




## Bridge Between Agile and Traditional Methods: Agile Requirements Documentation Structuring (ARDocS)



Hind Wissam Kalfat<sup>1,2\*</sup>, Mourad Oussalah<sup>2</sup>, Azeddine Chikh<sup>1</sup>

<sup>1</sup> Computer Sciences Department, LRIT Laboratory, Tlemcen University, Tlemcen 13000, Algeria

<sup>2</sup> LS2N, CNRS, Ecole Centrale Nantes, Nantes Université, UMR 6004, Nantes F-44000, France

Corresponding Author Email: [hind-wissam.kalfat@univ-nantes.fr](mailto:hind-wissam.kalfat@univ-nantes.fr)

Copyright: ©2024 The authors. This article is published by IETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/isi.290601>

### ABSTRACT

**Received:** 2 September 2024

**Revised:** 13 November 2024

**Accepted:** 26 November 2024

**Available online:** 25 December 2024

#### Keywords:

*software requirements engineering, traditional requirements, agile requirements documentation, hybrid documentation, meta-modeling*

Software Requirements Engineering (SRE) varies significantly between agile and traditional methods, particularly in documentation practices. In traditional methods, for instance, the team is required to produce one structured and detailed document which is the software requirements specification. While agile methods require less documentation, which is spread over several artefacts. These differences can lead to communication challenges in hybrid development environments, where both agile and traditional teams collaborate. In such contexts, documentation can serve as a crucial communication tool, bridging the gap between the two methods. This paper proposes ARDocS approach, which translates agile artefacts into a structured document compatible with traditional methods. ARDocS involves defining and specifying agile and traditional documentation through multiple abstraction levels using metamodeling, and mapping the concepts between these two metamodels. We validate our approach through a case study that applies ARDocS to the Scrum method for agile and the VOLERE template for traditional. ARDocS effectively consolidates information from various agile artefacts into structured documentation that can be understood and used by both agile and traditional teams.

## 1. INTRODUCTION

Software requirements engineering (SRE) is considered as the most important aspect of software development as it consists on understanding the product to be developed.

SRE is a process that differs from one method to another: in traditional approaches like waterfall, SRE is carried out at the very beginning of the project, enabling an upstream planning of the entire product. The plan is then strictly followed by the development team throughout the rest of the project [1]. These approaches are highly effective in safety-critical systems. According to Martins and Gorchek they are preferred in such domains because the new ones are not yet mature or not convincing enough [2]. Indeed, projects in defence or healthcare need a strict regulatory standard that takes every detail into account; to insure they stay within budget and time while addressing the correct requirements. On the other side, in agile approaches, SRE is informal and highly dependent on individuals' knowledge and skills. It is carried out throughout all the Software Development Life Cycle (SDLC) and requirements evolve as the project progresses [3]. These approaches are widely adopted by companies that prioritize customer feedback and adaptability. For instance, Microsoft's commitment to agile principles helped the company to continuously evolve, stay at the forefront of technological innovation and respond to market demands. By embedding SRE throughout the development cycle, they can address constantly changing needs and refine the product in real time.

The knowledge gathered during the SRE process must be documented. This documentation contributes to the success of the project, reducing the risk of misunderstandings and ensuring that the software development process is well managed and transparent. The documentation also varies depending on the approach used.

In traditional approach, for instance, the team is required to produce the software requirements specification document (SRS). The SRS is deemed good if it is unambiguous, comprehensive, precise, and can be easily communicated with all of the involved stakeholders [4]. In some cases, it can also be used as an integral part of the contract [5].

In agile approach, teams tend to minimise documentation in order to focus on software development, as outlined in the agile Manifesto [6]. However, it's acknowledged within the agile community that a certain level of documentation is essential [7] and even agile developers consider documentation to be an important issue [8]. During SRE applied in agile context, teams use a variety of artefacts [9], with some being specific to a particular agile method like SCRUM, such as user stories or backlogs, while others are not.

There are also other differences between the requirements documentation of these two approaches, such as the roles responsible for it, the style adopted and the content that makes up the documentation. All this influences the way in which the documentation is created and used throughout the project.

Nonetheless, these dissimilarities can have significant consequences, especially in the case of hybrid development.

