while maintaining typical ASD benefits such as faster development rate and better customer satisfaction. The study also reveals that the majority of approaches are MDD-based, meaning that Agile practices such as Scrum are incorporated in the MDD process. While the combination of Agile and MDSE can help to overcome the problems with the individual approaches [24], an exact roadmap on understanding how to exactly integrate ASD and MDSE is still lacking [4].

The Agile Unified Process [6] (AUP) constitutes a lightweight version of the Rational Unified Process (RUP) [21]. The AUP is intended to include agile techniques like *test-driven development* (*TDD*), *agile modeling*, *agile change management* and *database refactoring*. Because of the use of the RUP iterative structure (composed of 4 predefined phases themselves made of iterations), the AUP is rather devoted to integrating agile practices in a plan-driven environment than making use of MDSE in an inherently agile lifeycle. The RUP lifecycle is indeed a predefined canvas with overall project steps rather than a neutral (not inducing any predefined number of iterations or project length) sprint-based cycle. As seen in Christou et al. [9], the use of the RUP life cycle also implies that the releases produced at the end of each iteration are not necessarily deployable; the effective roll-out is mostly made in late stages of the project (at earliest within the *construction* phase). The AUP nevertheless deviates from pure UML-only modeling artefacts. Christou et al. [9] emphasize that "*Use cases are not the best for documenting business rules, user interface requirements, constraints, or non-functional requirements*"; consequently agile modeling artifacts like US can be introduced in AUP-based projects; often these are nevertheless used conjunctively with Use Cases leading to unnecessary overhead [47].

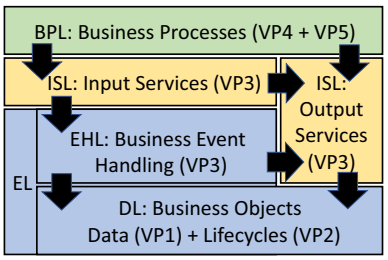

Fig. 1 MERODE layers: BPL (green), ISL (yellow), EL (blue) with two sublayers: EHL and DL

2.2 MERODE essentials

MERODE applies the principle of a layered architecture and identifies three major layers (see Fig. 1). The bottom layer is the *Enterprise Layer* (*EL*), containing two sublayers: the *Domain Layer* (*DL*) and *Event Handling Layer* (*EHL*). These layers are similar to the layers of the artifact-centric BALSA modeling approach of Hull [19]. The DL consists of the *business objects* and their associations. Additional logic is given in the *Object Life Cycles* (*OLCs*) defining the states a business object can be in, and the *business events* that cause the transitions between states. The EHL offers an interface to invoke the business events and routes these to the relevant business objects that will handle the event by means of a corresponding operation performing the required state changes. The *Information System Services Layer* (*ISL*) offers input and output services to access the EL. Output services allow querying the attributes and states of business objects. Input services capture input data but do not directly invoke operations on business objects. Rather, they achieve the requested input or update of information by triggering one or several business events via the EHL. The business events and their handling in the EHL allow combining the advantages of an event-driven architecture with the advantages of the layered architecture. Finally, the *Business Process Layer* (*BPL*) defines the work processes. Activities in the business processes may invoke the output and input services in the ISL to obtain information from the data layer and/or update information in the data layer. While the EL captures behavior on a per business object type basis, the BPL will capture other aspects of behaviour relating to users, task attribution and permissions [41].

Figure 2 shows the relevant parts of the MERODE metamodel. As the method uses a subset of the UML notation, its meta-model is much simpler than the one of UML. The blue box denotes the part corresponding to the DL's metamodel. It shows how the business object types are related by binary associations or inheritance associations (not shown in

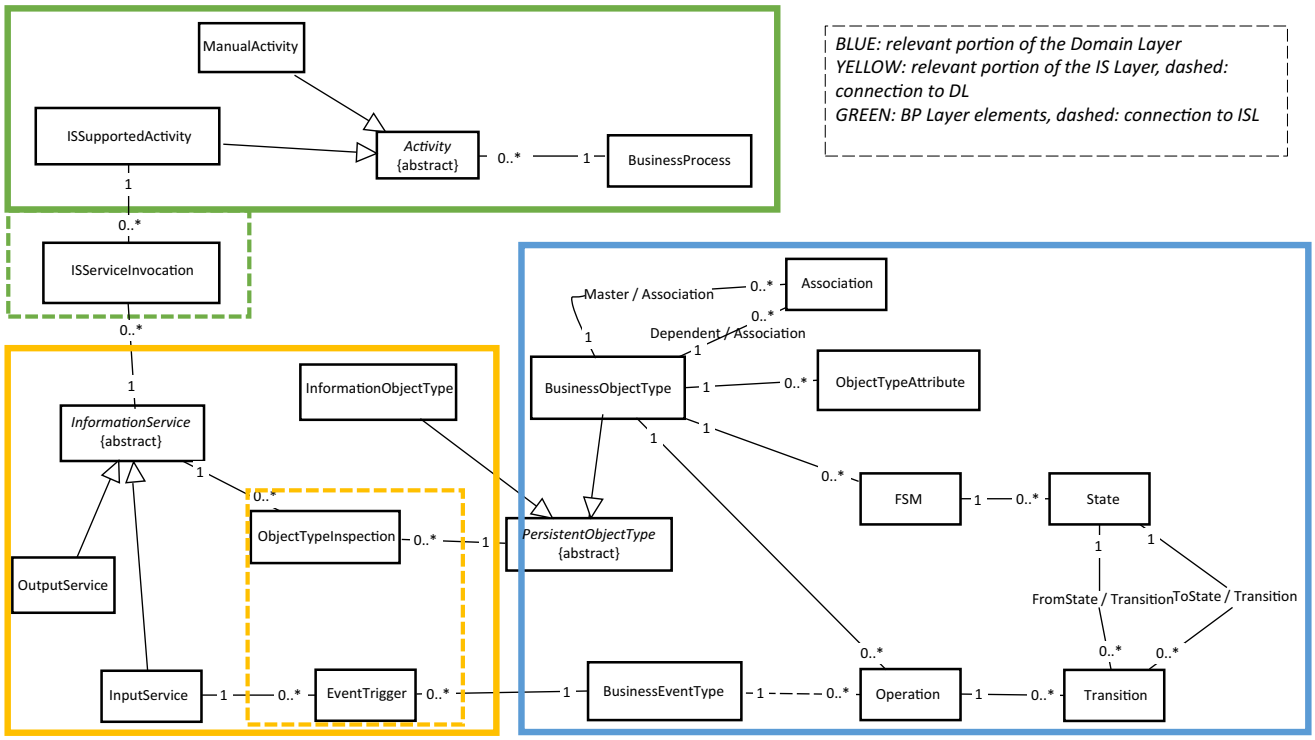

Fig. 2 MERODE meta-model (relevant part)

the meta-model). Business object types have attributes and operations. Each operation is triggered by exactly one type of business event. Conversely, a single business event may trigger an operation in several business objects, hence the need for an EHL.

The yellow box indicates the meta-model part corresponding to the ISL. The dashed part shows the meta-model elements that connect the ISL to the EL. The green box, finally, shows relevant elements of the BPL, and the connection to the ISL.

A specific aspect of MERODE is that it requires all associations in the UML class diagram to express existence dependency. In this way, the UML class diagram becomes a *Directed Acyclic Graph* (*DAG*), according to master-dependent associations. Such DAG can be obtained from a regular UML class diagram by systematically reifying all associations that do not express existence dependency. By transforming the UML class diagram in an *Existence Dependency Graph* (*EDG*), ambiguities are resolved, and consistency checking with OLC diagrams (the FSMs) becomes easier [39,40]. Thanks to this improved formality and correctness, the models from MERODE allow generating Java applications as prototypes of the EL with default *Information System* (*IS*) services in the ISL [38]. Alternatively, a web service interface can be generated for the DL and EHL. This allows connecting the prototype application to a business process engine, so as to be able to prototype the IS support for a business process. Currently, such code generation for the BPL is not yet available, though a first proof-of-concept with manual code writing has already been achieved [46].

3 Research paradigm, method and approach

This research follows the paradigm of Design Sciences (DS) [17]; DS aims to build generic solutions for identified issues. The result of DS-based research is some kind of artifact that can be a framework, terminology, a methodology, an engineering tool, software, and so forth. In this paper, an attempt has been made to build a methodology to engineer a software development project through models yet remaining compliant with fundamental agile practices. The built artifact is named Agile MERODE and aims to solve an unresolved issue or a problem considered being in a precarious state. In this research, we define Agile MERODE as a methodology to engineer software on the basis of USs, BDD scenarios, domain models, and transformations in order to increase the overall quality of the software architecture, thus reducing technical debt yet remaining maximally value-driven within a sprint's content determination. To answer the RQ given in the introduction and, in accordance with the DS research cycles defined by Hevner [16], we communicate an analysis of the *Relevance Cycle* in Sect. 3.1, the *Rigor Cycle* in Sect. 3.2 and the *Design Cycle* in Sect. 3.3.

3.1 Relevance cycle

The *Relevance Cycle* is concerned with the identification of opportunities/problems in the application domain. In the present context, we identify the problem of (heavy) technical debt induced by the use of pure agile methods. The problem has been identified in practice [18]. MDSE, by its structured and and holistic approach, allows to develop software having a well-organized architecture, thus ensuring a strong basis for optimizing code and minimizing the technical debt. The combination of both approaches, even if it requires some compromise, thus holds the promise of combining the benefits of each.

3.2 Rigor cycle

The *Rigor Cycle* refers to the theories/methods that are used to ground the construction and evaluation of the artifact. The methodology developed in this paper adapts the MERODE method for use in an agile (Scrum-based) context; MERODE as a standalone method was validated in the previous research, and this is therefore taken as a given. Also Scrum is taken as a given, for being the industry standard for ASD [28]. With respect to its classical lifecycle, MERODE was adapted/enhanced in terms of:

- **Requirements gathering and analysis** USs coupled with BDD scenarios are used to express requirements with a different level of detail; these artifacts (especially USs) are structural in Scrum because, besides their use for user-centric requirements elicitation, they are used to determine the sprint content. Indeed, USs receive a development priority factor based on their delivered value. They are thus the core artifacts to anchor into the Scrum life cycle. BDD scenarios offer the required details for US design and implementation. These industry-adopted agile modeling artifacts replace requirements elicitation with use cases and workflows traditionally used by MERODE;
- **Life cycle management** Agile MERODE has been built to be driven by USs (so these are the main scope elements within the Scrum's sprint planning (game)). Determining a sprint's implementation content is made by focusing on the (group of) stories delivering the highest value first. Although Agile MERODE aligns with this, architectural and design constraints do impose some precedence constraints leading to some deviations of the rule; a procedure to cope with this is thus detailed. The Scrum life cycle is the mostly used agile method in practice [28].

3.3 Design cycle

The *Design Cycle* refers to the construction and the evaluation of the artifact. As explained in the previous section, Agile MERODE has been built as an evolution of the traditional MERODE method. In order to proceed with the construction of the method, we used existing ontologies formalizing the type of elements present in USs [49] and BDD scenarios [44]. Elements constituting these ontologies have then been mapped to elements of the MERODE core ontology [38]. This way, forward engineering rules could be set up ensuring at the same time traceability between the requirements artifacts and the design models. An illustrative example is used to show the applicability and consistency of the framework through its constituting models' traceability. The evaluation of the planning game constrained by MDSE is done through a case study consisting in the development of a *Computer-Aided Software Engineering* (*CASE*) tool to support MERODE or Agile MERODE-based software developments.

4 The agile MERODE framework

4.1 Integrating user stories and BDD within MERODE

The aim of this section is to show the mapping between the concepts found in USs and BDD scenarios and the MERODE approach in order to define an integrated framework. As mentioned, USs and BDD are core concepts in ASD. Wautelet et al. [49] proposed a study of the US templates that are most frequently used in practice and unifies them in a consistent ontology. The latter allows to characterize functions of different nature without being overlapping yet exhaustive in the required coverage (so all of the possible USs instances could be associated with one and only one concept). The same approach was followed in [44], and both ontologies were merged to come to unification. While USs correctly depict a user situation, they often fail to give context to the execution of the functionality as well as enough details on how to implement the requirement. USs do nevertheless constitute the ideal communication artifact with customers and other stakeholders and the proper way to define sprints' content (the life cycle of Scrum) in terms of US to include in the produced release. USs and BDD scenarios are thus perfectly complementary and serve as the pivot onto which we interface with Scrum for MDSE. Implicitly BDD scenarios are always defined in the context of one and only one US. We here use the BDD scenarios (more detailed and specific) for forward engineering to the software architecture. To illustrate the connection between the above meta-model and these Agile concepts, we start from the set of USs for a Simple Shop[5] represented in Table 1. The MERODE representations of the Simple Shop as well as the prototype Java application

[5] Source: https://github.com/kunicmarko20/Simple-Shop/tree/master/features.

Table 1 User stories for the simple shop

1	Login	In order to buy cool products As a web user I need to be able to login
2	Order	In order to buy cool products As a web user I should be able to add products to cart and checkout
3	Product admin	In order to maintain the products shown on the site As an admin I need to be able to add/edit/delete products
4	Register	In order to login and shop As a web user I need to be able to create personal account
5	Subscription	In order to get cool product pack As a web user I should be able to subscribe to a plan
6	Card	If I continue using this shop As a web user I should be able to update my credit card info

generated from these and documented in this section have been placed on a persistent URL.[6]

As previously said, each US has a number of BDD scenarios attached to itself allowing to describe satisfactory scenarios for the testing and thus validation of the US. Example BDD-scenarios for the second US are given in Table 2.

Each of the US and scenarios allows distilling elements in the different layers. As an example, the order US "*In order to buy cool PRODUCTs, As a web user, I should be able to add PRODUCTs to CART and checkout*" assumes the domain concept of PRODUCT and CART, the business events of *add_product_to_cart* and *checkout* of a cart, and IS services with a web interface to add products and checkout. Looking at the individual scenarios, additional model elements can be derived. Scenario 2.2 suggests an attribute "quantity" for the association (class) relating a product and a cart. Scenario 2.3 suggests the need for card information, but from the formulation, we cannot infer that cards and their related info will be stored, so at this point, this information can be part of IS Service invocation and needs not (yet) to be captured as part of the domain layer.

Figure 3 presents the essential elements of a meta-model for BDD templates (extracted from [49] and [44]) that are relevant for this paper. When operationalizing the context part of a BDD scenario (respectively the outcome part), the precondition (respectively postcondition) may refer to the existence or any other particular state of business objects and to values of their attribute, e.g., a product exists, it is on sale, and the available quantity is larger than 5. Besides elements of the domain, also system status or user characteristics may be referenced. The User Behavior part will refer to a user action that invokes an information system service. In the case of an input service, this refers to triggered business events. Table 3 exemplifies this for the first two BDD scenarios for US2. When the "Given" and "Then" part of a BDD scenario are operationalized by formulating a precondition and postcondition, respectively, these Boolean expressions may refer to the states and attributes of *Business Object Types* (*BOTs*). The user behavior, on the other hand, can be operationalized as the invocation of an IS Service. These links are shown in Fig. 4 in the purple dashed box. As such, an interactive task in a business process can be further detailed by defining USs for this task and a set of corresponding BDD scenarios. The direct link from the interactive task to an IS Service invocation from Fig. 2 is thus now replaced by many-to-many associations to use cases, instantiated as "TaskOperationalisation" in Fig. 4.

As MERODE allows generating prototypes from domain models, many elements of BDD scenarios can be verified in just a few clicks once the domain model is specified. Minimal input suffices as, when creating a domain model, the MERODE tool generates default create and end events for each BOT, as well as a default "name" attribute in case no attributes have been specified. Figure 5 shows the domain model for the Simple Shop based on all USs and expressed as UML class diagram.

Figure 6 shows the main screen generated from this domain model, and how the "Given" part of BDD scenario 2.2 can be tested, i.e., viewing the product "Some Random Product". Figure 7 depicts testing the "When" part of this sce-

[6] https://zenodo.org/record/6470567.

Table 2 BDD scenarios for US2

Scenario 2.1: Adding new product to cart success
Given I am viewing product with Name "Super Random Product"
And I press "Add to Cart"
Then I should see "Product added to Cart"
And I should see one extra item in cart
Scenario 2.2: Adding extra already selected product to cart success
Given I am viewing the content of my cart
And my cart already contains the product with Name "Super Random Product"
When I press "+"
Then I should see "Product added to Cart"
And I should see one extra item for "Super Random Product" in my cart
Scenario 2.3: Adding card to cart success
Given I fill card field "card-number" with "4242424242424242"
Then I press "Checkout"
And I wait "10000" ms for javascript to process
Then I should see "Order Complete"
Scenario2.4: Coupons
Given I fill card field "card-number" with "4242424242424242"
And I press "I have a couponcode"
Then I press "Add"
And I should see "Missing couponcode!"
Then I press "I have a couponcode"
And I fill in "code" with "MEGAOFF"
Then I press "Add"
And I should see "- $500"
And I should see "Checkout forFree!"
Then I press "Checkout forFree!"
And I wait "10000" ms forjavascript to process
Then I should see "OrderComplete"
Scenario 2.5: Card declined
Given I fill card field "card-number" with "4000000000000002"
Then I press "Checkout"
And I wait "3000" ms for javascript to process
Then I should see "There was a problem charging your card: Your card was declined."

nario, i.e., adding the product to the cart, and Fig. 8 shows how the outcome (the "Then" part) can be tested.

4.2 Agile lifecycle support and planning game with agile MERODE

While the domain-driven spirit of the MERODE approach may suggest that it requires a plan-driven approach, we can proceed in an agile fashion. Indeed, inherently the model-driven architecture of MERODE natively supports incremental development. The key objects can be built up at first, and other dependent ones can be developed later on the basis of the previously developed architecture. Some constraints do exist, but the process remains largely flexible and value-driven. Moreover, each increment can lead to an executable (and deployable) release. The iterative and incremental support thus aligns with the agile principles. As an example, Fig. 9 shows how a minimal model containing just the two BOTs "Product" and "Customer" can already be prototyped. The fast and automated prototypes production—using the Merlin CASE-tool—at an early development stage also benefits close interaction between developers and customers while engineering the requirements; this also aligns with the precepts of agility.

When expanding the model, one nevertheless needs to observe the restriction that iterations and increments are constrained by referential integrity between objects so that sprints cannot address the development of features in any

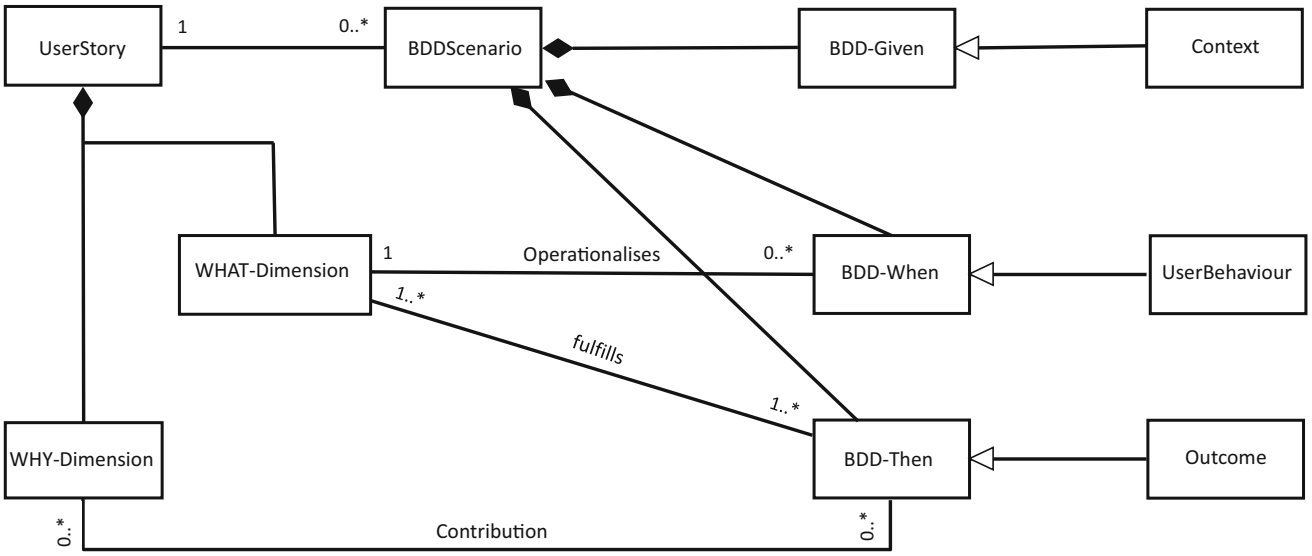

Fig. 3 Ontology for USs and BDD templates (extracted from [49] and [44])

order. MERODE's concept of existence dependency allows for the automated arrangement of objects according to their dependency. The "Top-Down-Level" identifies the absolute master BOTs that are not depending on any BOT as "level 0" BOTs. From there, BOTs that depend on other BOTs get a level that is one number more than the maximum level number of their masters. As such, looking at the complete domain model in Fig. 10, one can see that ITEMINCART has level 2 because it requires the existence of PRODUCT (level 0) and the existence of CART (level 1), which itself requires the existence of CUSTOMER (level 0).

The levels allow partitioning a domain model into partial models that can be implemented in separate sprints. The levels are indicative for sequencing the sprints according to referential integrity requirements. Figure 10 shows a possible partitioning with their dependencies. Partition 1 contains the BOTs Product and Customer which are required to be realized first, given all other BOTs are referencing them. The realization can be tackled in one or two sprints. As soon as this part of the DL has been implemented, US related to these BOTs can be addressed by implementing the required IS services. In next iterations, additional services can be realized for this same partition, or a next partition can be addressed. In the given example, Partitions 2A and 2B can be addressed in any order. The same goes for partitions 3A and 3B. However, the realization of these partitions will require the realization of partition 2A first, as both partitions 3A and 3B contain a BOT that references the BOT CART. BOT COUPON could have been added to partition 1, but its inclusion in partition 3B shows how realization of certain BOTs and their associated IS services can also be delayed if other parts of a system are considered to have a higher value-based priority.

Figure 11 depicts the global development process for the Simple Web Shop example. The sequencing arrows show the dependencies, and the parallel gateways indicate tasks that have no mutual dependencies. After developing the DL for partition 1, three different tasks can be added to the backlog and addressed either in parallel or in random sequence. The sprint planning can use value-based prioritization to decide upon the sequencing of the development of services and/or tackling a next DL partition. Each of the "*Implement IS Services ...*" has been adorned with a loop symbol as also the implementation of a set of IS services can be planned across several sprints. As such, the representation in Fig. 11 is imperfect as many more degrees of freedom are possible as long as the sequencing dictated by the architectural dependencies are respected.

Obviously, this way of identifying of architectural dependencies within the DL assumes a pre-existing global domain model. Creating a complete domain model, in particular for larger projects, may be contradictory to an agile approach. Creating only a partial domain model is better in line with ASD but may induce a technical debt; indeed, within a later sprint, the initial domain model and/or implementation of related services could reveal to be suboptimal and thus require refactoring of an earlier implemented DL partition and its related services. "Just enough modeling" as proposed by Agile Modeling[7] can help achieving a balance between agility and avoiding costly refactoring.

To summarize, the Agile MERODE life cycle should include a preliminary initial sprint (called sprint 0) where, from a project's initial US set, a first creation/generation of domain objects is made along the idea of "just enough mod-

[7] http://agilemodeling.com/.

Table 3 Mapping BDD scenarios to domain model elements and required IS services

BDD scenario		Business object type referenced	Attribute referenced	BOT state referenced	Input service referenced	Output service referenced
2.1	1. Given I am viewing product with Name “Super Random Product”	1: Product	1: Product Name	1: Product Exists	1:	1: View Product
	2. And I press “Add to Cart”	2: Cart	2:	2: Cart Exists	2: Create ItemInCart	2:
	3. Then I should see “Product added to Cart”	3:	3:	3:	3: Response Success/fail	3:
	4. And I should see one extra item in cart	4: ItemInCart	4:	4: ItemInCart exists	4:	4: View Cart, View ItemInCart
2.2	1. Given I am viewing the content of my cart	1: Cart	1:	1: Cart Exists	1:	1: View Cart
	2. And my cart already contains the product with Name “Super Random Product”	2: ItemInCart	2:	2: ItemInCart Exists	2:	2: View ItemInCart
	3. When I press “+”	3:	3:	3:	3: modifyItemInCart	3:
	4.Then I should see “Product added to Cart”	4:	4:	4:	4: Response Success/Fail	4:
	5. And I should see one extra item for “Super Random Product” in my cart	5: Cart, ItemInCart	5:	5: Cart Exists, ItemIncart Exists	5:	5: View Cart, View ItemInCart