Hybrid development involves integrating both agile and traditional methods within a single project. Organizations intentionally opt for hybrid development when they deem agile suitable for certain scenarios, while preferring a more traditional method in others [10]. We think that the best way for teams using these hybrid methods to communicate with each other is through documentation as it provides a clear and consistent reference point for all team members. However, documentation produced by agile teams is challenging for traditional teams to understand. This is due to its level of detail, which is considered to be very low, but also to its structure, which is different.

This research presents a solution that bridges the gap between agile and traditional methods within the context of hybrid development. The solution involves transforming the fragments of documentation produced during an agile development process into a structured requirements documentation. Our main contribution is a new approach, named ARDocS (Agile Requirement Documentation Structuring), which is based on metamodeling techniques across different layers to abstract and generalize documentation concepts for both agile and traditional methods, facilitating their reuse and enhancing their understanding. The designed metamodels are used in a mapping process to establish connections between their concepts. Ultimately, the goal is to improve communication, collaboration, and overall efficiency in software development processes through enhanced documentation. In this paper, we make three contributions:

- (1) Specifying both agile and traditional SRE, using metamodels, specifically from the point of view of documentation
- (2) Structuring agile documentation artefacts, according to a traditional documentation structure, using a third mapping metamodel.
- (3) Presenting of a case study for demonstrating the application of ARDocS approach.

The remainder of this paper is structured as follows. Section 2 gives an overview about SRE in agile and traditional methods and also introduce the metamodeling approach. Then, section 3 presents the related works and points out the research gap. Section 4 describes the proposed approach for creating and instantiating the metamodel. Section 5 details ARDocS approach and describes the metamodels. We validate the proposed approach in section 6 using a case-study. Finally, section 7 presents our conclusions, discusses validity of our research and indicates directions for further research.

## 2. BACKGROUND

SRE plays an important role in the success of SDLC. Effective requirements documentation is essential to ensure clear communication, alignment of stakeholder expectations and the overall quality of the software product. This section provides a comprehensive background on the subject. It first defines requirement concept, then highlights the differences between traditional and agile methods in SRE, with a particular focus on the documentation activity, which remains a sensitive issue in both methods, mainly in the second one. It finally presents the metamodeling approach.

### 2.1 Requirement definition

There is much debate about what should and should not be

considered a requirement, as well as the necessary characteristics it should have. However, the BABOK guide ensures that the term “requirement” is understood in the broadest possible sense. The BABOK states that Requirements include, but are not limited to, the past, present and future conditions or capabilities of a business, as well as descriptions of organisational structures, roles, processes, policies, rules and information systems [11]. It is this definition that will be considered in this research work.

### 2.2 Traditional requirements engineering (TRE)

In TRE such as the waterfall model, all requirements are elicited and analysed before the actual development process begins. Sommerville and Sawyer claim that RE covers all of the activities involved in discovering, analysing, documenting, and maintaining a set of requirements for a system [12]. Throughout this TRE process, it is imperative for the team to thoroughly document all discussed requirements within a software requirement specification. The former could be supported by additional documents describing other types of requirements. To facilitate the documentation process in TRE, several templates are available, enabling the team to adopt a precise structure. At the final stage, all documented requirements have to be implemented afterwards.

The documented requirements serve as the foundation for verification activities, ensuring that the customer’s needs are accurately and fully captured. As a result, these requirements are formally certified as the accepted specifications to be implemented [13, 14]. Various techniques are used by teams to review requirements. One such technique is the walkthrough review [15], which is an informal process where feedback is gathered regarding the technical content of the software product document. Another valuable technique involves prototyping, which helps stakeholders verify that their needs are correctly understood and addressed. In the case of safety-critical systems, more formal methods may be used, which involve converting requirements into a mathematical model to assess their consistency and completeness with respect to system properties [16]. Additionally, it is crucial during the verification process to ensure that the requirements are testable. Testing-oriented techniques focus on evaluating whether use cases derived from the requirements are practical and easy to generate. Test case generation not only provides early feedback from users but also guides developers in understanding how the system should behave [16]. This early verification process of TRE offers numerous benefits, such as the early identification of errors and gaps, which ultimately leads to better-defined product requirements. It also helps uncover additional requirements that need to be addressed before the design phase [17].