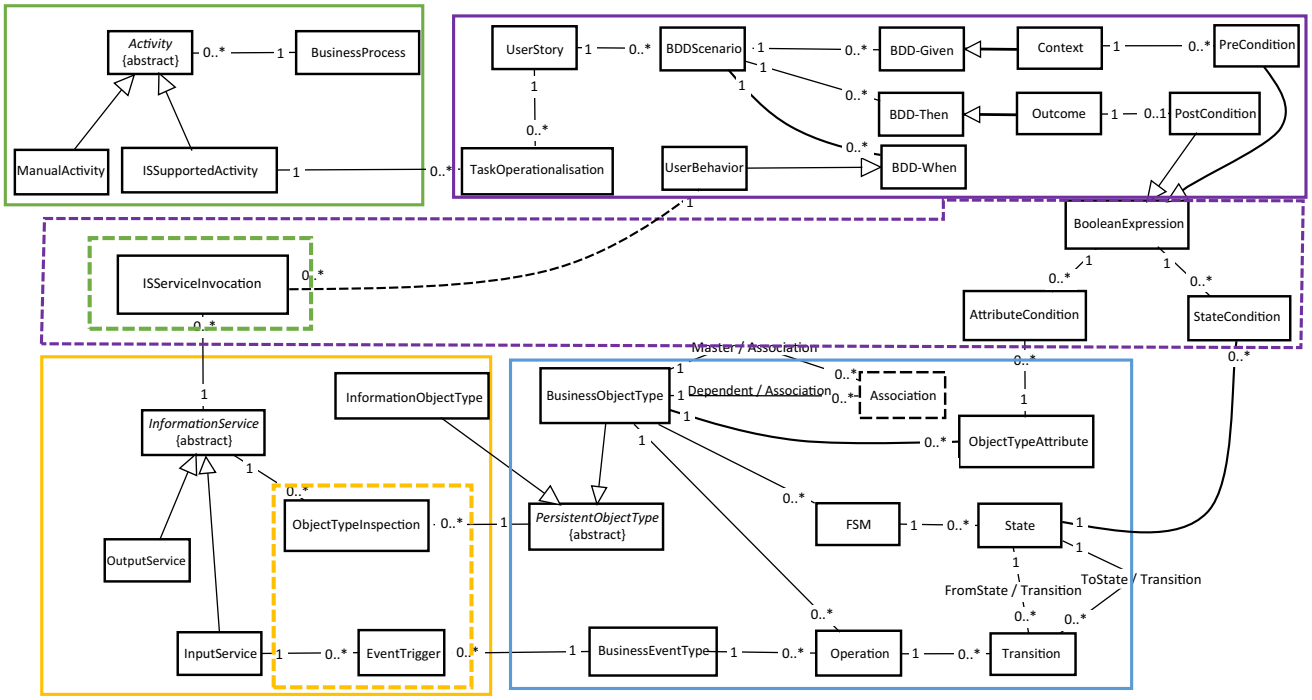

Fig. 4 Linking BDD elements to MERODE meta-model elements

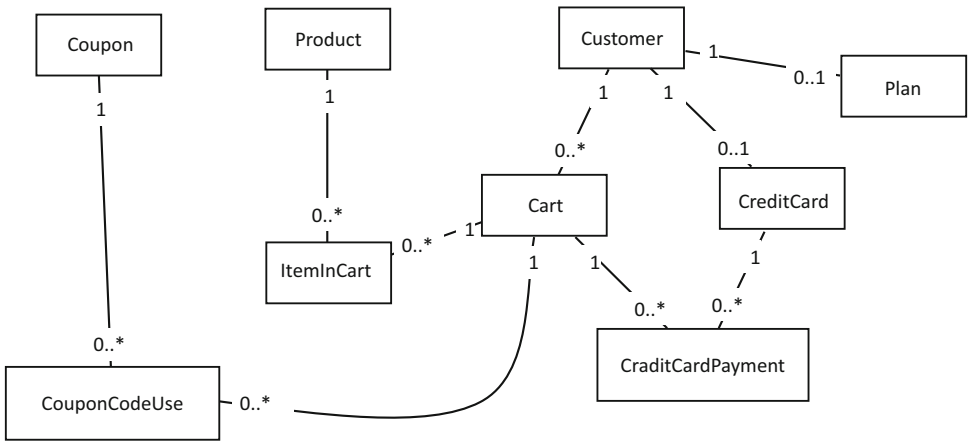

Fig. 5 UML Class diagram for the Simple Shop

eling". This constitutes the so-called baseline architecture of the project. Besides generating the baseline architecture, sprint 0 also serves for building an initial next sprint planning and, more importantly, determining the sequence constraints that might appear for the USs development into coming sprint(s). After sprint0, the product owner can determine the next increment content (realized in the next sprint) by selecting the USs (functions) on the basis of value if no other function needs to be pre-built; otherwise, the content is selected on the basis of the technical constraint.

5 Agile MERODE process view

To further document the integration of MERODE into the Scrum life cycle, we document, in this section, (i) a dynamic view of the Sprint sequencing with the impact on the produced artifacts (like software architecture, a prototype, sprint constraints, a sprint planning, etc.) in Sect. 5.1 and (ii) a static view of the actions taken within a sprint that we call the process fragment in Sect. 5.2.

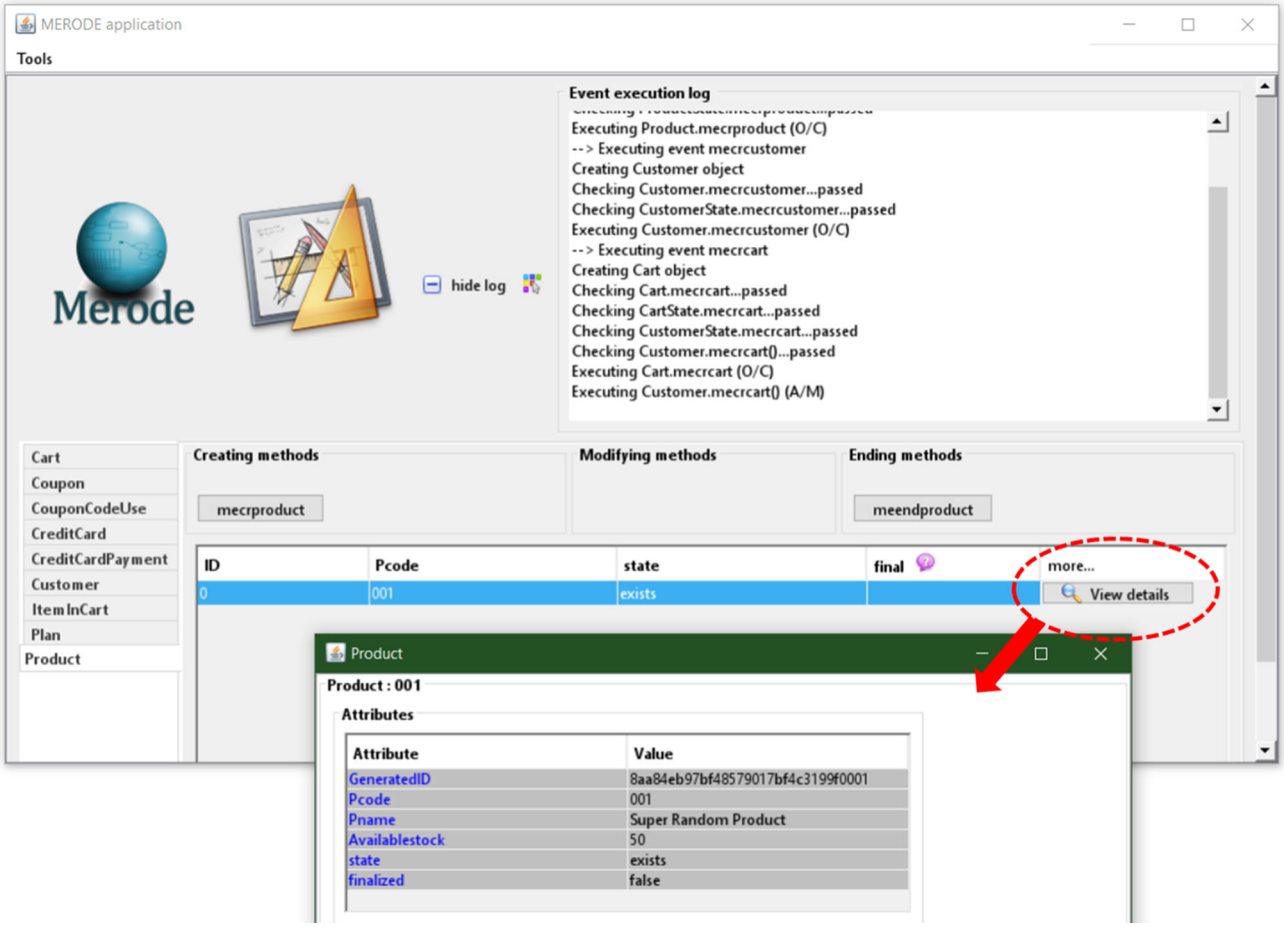

Fig. 6 Application generated from the domain model

5.1 Agile MERODE process structure

Figure 12 represents a (dynamic) view of the Agile MERODE process' Sprints. As discussed earlier, the process is initiated with Sprint0 that creates a baseline architecture, overviews USs precedence constraints and generates a first software prototype. Other Sprints devoted to produce executable releases supporting USs through their associated BDD scenario are then performed following the value provided by the features described in the USs and the precedence constraints. The construction of the *executable* releases requires hand coding the ISL as the default generated information services and user interface are not likely to satisfy the requirements of the end users. Nevertheless, the construction of the *executable* releases benefits from code generation for the EHL and the EL, which are strictly separated from the ISL and BPL. The MERODE code generator uses a template-driven approach, whereby the templates can be adjusted to the needs of a project. Such adjustments may be the source of template design activities performed in parallel with sprint implementation, as suggested by [14].

5.2 Agile MERODE as a process fragment

The i* framework [12] is used to represent the process fragment of Agile MERODE (it is a fragment because it is meant to be integrated in the broader Scrum process). I* was already used in a similar fashion to depict process fragment for other methods integrated or not within Scrum (see [43,48]). The benefit of i* is its ability to represent the social dependencies between the actors/roles while, at the same time, being static and thus non-sequential. Inherently to the nature of agility, different actions can be taken in different sequences for each of the Sprints: multiple activities can be performed at the same time, some can be omitted, others can be added and the sequence is always dynamically built up on the basis of the development context. Workflow-based notations are inherently more directive in terms of sequence, do not highlight social dependencies and are less tailorable/customizable and thus not capable of representing the flexibility needed in Agile processes. The i* notation better allows to deal with the variability in the activities' execution and selection. Finally,