### 2.3 Agile requirements engineering (ARE)

The ARE process differs significantly from the TRE process. Indeed, the elicitation, analysis, specification, validation and management of requirements in agile are performed iteratively and carried out in collaboration with the stakeholders.

In ARE, various techniques are used across its key activities, as identified in a survey conducted by Elshandidy and Mazen [18]:

- Elicitation: Direct, face-to-face communication is a primary technique, enabling stakeholders to share their needs through structured interviews or collaborative sessions such as

Joint Application Development (JAD).

- **Analysis:** The development team examines the collected requirements for completeness, consistency, and feasibility. This phase often involves prioritizing user stories to focus on the most critical elements for the upcoming iterations.

- **Specification:** Agile teams typically use concise user stories to outline functional requirements from the end user’s perspective. These stories, usually written on simple note cards, are prepared by the customer’s team to ensure they are expressed in business language they understand.

- **Validation:** Review meetings are a common technique, allowing stakeholders to interact directly with the product. This hands-on approach facilitates immediate feedback, ensuring any issues or concerns are identified and addressed early in the process.

Encouraging customer involvement in agile development practices promotes a preference for more face-to-face communication, which reduces the volume of documentation. However, there are some situations when the communication is insufficient like: sudden changes in requirements, unavailability of appropriate client representatives, project complexity, customers at distributed geographical locations and not collocated or onsite [19]. Therefore, it becomes cumbersome to tackle such situations with little or no documentation [20]. In fact, many studies investigated the challenges that result from ARE practices, with documentation consistently identified as one of the prominent issues [19, 21]. Agile is conducted by a set of values and principles described in the agile manifesto, but there are multiple methods applied with different practices. For this research, we opted for Scrum, identified as the most widely used agile development method [22].

### 2.4 Metamodeling approach

Metamodeling defines abstract models that describe the structure, behavior, and semantics of other models. It has the same objectives as a modelling act, the only difference being the object of the modelling. This approach is very useful as it helps for:

- **Standardisation:** By defining meta-metamodels and metamodels, we create common standards and structures for documentation in both agile and traditional methodologies. This facilitates consistency and understanding between teams working in different environments.

- **Reuse:** Metamodels enable to define abstract models that can be reused in different projects.

- **Comparison:** By matching the concepts of agile and traditional methods, we make it easier to compare their practices and processes. This can help developers identifying similarities, differences and best practices to adopt.

- **Defining and integrating multiple models:** It allows to define and integrate multiple documentation models, offering significant flexibility and adaptability. We can take into account different methodologies, documentation templates, or even specificities of each organisation.

## 3. RELATED WORKS

In this section, we explore existing work related to four key areas to clearly describe the research context and understand the current state of the issues being addressed. In the first part, we present the papers that provide a specification for both

ARE and TRE. Then, we introduce the concept of hybrid development. The second part examines what has been proposed in the literature in relation to our problematic. Here, we discuss the techniques currently used to improve the structuring of documentation in the context of ARE. Then, we investigate the metamodeling approach used in this research, providing an overview of its application and relevance in the context of SRE. Finally, we summarise the state of art.

### 3.1 The specification of traditional and agile methods

Agile is generally defined in terms of its methods. There are several agile methods, practices and frameworks (for example, Scrum, XP, Kanban, Safe, LeSS). Each of these methods uses its own terminology. There are studies that have modelled the concepts of ARE in its entirety [23], and others that have done so for specific agile methods [24]. On the other side, traditional approach is usually represented by the waterfall method. However, there are other traditional methods such as V-model, Incremental model, and Spiral model. Batool et al. [25] did a mapping of traditional RE and agile Scrum RE with respect to their roles, activities and artefacts. In Table 1 we can see how TRE and ARE specified throughout the RE process.

**Table 1.** Definition of TRE and ARE according to literature (Adapted from Batool et al. [25])

|                   | TRE   | ARE   |
|-------------------|---|---|
| <b>Role</b>       | Software Engineer,<br>System Analyst,<br>Requirement Engineer.  | Stakeholder, Product<br>Owner, Development<br>Team, Scrum Master.   |
| <b>Activities</b> | Requirement Elicitation,<br>Requirement Analysis,<br>Requirement<br>documentation,<br>Requirement validation,<br>Requirement management | Refine the backlog,<br>Update Sprint Backlog,<br>Prioritize functions,<br>Project status meeting,<br>project demonstration<br>meeting, Retrospective<br>meeting |
| <b>Artefacts</b>  | Valid requirements,<br>Software Requirements<br>Specification   | Vision, Product backlog,<br>Sprint backlog, User<br>stories.  |

### 3.2 Hybrid development

Several studies discuss hybrid development approaches, all defining hybrid as a combination of both agile and plan-based (or traditional) methods. The strength of the hybrid approach lies in its ability to combine the benefits of both methods in a single project and therefore maximise its chances of success.