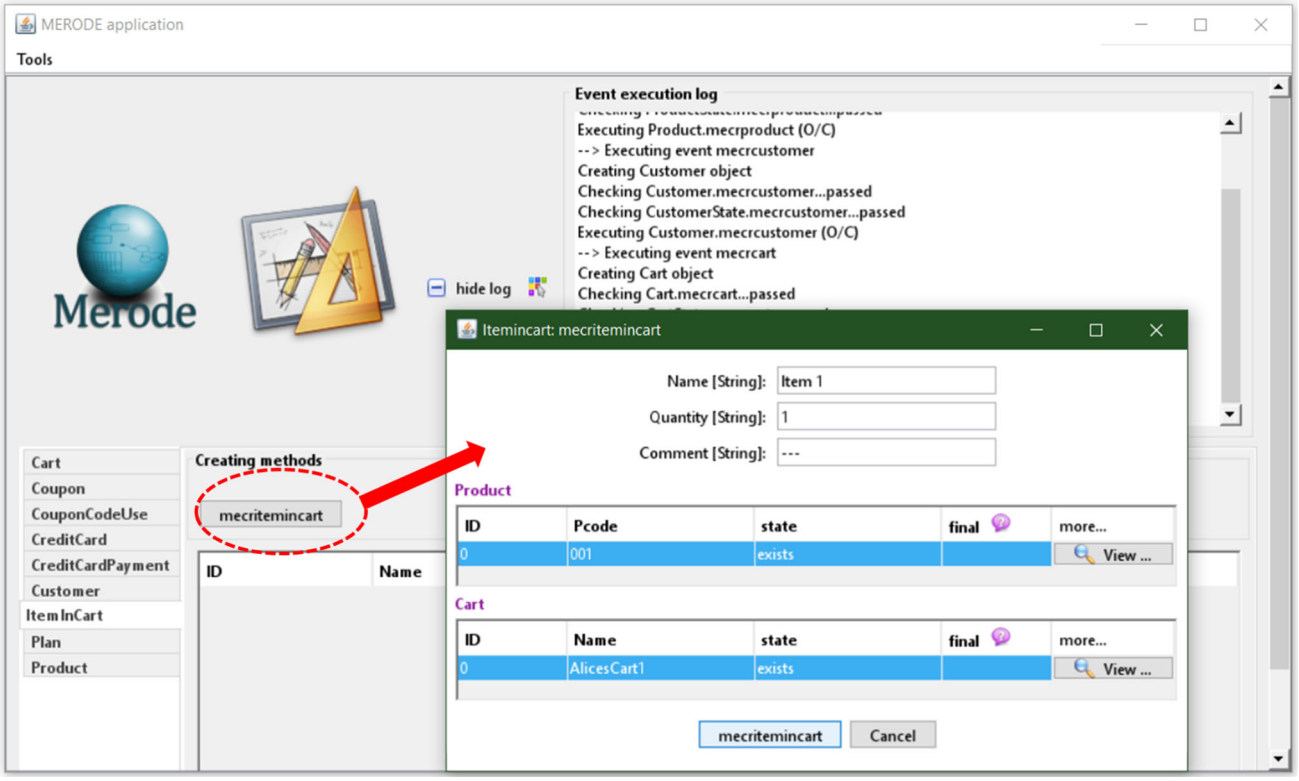

Fig. 7 Adding a product to the cart

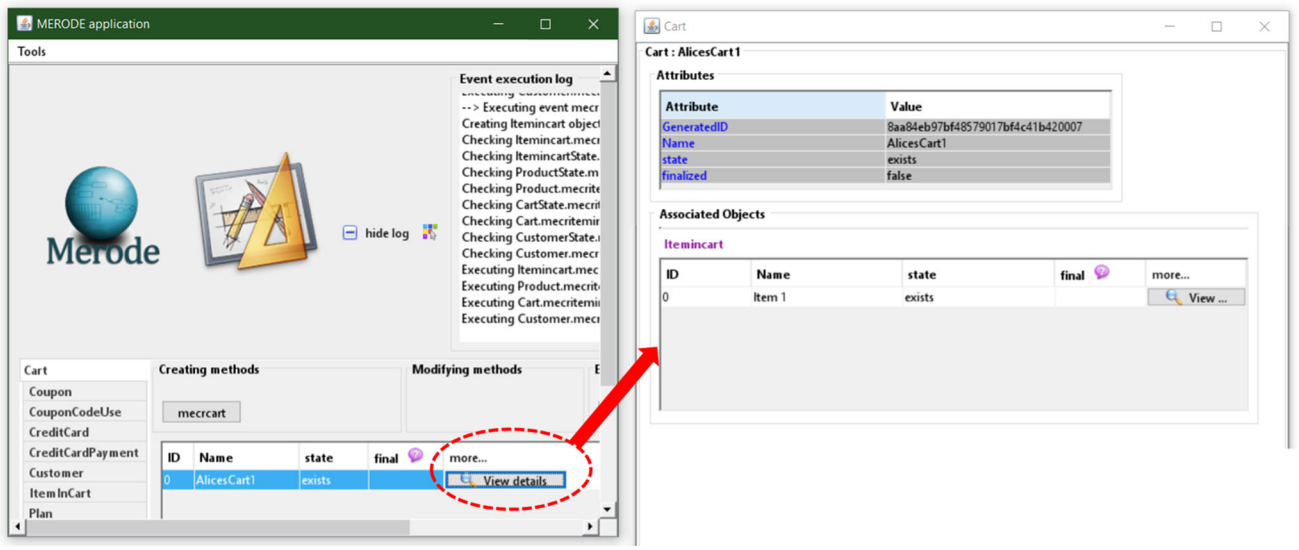

Fig. 8 Viewing the content of a cart

i* also allows to immediately link the tasks that *Help* or *Hurt* the realization of the Agile MERODE process' goals.

Figure 13 illustrates the Agile MERODE process fragment through an i* Strategic Rationale Diagram. It distinguishes five roles involved in the software development, more specifically:

– The *End User* ***Role*** is played by individuals that will be the users of the to-be software. The main ***Goal*** of the end user is to *Provide Requirements* and, as a means to the end of the goal, the *End User* performs the ***Task*** **Build Individual User Stories**. Involving *End Users* into this task *Helps* fulfilling the ***Softgoal*** of *User Centricity*. The

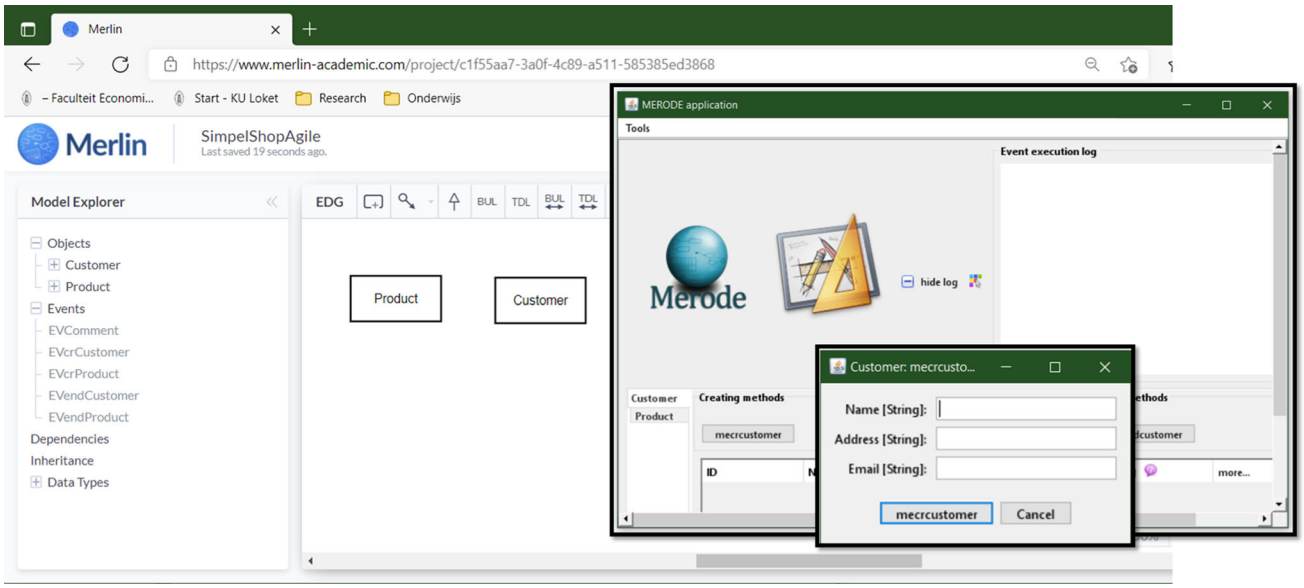

Fig. 9 Generating a prototype from a minimal model

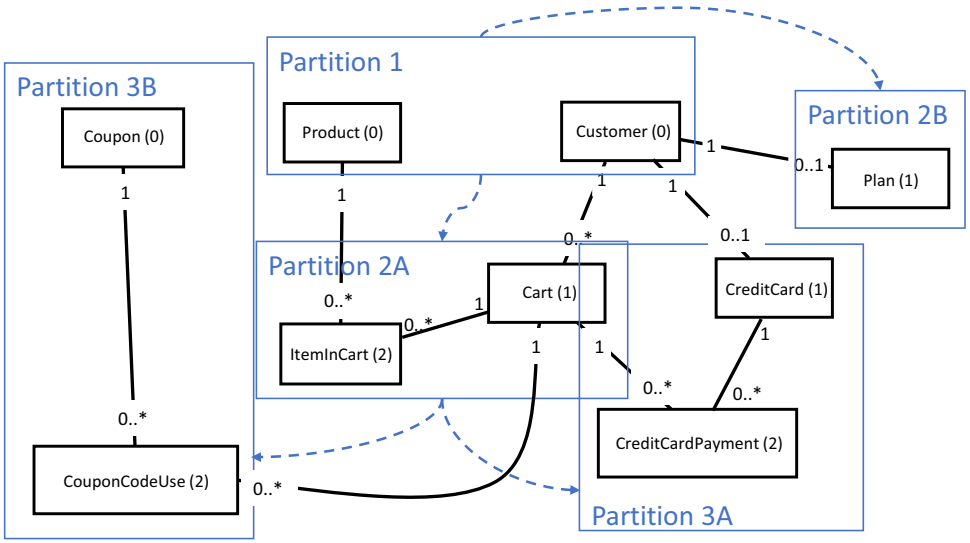

Fig. 10 Domain model with level numbers and possible partitioning

Product Owner thus depends on the *End User* to collect the *User Stories*;

– The *Product Owner* ***Role*** is played by individuals in the development team. The main ***Goals*** of the product owner is to *Gather Requirements* and *Ensure Project Development*. As a means to the end of the first goal, it performs the ***Task*** *Build/Structure User Story Set*. To achieve the end of the second goal, the product owner performs the ***Task*** *Manage Sprints*. The latter task is very important for the success of Agile MERODE since, for its fulfillment, it needs the realization of the tasks *Determine User Stories Value* and *Evaluate (User Story) Precedence Constraints*. The first task *Helps* the realization of the ***Softgoal*** *High Value Creation with Quick Access*, while the second *Hurts* it. These two tasks need to be realized to, in turn, realize the main task *Suggest Features to Implement in the Next Sprint*; these features are contained in the USs offering the highest value that are not facing any remaining precedence constraint anymore;

– The *Test Engineer* ***Role*** is played by individuals in the development team. The main ***Goal*** of the test engineer is to *Validate Features*. As a means to the end of this goal, it performs the ***Task*** *Build BDD Scenarios out of User Stories* which *Helps* the realization of the ***Softgoal*** *User Centricity*. The test engineer further transmits the *BDD Scenarios* ***Resource*** to the end user of which it depends on to fulfill the task *Validate BDD Scenarios*.

– The *Software Architect* ***Role*** is played by individuals in the development team. The main ***Goal*** of the software architect is to *Design Software*. As a means to the end

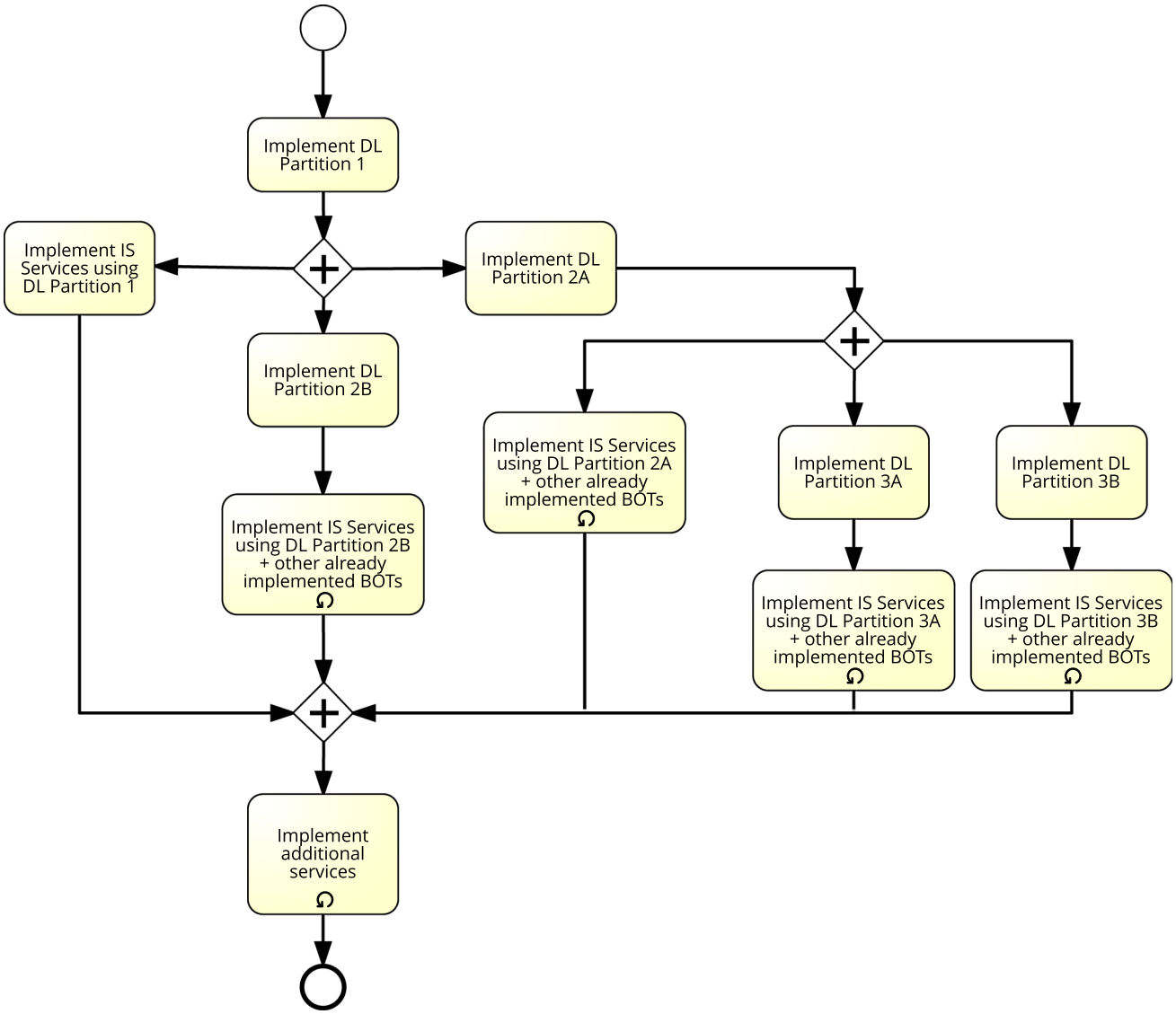

Fig. 11 Planning game

of the latter goal, it performs the ***Task*** *Structure Software Architecture*. The latter task is very important for the success of Agile MERODE since it needs, to be fulfilled, the realization of the tasks *Generate Baseline-Architecture* and *Refine Software Architecture*. The two former tasks *Help* the realization of the ***Softgoal*** *High Value Creation with Quick Access*. Indeed, it is further decomposed in the tasks *Generate Domain Model* and *Generate User Interface* allowing fast development of value-supporting software. Both generation tasks may involve the development of adjusted code generation templates. The generation of user interfaces may further benefit from the incorporating the FENIkS extension [34], which allows for defining a presentation model that captures UI requirements, and from which user interfaces can be generated accordingly. *Generate Baseline Architecture from BDD Scenarios* requires, to be achieved, the *Project User Stories (with BDD Scenarios) Backlog* ***Resource*** that should be furnished by the product owner role (social dependency).

– The *Developer* ***Role*** is played by individuals in the development team. The main ***Goal*** of the developer is to *Implement Features*. As a means to the end of this goal, it performs the ***Task*** *Implement User Story Support through BDD Scenario Realization*. Achieving the latter task requires the *Software Architecture* ***Resource*** furnished by the software architect role.

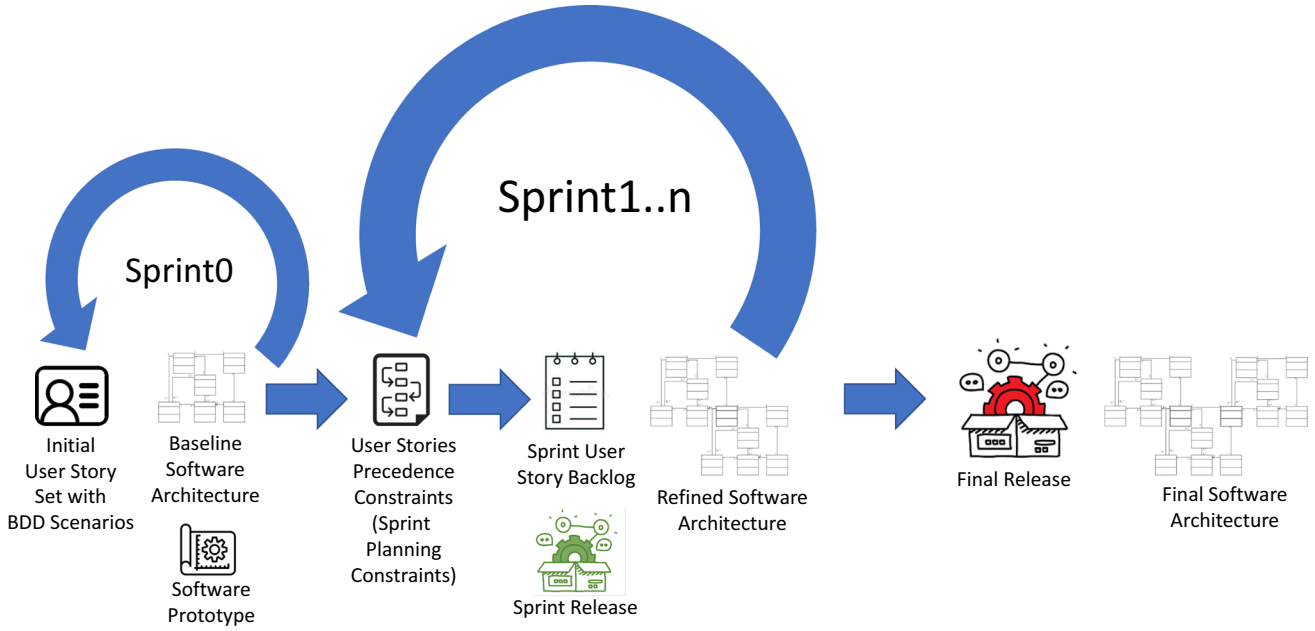

Fig. 12 General process architecture

6 Case study

Agile MERODE has been applied to the development of the CASE tool—called Merlin—aimed to support the MERODE or Agile MERODE methods' practitioners (students in business engineering at KU Leuven or professionals enrolled in the Master of Science in Enterprise Architecture at the IC Institute) when developing software in a model-driven fashion. In this project, that we use here as case study, the (development) team was structured following the precepts of Scrum. More specifically, a *Product Owner* (*PO*)—specialist in MERODE and coordinating all the demands and feedback coming from the user base—was reporting user requirements to the development team. The latter team was composed of one person being a user interface specialist (so designing these) but acting also as project coordinator and scrum master as well as 2 developers (pure coders). As can be seen in the rest of this section, version 1 had 13 sprints, while version 2 had 4 sprints; the total duration of the project was about 18 months for a total effort of approximately 600 hours. Purely technically speaking, the size of the project was 10.2 MB and involved 1.387 Files in 342 Folders.[8] The tool's first goal was to furnish a usable and bug-free environment to practitioners; it was developed using a combination of the *React* framework (*Javascript*) and *Node.js*. Master-level students studying MERODE and using the tool for the term assignment together with the method specialists served as a user-base for defining the requirements of Merlin but also to determine the value associated with the functionalities expressed in USs. As for classical Scrum-based developments, the PO served here as interface between the user base and the development team.

Agile MERODE's transformation rules from BDD scenarios (each of them associated with a specific US) to the software architecture, as defined previously in the illustrative example, have been applied on the case study leading to a base architecture. Since this forward engineering follows well-defined rules and partially benefits of the experience accumulated with the use of "non-agile" MERODE, we focus here on the scoping analysis and development sequence part of the project. Indeed, we want to study how the project has been divided into valuable features and how this value was evaluated at "Scrum planning game" time. This incorporation in the Scrum life cycle is perceived as a key component of agility and allows demonstrating the feasibility, advantages and drawbacks of the approach.

The development project has been split in two major versions. The main purpose was to align with the academic calendar that constrained the access to the user-base. Indeed, the first version was aligned with the second semester of the academic year 2018–2019 (so during the months February to May 2019) so that for each sprint an executable release could be produced and used by the user-base working on their term assignment (so implicitly tested and co-created with these users). A major version was made after a set of functions had been developed, and the semester came to an end. Each sprint took from 2 to 4 weeks. This first major version was presented as a fully integrated development that could work as a consistent whole and was viable as a standalone product

[8] According to https://ddiy.co/software-development-statistics/, this corresponds to the size of an average software development project.

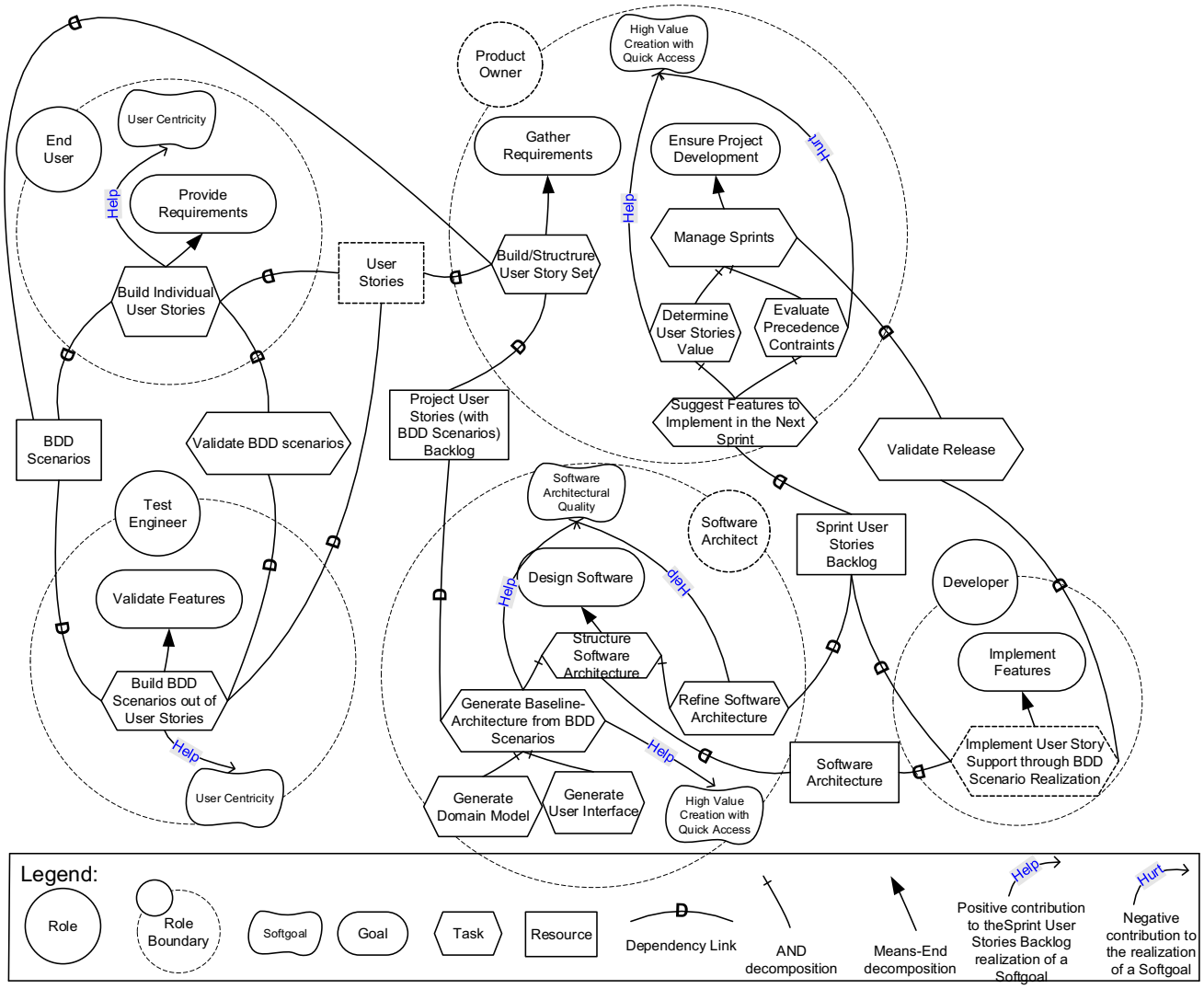

Fig. 13 Agile MERODE process fragment

for user support in development. A second version, improving on the first one, was developed after the summer break (so from October to December 2019); this second version essentially concerned refinements of version 1 with no major feature addition and is thus less interesting "planning-game wise". A document[9] presenting sample user stories from the MERLIN development project and explains how they were prioritized has been placed on a persistent URL.[10]

The sprints of the first major version could essentially all be planned on the basis of value even if precedence constraints did exist; this will be illustrated in the rest of this paragraph. For readability purpose, we do not get to the level of the individual US but aggregate the discussion at the level of the US theme[11] as summarized in Fig. 14. Sprint (1) was concerned with the building of a base environment in which the core functions of the application could be developed and was thus seen as preliminary programming work that had to be done independently of the traditional value frame and was not deployed to the entire user base. The next sprint (2) was the first one that concerned a core function of Merlin; its content was, at planning game time after sprint (1), driven by value (only). The PO indeed gave the building of the editor for the EDG (as seen earlier it is basically a refinement of the UML class diagram) as main priority to start up using the tool, USs related to this were thus grouped in a theme and developed. After that, the next sprint (3) concerned building a basic environment for the edition of *Object Event Tables* (*OET*); it was also driven by (highest) value as determined

[9] https://MERLIN-US-BDD-PrioritizationExample.pdf.

[10] https://zenodo.org/record/6470567.

[11] Themes are basically groups of USs around a particular topic that can be or need to be developed together.

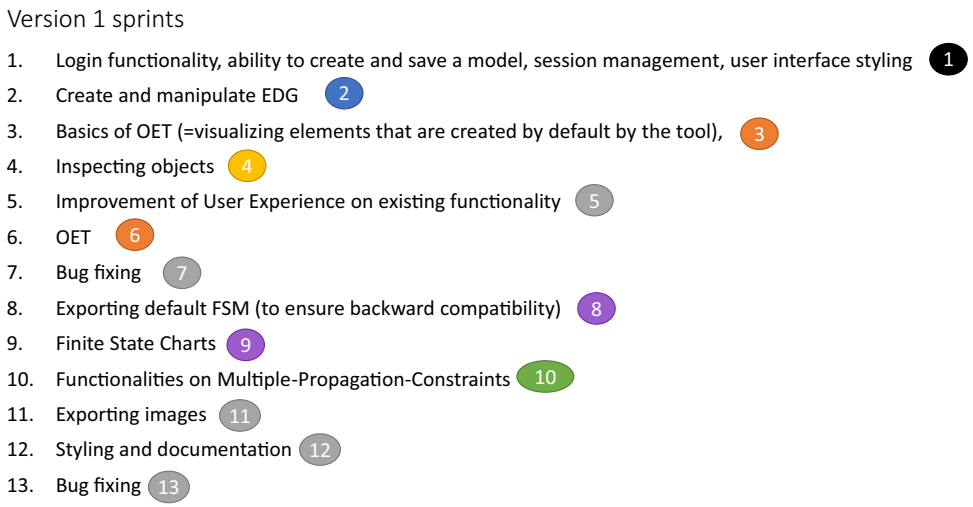

Fig. 14 Merlin version 1 sprints

by the PO after sprint (2). After sprint (4) involved the development of the *Inspection of Objects* feature (so group of USs around this theme); this needed the EDG to be developed as precedence constraint, but it appeared that it was already developed so no constraints needed to be respected for the sprint content to be purely value-driven (after sprint (3)). Sprint (5) focused on user experience and existing functionalities refinement on the basis of new requests from the user base as setup by the PO; this sprint was raised by specific feedback from the user-base and is thus driven by immediate user value. Sprint (6) focused on further developments related to the OET; this needed the basis of the OET to be developed as pre-condition, but it appeared that it was already developed so no constraints needed to be respected and sprint (6) content is driven by the highest perceived value at the end of sprint (5). Sprint (7) was devoted to some bug fixing as the user-base pointed out several elements that needed to be fixed/improved for a smooth modeling experience; this was thus driven by immediate user value. Sprint (8) concerned the exporting of *Finished State Machines* (*FSM*) diagrams to ensure backward compatibility. This allows using Merlin models for code generation, which is one of the most highly valued features of this tool; thus sprint (8) was purely driven by user value as defined by the PO at the end of sprint (7). Sprint (9) concerned the edition of Finished State Charts. Sprint (9) had sprint (8) as precedence constraint but the since the content of sprint (8) was judged more valuable ex ante, the two sprints were basically value-driven. Sprint (10) was the last main function of the version and concerned the functionalities on multiple propagation constraints. Finally, sprint (11) concerned the exporting of images, sprint (12) the styling and documentation and sprint (13) some bug fixing. These three last sprints did not concern core functionalities but were rather driven by the user-base feedback and late requests so driven by immediate value (and co-created with users).