Prenner et al. [26] found that hybrid development approaches are primarily used to meet security and safety requirements, manage large-scale or distributed development, and remain adaptable to changes while dealing with uncertain requirements. They identified 13 goals for using agile methods and 12 goals for using plan-based methods in Agile-Plan-based Hybrid approaches (Short: APH approaches). Indeed, plan-based methods are suitable for large-scale projects or safety-critical applications but face challenges when customers do not fully understand requirements or when frequent changes occur. On the other hand, agile methods are known for their ability to adapt to changing requirements and technology choices. The authors suggest that combining the two methods can create a more flexible and adaptive development process that accommodates evolving customer needs and project complexities.

Hess et al. [27] compare the scope and content of the

outcome produced by the ARE practices to artefacts of a traditional software development framework. This enabled identifying the differences, similarities, and potential gaps between the information communicated through various artefacts in TRE and ARE practices.

Boehm and Turner [28] identify key challenges in integrating agile and traditional methods, citing those related to RE, which include:

- **Documentation Practices:** Traditional methods rely on extensive documentation, while agile methods prioritize working software, which may result in insufficient records for future use. Enhancing agile practices, such as enriching user stories with more detail, can help bridge this gap.
- **Requirements Formality:** The informal nature of agile requirements can conflict with the detailed, formal approach of traditional methods. To bridge this gap, Gupta et al. propose a method for integrating conceptual models into agile projects by automatically generating them from user stories [29].
- **Stakeholder Involvement:** Agile’s need for continuous interaction differs from traditional structured engagement, leading to role confusion. Educating stakeholders in traditional methods on agile practices and emphasizing active participation helps improve collaboration and alignment.

### 3.3 Techniques for structuring agile documentation

While natural language is widely used for documentation [30], some have considered adopting structured or semi-structured templates to minimize ambiguities and enhance clarity. Some studies discuss structuring documentation produced in agile, particularly the backlog and user stories, but their objectives differ from ours as this structuring is not tailored to traditional methods. Mahmud and Veneziano introduce the use of mind-mapping as a technique for enhancing SRE, particularly in the process of creating a backlog. This approach has been shown to improve the quality of the backlog, with a case study using function points as a performance measure. However, it’s important to note that the article lacks specific details on how mind-mapping was applied [31].

Gupta et al. address challenges in agile requirements engineering and introduce a method for generating conceptual models from user stories. These models, which include use case, domain, state machine, and process models, enhance communication and understanding within agile projects. The paper emphasizes the importance of seamless integration and automation to prevent additional burdens on agile teams and outlines conditions for model adoption, proposing the use of NLP techniques for automation [29].

### 3.4 Metamodeling approach

The metamodel has been explored in various studies, both general and domain specific, focused on RE. In the existing literature, the development of metamodels for SRE has been a significant concern for providing practical support.

Koutsopoulos et al. [32] explore the differences between plan-driven and agile approaches to SRE, highlighting the strengths and weaknesses of each. They conceptualize the main concepts of the two approaches through individuals metamodels, then design an integrated metamodel to understand how they relate to each other. The integrated metamodel helps make differences of the two approaches and groups together the different methods being used in a project.

The use of the integrated meta-model was demonstrated through illustrative examples to showcase its practical application and effectiveness in combining agile and plan-driven approaches.

Schön et al. introduces a metamodel designed to describe key concepts within ARE, promoting a common understanding of this complex research field. Through case studies in Scrum and Kanban environments, the authors demonstrate how to instantiate the metamodel to develop concrete process models. Their contribution to the software development body of knowledge includes providing a metamodel for ARE, which holds implications for both researchers and practitioners. Researchers can utilize the metamodel to design new value-driven process models while practitioners can evaluate and enhance existing ones using the metamodel as a guide [23].

### 3.5 Summary of the related works

In summary, our review of the state of the art reveals distinct differences in the documentation processes between agile and traditional methods. Since the hybrid approach aims to combine the advantages of both, it should also integrate the strengths of their documentation practices. We have selected several criteria to analyse the state of documentation across agile and traditional methods and then explain how it should be in hybrid approach so that both teams can benefit from it, as shown in Table 2.

**Table 2.** Comparing documentation in agile, traditional and hybrid approaches

| Criteria                      | Agile   | Traditional  | Hybrid  |
|-------------------------------|---|--|---|
| Level of detail               | Minimal   | Detailed   | Evolving documentation  |
| Structure                     | Several artefacts   | One structured document  | One structured document   |
| Clarity and understandability | The short content is easy to understand but its structure is likely to cause problems | The large content is hard to understand despite its clear and easy structure                     | Easy structure with simple content                                    |
| Collaboration                 | Collaborative (Strongly support collaboration)  | Centralized (There are specific roles designated for creating and maintaining the documentation) | Role-Based (Everyone knows what information they are responsible for) |

We also note that the metamodeling has been used for specifying agile and traditional methods, but not as a technique for structuring agile documentation like those mentioned above. Additionally, it has been employed to map ARE and TRE processes, supporting their separate or combined use. In this paper, we have combined these elements to provide a metamodel-based solution, called ARDocS, that not only specifies and describes ARE and TRE at different levels of abstraction, but also serves as a technique for structuring agile documentation in a traditional manner, fostering hybrid approaches.

It should be noted that the cited methods for structuring agile documentation have different objectives than ARDocS. Mind-mapping focuses on backlogs, and conceptual models center on user stories, but both are limited to specific agile

artefacts and do not produce formal, structured documentation aligned with traditional templates. They also don't address collaboration between agile and traditional teams with differing documentation needs. In contrast, ARDocS considers all types of agile artefacts and translates them into structured documents compatible with various traditional templates. It supports both lightweight documentation for agile processes and formal outputs for traditional or regulatory contexts, encouraging smoother communication and reducing potential conflicts in hybrid environments.

#### 4. ARDOCS: AGILE REQUIREMENT DOCUMENTATION STRUCTURING

The findings of this research are intended for an agile team that, after documenting its requirements as it usually does by producing several distinct elements, will be able to assemble them into a more structured documentation in the traditional way. This is useful for hybrid projects, but also for large-scale projects with large teams and a high level of criticality. This chapter will explore the significance of organizing agile documentation and provide guidance on the process. This section is divided into three parts: "Solution context" explains the problem and need for solving it, "Objective and overview" defines the goals, and the other subsections outline our approach and methodology.

##### 4.1 Solution context

The agile process is continuous and iterative, as is its SRE process. In scrum for example, there is a correspondence between the ceremonies and ARE activities. Indeed, each ARE activity is associated with its corresponding Scrum ceremony. Documentation, on the other hand, is needed several times in a single iteration and can be applied in parallel with the other activities [33]. This leads the team to generate various artefacts tied to software requirements. Some of them are discussed verbally among team members, while others are documented. The agile team's documentation through distinct artefacts results in unstructured documentation, leading to a lack of standardization and uniformity. This creates difficulties related to clarity, traceability, and consistency.

Accordingly, there is a real need for structured documentation as part of the agile approach. While the informal and unstructured documentation approach might prove functional and efficient in certain small-scale projects, there are other cases where it can cause problems and lead the project to failure. The first case is when organisations use hybrid approaches which means that they combine both agile and traditional methods and this can be used particularly in complex, critical, or regulated project environments involving large teams and diverse stakeholder needs [26]. Gill et al. [34] state that large-scale projects often face challenges in meeting stakeholders' expectations, and there is a need to address these challenges by integrating both agile and non-agile elements to create hybrid adaptive methodologies [35].

Wagenaar et al. also point out that despite the fact that the use of documentation in agile software development (ASD) is perceived as "old-fashioned", recent research reveals a combination of traditional software development methods and ASD in hybrid approaches [9]. Among the different combinations, Scrum, the classic waterfall model and V-processes represent the majority [36, 37]. Kuhrmann et al. also

found similar results in their survey, with around 75% of participants deliberately mixing different development approaches. Typically, hybrid development is used intentionally when organisations feel that the agile approach is suitable for certain scenarios, while a more traditional approach is preferred in other situations [36].