As mentioned, a few months after version 1 was released, new developments with the same setting were performed to improve the quality, usability and functions of the Merlin CASE tool. No precedence constraints had to be respected, and the model-driven transformation process could be executed. These improvements did not impact the software architecture of the previous version but can rather be seen as a global add-on and refinement of the developed features. Requirements for these improvements (the list is given in Fig. 15) have been collected on the basis of user feedback obtained during the longer window of operation of version 1, and these stories were nevertheless perceived as of equal value and developed in an almost sequential manner. So, although the transformation process has been applied, agility was not a key component for version 2 due to the nature of the developments that were made.

Figure 16 summarizes the software architecture of the Merlin CASE-tool after version 2; the matching with the sequence of the sprints is given through the numbers. We can highlight an important finding from the sprint execution sequence that was established for version 1. In the end, all of the sprints (but the first one where a base environment for the tool was developed) have been driven by the highest user value for the next release when a sprint ended. Indeed, even if precedence constraints did exist, they have never concretely constrained the development sequence since the features that needed to be build first always had a higher user value than the ones they were constraining. We cannot establish a pattern on the basis of this single case study only, but it could appear that, in many cases, the precedence constraints induced by the model-driven development way of working are not constraining the final (sprint-based) development sequence (because the features that need to be developed ex ante tend to be more valuable for the users). This hypothesis, which is an important finding from the case study, should be further studied and will be the subject of future work.

Fig. 15 Merlin version 2 functions

Sprint 1:
Add support for inheritance to the merlin data model
Add support for inheritance to the MXP import
Add support for inheritance to the MXP export
Add support for visualising inheritance to the EDG
Add support for adding an inheritance relation between two object types
Add support for removing an inheritance
Add inheritances to the Model Explorer
Add inheritance to the object inspector
Add an inheritance inspector

Sprint 2:
Add support for visualising inheritance to the OET
Add support for adding a specialised event
Add info about specialization in the event inspector
Add support for inheritance to all event/method propagation code
Add via inheritance info in the method inspector
Add support for inheritance in the OET CSV export
Update (OET) remove tool to support specialised events/methods
Add support for adding a specialised/inherited method
Highlight propagation path of selected method in OET
Update sorting of methods in Model Explorer

Sprint 3:
Add support for inheritance to the existing checks
Add inheritance specific checks
Add support for inheritance to the event logging
Add inheritance to the learning report
Add support for inheritance to the image exports

Sprint 4:
Take inheritance into account when calculating paths / in MPCs
Update all commands for inheritance
Removing elements sometimes generates mobx-state-tree warnings
(BUG) Merlin hangs when creating ED from an object type to itself (Edge + Firefox)

7 Discussion, threats to the validity and related work

7.1 Impact of agile MERODE on the technical debt and communication improvements

This section briefly discusses how the technical debt is minimized through the application of the Agile MERODE process. The use and quality of the models is a key issue for producing qualitative software through an MDSE approach. This principle holds for the use of MDSE within an agile or non-agile fashion. For instance, the MERODE approach strongly focuses on the quality of the created domain model. The method has been grounded in process algebra, thus resulting in extensive quality checks as presented in the following papers [11,15,39,40]. We therefore consider the use of the MERODE approach per se as one element that contributes to the minimization of Technical Debt by its powerful quality checks on the domain model. The second contributing element is the domain-driven nature of the MERODE approach. We propose to strengthen the combination of Agile with the domain-driven approach by the use of a preliminary sprint (the so-called Sprint0) where, out of a project's initial US set, a first creation/generation of domain objects is made along the idea of "just enough modeling". The extent to which this initial domain model covers a sufficiently large portion of the final domain model is a strong instrument to avoid Technical Debt as much as possible. Moreover, MERODE's principle of existence dependency helps to clearly outline technical constraints to be considered when performing the value-based prioritization of USs during sprint planning.

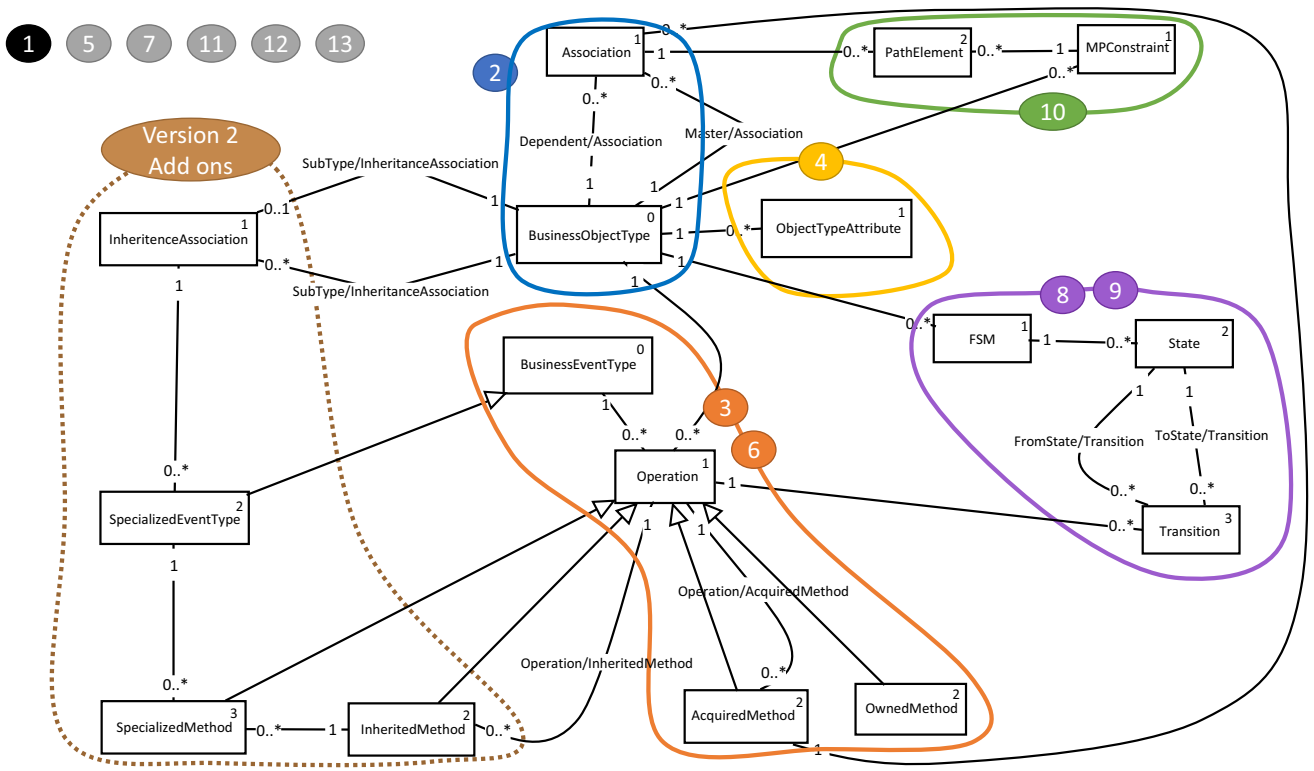

Fig. 16 The Merlin Software Architecture: the numbers correspond to the number of the sprint in which the architectural component has been developed

Benefits in terms of communication generated by our framework can be approached from different perspectives. First, when compared to a classical Scrum process, Agile MERODE and its intrinsic MDD allow fast (generated/no-code) prototyping. Thanks to this, workable interfaces can be delivered and validated early on in the software development. Second, when compared to a classical MERODE-based development, the inclusion in the Scrum life cycle allows (i) the adoption of USs, which is a requirement artifact allowing an easy communication with the end users, and (ii) a sprint-based approach generating user-feedback early on and continuously through the development allowing to optimize communication on requirements, expectations, domain-specific elements, etc.

7.2 Flexibility in planning

The (model-driven) approach from MERODE inherently leads to constraints in the technical development (complete coding) of the software which are inherent to any coding: whatever module using the service of another requires the latter to be created first, thus implying that a specific sequence in building and coding the objects of the object-oriented application needs to be respected. This obviously impacts the iterative (sprint) scheduling of software functional aspects. The selection of USs for the next sprint is thus constrained and cannot be realized in a totally neutral value-driven perspective. The benefit of the MERODE approach is that the layers and the clearly identified dependencies in the domain model provide clarity in what the technical dependencies are thus helping the sprint planner taking well-informed decisions. The approach, in its entirety, thus allows to reduce the technical debt by building the architecture in a logical, consistent and complete manner. In order to avoid technical debt, the MERODE life cycle thus always requires a compromise to the pure agile planning based on value. All in all, using the MERODE approach implies making a trade-off in the value-driven approach to get a clean software architecture for the software application delivered as ultimate project output.

The inclusion of such constraints in sprint-based planning is a known problem in approaches combining ASD and MDSE. Shafiee et al. [37] identify that, for the development of product configuration systems, based on so-called Product Variant Master models [20], precedence constraints do appear because of dependent features also influencing the sprint planning. In Agile MERODE, the planning game is nevertheless not comparable with the approach of the AUP. As said earlier in the paper, by following the RUP template inducing a clear segmentation in phases, AUP implicitly leads to a plan-driven iterative planning from the beginning of the project without the aim of producing a deployable release at the end of each sprint. Even if releases only par-

tially address the solution space, releases of each sprint of Agile MERODE are made to be deployable.

7.3 Heaviness and scalability in the method application

The application of the Agile MERODE method involves the characterization (tagging) of elements constituting the US and BDD scenarios. Even though this may be seen as a time investment from requirement engineers, the process in fact allows (i) to sort the requirements and make them more consistent and (ii) to spare a lot of design time.

Sorting the US is a natural process made in the agile development. The process is often made in the form of *User Story Mapping* (*USM*) [31], leading to grouping a set of US under a major feature contained in a so-called Epic US. Tsilionis et al. [42] have compared the process of building a Rationale Tree (which is a visual representation of US elements defined in [50]) with the ability to structure a set of US with USM through a controlled experiment. This experiment is relevant here because part of the work to build the visual model implies making the same kind of tagging of US elements as implied by the application of Agile MERODE. Tsilionis et al. [42] conclude that while a USM is easier to build-up, the evaluation of the US nature and structuring leads to discovering missing requirements and increasing the consistency of the entire US set. This ends up being very useful in the overall approach, and also, implicitly will reduce the technical debt if missing base requirements can be identified early on. As seen in the paper, when appropriately evaluated, the nature of the US and BDD scenarios' constituting elements lead to an easy process of forward engineering to a software architecture that can largely be automated through the use of the CASE-tool. Such automation is only possible through the use of MDSE and is thus generally not used in ASD while it leads to a significant design-time reduction.

Finally, we could question the scalability of the approach. As such, the larger the case, the more dealing with technical debt becomes an issue and the more the approach makes sense when compared to a classical ASD one.

7.4 Integration of MERODE within other agile methods and practices

As seen, Agile MERODE integrates the MERODE approach within the Scrum development life cycle. One could raise the question of how to integrate MERODE with other agile methods and/or practices. Basically, we can distinguish two ways of integrating ASD with MDSE. On the one hand, we can perform a structural integration of a MDSE-based method within an existing agile life cycle to anchor the practices of MDSE into ASD. This is precisely what we did to build Agile MERODE: we started from the Scrum lifecycle and anchored MERODE's MDSE practices to user stories and BDD scenarios (natural requirements engineering artefacts of Scrum). This way we now dispose of the sprint-based life cycle of Scrum enriched with new models and concrete support to build a neat software architecture. On the other hand, a myriad of agile practices are reported on in literature (see, for example, [23] for a literature review and an adoption approach). These practices are prescribed by some agile methods but can also be adopted independently of any agile method.[12] It is indeed not because a development team adopts an agile method that every prescribed practice should necessarily be included and, also, an agile practice can be adopted independently of the method prescribing it. Each known agile practice can thus be adopted within MERODE or Agile MERODE developments to increase the overall level of agility. Such an ad hoc adoption is, however, not a lifecycle adaptation exercise (as described formerly) but rather the punctual change of a method's application context to include a chosen practice(s). Most agile practices do indeed not have a structural impact on the software development process itself but rather impose contextual constraints on the way to operate (e.g., develop in pairs, be on the same site of the customer, etc.).

eXtreme Programming (XP) [7] is one of the earliest agile methods; its (structural) agile life cycle is much less advanced than the one from Scrum. Adapting MERODE to XP would thus rather lead to ad hoc agile practices adoption within the MERODE method's application context than generically restructuring MERODE around an advanced agile life cycle. As an example, pair programming, one of the core prescribed XP agile practices, can be integrated in a MERODE development project. Similarly, an artefact like the KANBAN board [1] can be adopted as a practice in an Agile MERODE development (which is classically done within Scrum developments). Several practices can thus be adopted in an ad hoc fashion if required. All in all, Agile MERODE integrates the Scrum life cycle because it is an advanced and widely validated life cycle leading a structured approach of agility in MDSE developments. It seems the best choice compared to less complete or less often used agile methods. Agility nevertheless remains a non-finite concept. The panel of agile practices at disposal and prescribed by various methods is broad, and new adoptions can be made to MERODE or Agile MERODE. The aim was to provide a basis for a structured approach to agility in MDSE that can be further customized to integrate more agile practices as required by the development context.

[12] See, for example, the Agile Subway Map for a method independent description of agile practices: https://www.agilealliance.org/agile101/subway-map-to-agile-practices/.

7.5 Threats to validity

7.5.1 Construct validity

A threat to the construct validity is that the modeling constructs are incorrectly interpreted by the modelers or any other practitioner aiming to put Agile MERODE into use. This includes the risk of using the concepts of the framework in another way than intended. In the present case, the application was done by the creators of the method so it does not constitute a serious issue. Nevertheless, to deal with such issues within the application of the method by others, specific guidelines can be integrated within the Merlin CASE tool providing help for the relevant concepts within the functions of the supporting tool.

7.5.2 Internal validity

The internal validity concerns the knowledge acquisition process and more precisely the objectivity of the views gathered. The knowledge acquisition holds no threat to validity for the authors being MERODE expert on the one hand and Scrum expert on the other hand. The issue of objectivity could on the other hand pose a threat, given the authors potential bias in favor of the method. The threat has been mitigated by careful analysis of other approaches and by the having the case study analysis performed by the author who was not involved in the project. Additional case studies where the method is applied by practitioners who were not involved in the method's development could provide additional robustness to the validity of the method.

7.5.3 External validity

The *external validity* is threatened by the incapability of applying the Agile MERODE method in other circumstances. The process has until now only been applied by the same set of people considered as experts in model-driven development and agility. Previous knowledge with techniques like US writing and mapping, BDD, UML and other techniques and notations can have a significant impact on the ability to apply the method. Experiments on the ability to perform US-based analysis and mapping (see [42]) by novice modelers and MERODE-based model-driven development [36] both showing the ability of novice modelers to apply the theoretical foundations with proper guidance. Also, MERODE is mainly applied in data-intensive domains, where the domain model is an essential part of the system to build. This does not preclude computationally intensive services using this data, but the design and modeling of such services are not supported by the MERODE method, since it only offers UML class diagrams and UML state charts as most essential models for the domain layer and relies on BPMN for the process layer. The use of an event-driven architecture when defining transformations for the EHL would allow to deal with reactive systems; more investigation is nevertheless necessary for defining an exact process to approach such systems in an agile fashion.

7.6 Related work

Overall, when comparing Agile MERODE to the related work reviewed in Alfraihi et al. [3] and published in more recent papers, the main distinctive factors are (1) that, in contrast to the dominantly MDD-based character of pre-2016 methods [3], Scrum is taken as a basis, and MDSE principles are incorporated via the combination with MERODE and (2) the explicit attention for how to deal with architectural constraints.

Table 4 provides an overview of the Agile and MDD practices used in [3] to compare the different approaches, together with the number of occurrences in the reviewed studies. The additional columns extend this overview with the proposals by Essebaa and Chantit [13], by Romano et al. [33], by Alam [2] and by the approach presented in this paper.

In terms of Agile practices, Agile MERODE addresses test-driven development through the use of BDD. Pair development is possible even if the practice adoption does not depend on the process itself but is rather a matter of internal organization of the development team working on a sprint. We consequently see it as not immediately related to the process as we have defined it. Continuous integration is essentially possible if the deployment constraints are treated upfront within the release development during the sprint. Agile MERODE could be supplemented in this way, but it is not included as such in the process as described in the paper; we indeed only focus here on the development teams' activities. Because of the combination with MDD, it is assumed that the need for refactoring is drastically reduced, and in case refactoring is needed, this should be addressed by means of redesigned code generation templates, thus being part of the MDD practices rather than the Agile practices. The remaining practices are addressed by the fact that Agile MERODE incorporates the Scrum practices.

In terms of MDD practices, MERODE generates code automatically from the domain model, which was created from a manual analysis of the US and BDD. The automatic (partial) creation of domain models from US and BDD as in [52] could be investigated in future work. The class diagram and state charts comply to UML modeling, but the OET is a MERODE-specific modeling techniques, here labeled as "Domain Specific". Models are not directly executable, except for the BPMN models that define the process layer. Model-to-model transformations are not used by default. The addition of UI-design models in MERODE has been explored by Ruiz et al. [34] and makes use of model-to-model trans-

Table 4 Evaluation and comparison of agile MERODE with respect to other frameworks

	Nr of studies that report the practice according to SLR	Agile MERODE	Essebaa and Chantit [13]	Romano et al. [33]	Alam [2]
Agile practices					
Test-driven development	5	BDD	Yes	Yes	?
Pair development	5	–	?	?	Yes
Continuous Integration	5	–	–	–	–
Refactoring	5	–	?	?	Yes
Prioritised Backlog	5	Yes	Yes	Yes	Yes
Direct Customer Involvement	5	Yes	Yes	Yes	?
Stand-up meeting	4	(Scrum)	(Scrum)	?	(Scrum)
Collective Ownership	2	(Scrum)	(Scrum)	?	(Scrum)
Self-selected team	1	(Scrum)	(Scrum)	?	(Scrum)
Burn-down Chart	1	(Scrum)	(Scrum)	?	(Scrum)
Release Planning	1	yes	(Scrum)	?	(Scrum)
MDD practice					
Automated code generation	13	Yes	Yes	Yes	Yes
UML modeling	7	Yes	Yes	Yes	Yes
DSL modeling	6	Yes (OET)	SBVR	Yes	Yes
Executable Models	4	BPMN	No	No	No
Model-to-model transformation	4	Possible [34]	Yes	?	?
XML modeling	2	No	No	No	No
Model-based testing	1	Yes [27]	Yes	?	?
Round-trip Engineering	1	No	No	?	?

formations for the generation of an Abstract User Interface. While we did not discuss the addition of model-based testing, this has already been explored in the context of MERODE [27]. Finally, XML modeling and round-trip engineering are not used by Agile MERODE.

Essebaa and Chantit [13] propose a combination of MDA, Scrum and the V-life cycle to improve the process of individual sprints and generating tests using model-based testing principles. The description of the approach mostly focuses on the MDD practices. As the authors build on Scrum, it can be assumed that most agile practices are incorporated in the approach, although this is not explicitly described. The authors also do not provide indications on how to address the architectural constraints in the planning of sprints. In Romano et al. [33], a framework and method for Agile and Collaborative Model-Driven Development framework for web applications (Web-ACMDD) is presented. While the description of Web-ACMDD provides an overall flow and a UML activity diagram describing the steps, the method is at an early stage of development. The actual incorporation of Scrum activities nor MDD practices is described in detail. On the other hand, the authors propose a UML profile to allow for the agile modeling of web applications. Here too, architectural constraints are not discussed. In contrast to our proposal, their sprint 0 does not advise to create a domain model. Portions of the domain model are created during the individual sprints. Besides domain class models and USs, this approach also includes models for Human Machine Interface.

Finally, the agile concern-driven development (CDD) process [2] is a reuse-focused development process in which an application is built incrementally by reusing existing building blocks. The framework builds on CORE, a framework inspired by separation of concerns, software product lines, goal modeling and aspect orientation. In the initial sprint, goal-oriented modeling languages are used to hatch out high-level requirements, upon which an iterative process develops the application by reusing concerns. The authors suggest that some reusable concerns such as concerns related to security, e.g., authentication or authorization, could be identified and reused early during the requirement phase, thus facilitating addressing architectural concerns. While the Sprint 0 advices the exploration of the domain by means of goal modeling, the creation of a conceptual domain model usable for code generation is not part of the approach.

8 Conclusion

At this stage, we can get back to the RQ stated in the introduction (*How can a MDSE method like MERODE be anchored in an ASD structure for user-centric and value-driven (i.e., agile) development to minimize the technical debt?*). To answer this RQ, we have developed the Agile MERODE framework. In terms of user centricity, it uses US and BDD scenarios as artifacts for the collecting and representing requirements. For flexible value-driven iterative development, the framework is anchored in the (industry-adopted) Scrum life cycle. The framework allows to reduce technical debt through the generation of a well-organized architecture on the basis of the (high-level) requirements elements found in the BDD scenarios related to US. A significant amount of time can also be saved through automatic generation of the software architecture.

The model-driven approach enables the automatic generation of a partial software architecture. Nevertheless, the software architecture implies a number of precedence constraints on the sequence in which USs are developed (e.g., in order to address USs related to behavioral modeling, we need the USs related to data modeling to have been implemented already). This did not appear to be an important issue at project time. Indeed, sprints do contain consistent developments in terms of common theme in the USs supported and the parts depending on other parts are often perceived by the user-base and the PO as less valuable than the ones that do not present such precedence constraints. Thanks to that, the pure value-driven way of working could be respected within the development of the Merlin tool. Future work involves the realization of more case studies in various domains to further test this hypothesis as well as further developments of the Merlin CASE tool to better support the sprint-based way of working through a custom interface; an expert opinion has also been collected and will be fully documented.

References

1. Ahmad, M.O., Markkula, J., Oivo, M.: Kanban in software development: a systematic literature review. In: 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, pp. 9–16. IEEE (2013)
2. Alam, O.: Towards an agile concern-driven development process. In: S.M.S. Jr., Armbrust, O., Hebig, R. (eds.) Proceedings of the International Conference on Software and System Processes, ICSSP 2019, Montreal, QC, Canada, May 25–26, 2019, pp. 155–159. IEEE/ACM (2019). https://doi.org/10.1109/ICSSP.2019.00028
3. Alfraihi, H., Lano, K.: The integration of agile development and model driven development: a systematic literature review. In: Pires, L.F., Hammoudi, S., Selic, B. (eds.) Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2017, Porto, Portugal, Feb 19–21, 2017, pp. 451–458. SciTePress (2017). https://doi.org/10.5220/0006207004510458
4. Alfraihi, H., Lano, K.: Practical aspects of the integration of agile development and model-driven development: an exploratory study. In: Burgue no, L., Corley, J., Bencomo, N., Clarke, P.J., Collet, P., Famelis, M., Ghosh, S., Gogolla, M., Greenyer, J., Guerra, E., Kokaly, S., Pierantonio, A., Rubin, J., Ruscio, D.D. (eds.) Proceedings of MODELS 2017 Satellite Event: Workshops (ModComp, ME, EXE, COMMitMDE, MRT, MULTI, GEMOC, MoDeVVa, MDETools, FlexMDE, MDEbug), Posters, Doctoral Symposium,

Educator Symposium, ACM Student Research Competition, and Tools and Demonstrations co-located with ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017), Austin, TX, USA, Sept 17, 2017, CEUR Workshop Proceedings, vol. 2019, pp. 399–404. CEUR-WS.org (2017). http://ceur-ws.org/Vol-2019/flexmde_3.pdf
5. Alfraihi, H., Lano, K., Rahimi, S.K., Sharbaf, M., Haughton, H.P.: The impact of integrating agile software development and model-driven development: a comparative case study. In: Khendek, F., Gotzhein, R. (eds.) System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering - 10th International Conference, SAM 2018, Copenhagen, Denmark, Oct 15–16, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11150, pp. 229–245. Springer (2018)
6. Ambler, S., et al.: The agile unified process (aup) (2005). Obtenido de Ambysoft: http://www.ambysoft.com/unifiedprocess/agileUP.html
7. Beck, K., Hendrickson, M., Fowler, M.: Planning Extreme Programming. Addison-Wesley Professional, Boston (2001)
8. Bucchiarone, A., Cabot, J., Paige, R.F., Pierantonio, A.: Grand challenges in model-driven engineering: an analysis of the state of the research. Softw. Syst. Model. **19**(1), 5–13 (2020). https://doi.org/10.1007/s10270-019-00773-6
9. Christou, I.T., Ponis, S.T., Palaiologou, E.: Using the agile unified process in banking. IEEE Softw. **27**(3), 72–79 (2010). https://doi.org/10.1109/MS.2009.156
10. Cohn, M.: Agile Estimating and Planning. Pearson Education, London (2005)
11. Dedene, G., Snoeck, M.: Formal deadlock elimination in an object oriented conceptual schema. Data Knowl. Eng. **15**(1), 1–30 (1995)
12. Eric, S., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering. MIT Press, Cambridge (2011)
13. Essebaa, I., Chantit, S.: Model driven architecture and agile methodologies: reflexion and discussion of their combination. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Proceedings of the 2018 Federated Conference on Computer Science and Information Systems, FedCSIS 2018, Poznań, Poland, Sept 9–12, 2018, Annals of Computer Science and Information Systems, vol. 15, pp. 939–948 (2018)
14. Guta, G., Schreiner, W., Draheim, D.: A lightweight MDSD process applied in small projects. In: 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, Patras, Greece, Aug 27–29, 2009, Proceedings, pp. 255–258. IEEE Computer Society (2009). https://doi.org/10.1109/SEAA.2009.63
15. Haesen, R., Snoeck, M.: Implementing consistency management techniques for conceptual modeling. Consistency Problems in UML-Based Software Development (2005)
16. Hevner, A.R.: The three cycle view of design science. Scand. J. Inf. Syst. **19**(2), 4 (2007)
17. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
18. Holvitie, J., Leppänen, V., Hyrynsalmi, S.: Technical debt and the effect of agile software development practices on it: an industry practitioner survey. In: Sixth International Workshop on Managing Technical Debt, MTD@ICSME 2014, Victoria, BC, Canada, Sept 30, 2014, pp. 35–42. IEEE Computer Society (2014). https://doi.org/10.1109/MTD.2014.8
19. Hull, R.: Artifact-centric business process models: brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, Nov 9–14, 2008, Proceedings, Part II, Lecture Notes in Computer Science, vol. 5332, pp. 1152–1163. Springer (2008)
20. Hvam, L., Mortensen, N.H., Riis, J.: Product Customization. Springer, Berlin (2008)
21. IBM: The Rational Unified Process, Version 7.0.1 (2007)
22. Kautz, O., Roth, A., Rumpe, B.: Achievements, failures, and the future of model-based software engineering. In: Gruhn, V., Striemer, R. (eds.) The Essence of Software Engineering, pp. 221–236. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-73897-0_13
23. Kiv, S., Heng, S., Wautelet, Y., Poelmans, S., Kolp, M.: Using an ontology for systematic practice adoption in agile methods: expert system and practitioners-based validation. Expert Syst. Appl. **195**, 116520 (2022)
24. KolahdouzRahimi, S., Lano, K., Alfraihi, H., Haughton, P.H.: Extreme modeling: an approach to agile model-based development. J. Comput. Secur. **6**(2), 43–52 (2019)
25. Lano, K., Alfraihi, H., Rahimi, S.K., Sharbaf, M., Haughton, H.P.: Comparative case studies in agile model-driven development. In: Hebig, R., Berger, T. (eds.) Proceedings of MODELS 2018 Workshops: ModComp, MRT, OCL, FlexMDE, EXE, COMMitMDE, MDETools, GEMOC, MORSE, MDE4IoT, MDEbug, MoDeVVa, ME, MULTI, HuFaMo, AMMoRe, PAINS co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, Oct 14, 2018, CEUR Workshop Proceedings, vol. 2245, pp. 203–212. CEUR-WS.org (2018)
26. Liebel, G., Marko, N., Tichy, M., Leitner, A., Hansson, J.: Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. Softw. Syst. Model. **17**(1), 91–113 (2018). https://doi.org/10.1007/s10270-016-0523-3
27. Marín, B., Bañados, S.A., Giachetti, G., Snoeck, M.: Tescav: an approach for learning model-based testing and coverage in practice. In: Dalpiaz, F., Zdravkovic, J., Loucopoulos, P. (eds.) Research Challenges in Information Science: 14th International Conference, RCIS 2020, Limassol, Cyprus, Sept 23–25, 2020, Proceedings. Lecture Notes in Business Information Processing, vol. 385, pp. 302–317. Springer (2020)
28. Maarif, D., Yusnorizam, M., Hafifi Yusof, M.F., Mohd Satar, N.S.: The challenges of implementing agile scrum in information system's project. J. Adv. Res. Dyn. Control Syst. **10**, 2357–2363 (2018)
29. Mussbacher, G., Amyot, D., Breu, R., Bruel, J., Cheng, B.H.C., Collet, P., Combemale, B., France, R.B., Heldal, R., Hill, J.H., Kienzle, J., Schöttle, M., Steimann, F., Stikkolorum, D.R., Whittle, J.: The relevance of model-driven engineering thirty years from now. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfrán, E. (eds.) Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, Sept 28–Oct 3, 2014. Proceedings, Lecture Notes in Computer Science, vol. 8767, pp. 183–200. Springer (2014)
30. OMG: Omg unified modeling language (omg uml). version 2.5.1. Technical Report (2017)
31. Patton, J., Economy, P.: User Story Mapping: Discover the Whole Story, Build the Right Product. O'Reilly Media Inc., Sebastopol (2014)
32. Rios, N., Mendonça, M.G., Seaman, C., Spínola, R.O.: Causes and effects of the presence of technical debt in agile software projects. In: Proceedings of the 2019 Americas Conference on Information Systems (AMCIS). Article 3, Cancun, pp. 10 (2019)
33. Romano, B.L., da Cunha, A.M.: An agile and collaborative model-driven development framework for web applications. In: Information Technology-New Generations, pp. 383–394. Springer (2018)
34. Ruiz, J., Sedrakyan, G., Snoeck, M.: Generating user interface from conceptual, presentation and user models with jmermaid in a learning approach. In: Ponsa, P., Guasch, D. (eds.) Proceedings of the XVI International Conference on Human Computer Interaction, Interacción 2015, Vilanova i la Geltrú, Spain, Sept 7–9, 2015, pp. 25:1–25:8. ACM (2015)

35. Scacchi, W.: Understanding the requirements for developing open source software systems. IEE Proc. Softw. **149**(1), 24–39 (2002). https://doi.org/10.1049/ip-sen:20020202
36. Sedrakyan, G., Snoeck, M., Poelmans, S.: Assessing the effectiveness of feedback enabled simulation in teaching conceptual modeling. Comput. Educ. **78**, 367–382 (2014)
37. Shafiee, S., Wautelet, Y., Hvam, L., Sandrin, E., Forza, C.: Scrum versus rational unified process in facing the main challenges of product configuration systems development. J. Syst. Softw. **170**, 110732 (2020). https://doi.org/10.1016/j.jss.2020.110732
38. Snoeck, M.: Enterprise information systems engineering: the MERODE approach. In: The Enterprise Engineering Series. Springer (2014). https://doi.org/10.1007/978-3-319-10145-3
39. Snoeck, M., Dedene, G.: Existence dependency: the key to semantic integrity between structural and behavioral aspects of object types. IEEE Trans. Softw. Eng. **24**(4), 233–251 (1998). https://doi.org/10.1109/32.677182
40. Snoeck, M., Michiels, C., Dedene, G.: Consistency by construction: the case of MERODE. In: Jeusfeld, M.A., Pastor, O. (eds.) Conceptual Modeling for Novel Application Domains, ER 2003 Workshops ECOMO, IWCMQ, AOIS, and XSDM, Chicago, IL, USA, Oct 13, 2003, Proceedings, Lecture Notes in Computer Science, vol. 2814, pp. 105–117. Springer (2003)
41. Snoeck, M., Smedt, J.D., Weerdt, J.D.: Supporting data-aware processes with MERODE. In: Augusto, A., Gill, A., Nurcan, S., Reinhartz-Berger, I., Schmidt, R., Zdravkovic, J. (eds.) Enterprise, Business-Process and Information Systems Modeling: 22nd International Conference, BPMDS 2021, and 26th International Conference, EMMSAD 2021, Held at CAiSE 2021, Melbourne, VIC, Australia, June 28–29, 2021, Proceedings, Lecture Notes in Business Information Processing, vol. 421, pp. 131–146. Springer (2021)
42. Tsilionis, K., Maene, J., Heng, S., Wautelet, Y., Poelmans, S.: Conceptual modeling versus user story mapping: which is the best approach to agile requirements engineering? In: Cherfi, S.S., Perini, A., Nurcan, S. (eds.) Research Challenges in Information Science: 15th International Conference, RCIS 2021, Limassol, Cyprus, May 11-14, 2021, Proceedings. Lecture Notes in Business Information Processing, vol. 415, pp. 356–373. Springer (2021). https://doi.org/10.1007/978-3-030-75018-3_24
43. Tsilionis, K., Wautelet, Y.: A model-driven framework to support strategic agility: value-added perspective. Inf. Softw. Technol. **141**, 106734 (2022)
44. Tsilionis, K., Wautelet, Y., Faut, C., Heng, S.: Unifying behavior driven development templates. In: 29th IEEE International Requirements Engineering Conference, RE 2021, Notre Dame, South Bend, USA, Sept 20–24, 2021. IEEE (2021)
45. Uludag, Ö., Hauder, M., Kleehaus, M., Schimpfle, C., Matthes, F.: Supporting large-scale agile development with domain-driven design. In: Garbajosa, J., Wang, X., Aguiar, A. (eds.) Agile Processes in Software Engineering and Extreme Programming: 19th International Conference, XP 2018, Porto, Portugal, May 21–25, 2018, Proceedings. Lecture Notes in Business Information Processing, vol. 314, pp. 232–247. Springer (2018). https://doi.org/10.1007/978-3-319-91602-6_16
46. Verbruggen, C., Snoeck, M.: Model-driven engineering: a state of affairs and research agenda. In: Augusto, A., Gill, A., Nurcan, S., Reinhartz-Berger, I., Schmidt, R., Zdravkovic, J. (eds.) Enterprise, Business-Process and Information Systems Modeling: 22nd International Conference, BPMDS 2021, and 26th International Conference, EMMSAD 2021, Held at CAiSE 2021, Melbourne, VIC, Australia, June 28–29, 2021, Proceedings. Lecture Notes in Business Information Processing, vol. 421, pp. 335–349. Springer (2021). https://doi.org/10.1007/978-3-030-79186-5_22
47. Wautelet, Y., Heng, S., Hintea, D., Kolp, M., Poelmans, S.: Bridging user story sets with the use case model. In: Link, S., Trujillo, J. (eds.) Advances in Conceptual Modeling: ER 2016 Workshops, AHA, MoBiD, MORE-BI, MReBA, QMMQ, SCME, and WM2SP, Gifu, Japan, Nov 14–17, 2016, Proceedings, Lecture Notes in Computer Science, vol. 9975, pp. 127–138 (2016). https://doi.org/10.1007/978-3-319-47717-6_11
48. Wautelet, Y., Heng, S., Kiv, S., Kolp, M.: User-story driven development of multi-agent systems: a process fragment for agile methods. Comput. Lang. Syst. Struct. **50**, 159–176 (2017)
49. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I.: Unifying and extending user story models. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) Advanced Information Systems Engineering: 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16–20, 2014. Proceedings, Lecture Notes in Computer Science, vol. 8484, pp. 211–225. Springer (2014)
50. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I., Poelmans, S.: Building a rationale diagram for evaluating user story sets. In: Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, June 1–3, 2016, pp. 1–12. IEEE (2016)
51. Wortmann, A., Barais, O., Combemale, B., Wimmer, M.: Modeling languages in industry 4.0: an extended systematic mapping study. Softw. Syst. Model. **19**(1), 67–94 (2020). https://doi.org/10.1007/s10270-019-00757-6
52. Yue, T., Briand, L.C., Labiche, Y.: Atoucan: an automated framework to derive uml analysis models from use case models. ACM Trans. Softw. Eng. Methodol. (2015). https://doi.org/10.1145/2699697

Monique Snoeck holds a PhD in computer science from KU Leuven. She is full professor at the Research Center for Management Informatics (LIRIS), KU Leuven, and visiting professor at the UNamur. She has a strong research track in conceptual modeling, requirements engineering, software architecture, model-driven engineering and business process management. Main guiding research themes are domain modeling, business process modeling, model quality, model-driven engineering, and technology-enhanced learning. Previous research has resulted in the Enterprise Information Systems Engineering approach MERODE, and its companion e-learning and prototyping tool MERLIN and its companion prototyping tool. She is author of 2 books, (co-) author of over 130 peer-reviewed papers. She is associated editor of the BISE journal and (senior) member of the program committee of numerous conferences in the domains of Information Systems such as CAiSE, RCIS, PoEM and EMMSAD.

Yves Wautelet holds a PhD in economics and management from UCLouvain. He is professor at the Research Center for Management Informatics (LIRIS), KU Leuven (Brussels), and visiting professor at the UNamur. Yves has pursued research in software project management, requirements engineering, conceptual modeling, multi-agent systems, model-driven development as well as IT governance and strategy. He has published over 90 peer-reviewed papers. He is member of the program committee of numerous conferences in the domains of Information Systems such as ER, REFSQ, RCIS, PoEM and EMMSAD