In situations like these, where hybrid solutions are needed, traditional teams need to understand agile practices in order to collaborate effectively and to facilitate their transition [38].

A specific type of hybrid development involves agile and traditional teams working together on the same project. In such cases, it is important to avoid duplicating documentation in two different styles, as this would be inefficient. However, traditional teams may find agile artefacts unfamiliar or lacking in detail, which can result in confusion and misalignment. This disconnect can slow collaboration and reduce overall efficiency.

In the second case, during the transition from traditional to agile methodologies, organisations often adopt both agile and traditional practices simultaneously. This dual approach may be temporary during the transition from a traditional to a more agile method, or it may be a deliberate choice.

In the last case, customers sometimes require more tailored documentation than is typically produced as part of an agile process. This documentation may be required by law, to ensure clearer communication or to guarantee compliance.

We can conclude that there should be a common referential between the two development methods, and this is achieved naturally through a comprehensive documentation.

##### 4.2 Objective and overview

The idea behind this research work is to create a bridge between agile and traditional teams through documentation. The idea is to structure the various pieces of documentation created by the agile team during the development process into a single organized document. This solution is divided into three main components, as shown in Figure 1.

Agile structure component: It corresponds to the source model, and represents the structure that assembles the fragments of information collected during ARE activities.

- Traditional structure component: It corresponds to the target model and represents the structure of the final documentation. Here, source model components are structured on the basis of a predefined template familiar to the traditional team.

- Mapping component: It establishes the link between the elements of the source model and those of the target model, offering a projection of agile documentation artefacts into structured documentation.

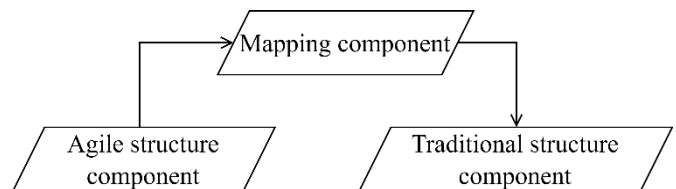


Figure 1. Solution components for structuring documentation

##### 4.3 Metamodeling levels

The suggested approach relies on metamodeling. Table 3 provided below shows the four levels of modelling, as

specified by the Object Management Group (OMG), and how they are applied in the context of this research.

**Table 3.** The four conceptual levels in agile/traditional documentation modeling

| Levels    | Agile   | Traditional   |
|-----------|---|---|
| <b>M3</b> | Agile Meta-metamodel<br>(Referred to as M <sup>3</sup> A)               | Traditional Meta-metamodel (Referred to as M <sup>3</sup> T)                    |
| <b>M2</b> | Metamodel (Scrum, XP, Kanban, ...)<br>(Referred to as M <sup>2</sup> A) | Metamodel ( <i>VOLERE</i> , IEEE 830, ...)<br>(Referred to as M <sup>2</sup> T) |
| <b>M1</b> | Scrum documentation model   | <i>VOLERE</i> template  |
| <b>M0</b> | Scrum documentation artefact instances                                  | <i>VOLERE</i> instances   |

At M3 level, we model the agile and traditional concepts that have an impact on documentation and are related to software requirements. The chosen concepts are represented at a high-level of abstraction, referred to as meta-metamodeling. At this level, the definition must be sufficiently generic to specify the concepts and relationships used to define the metamodels at level M2. In the context of agile methodology, for example, it should be possible to represent documentation concepts independently of the agile method employed. Similarly, on the traditional side, it should represent documentation concepts regardless of the template chosen. The aim of the M3 level is to enable the comparison and alignment of two languages represented by metamodels at M2 level, corresponding to the agile method chosen and the selected traditional documentation template.

M2 level represents the concepts of a selected agile method, as well as the concepts of a chosen traditional documentation template. The metamodel designed at this level defines the structure and semantics of the models created at M1 level.

M1 level represents the different types of agile documentation resulting from the chosen method as well as the selected template from the traditional side.

Finally, M0 level represents the actual instances of agile documents that we want to structure, as well as the instances that correspond to them in the traditional.

#### 4.4 The mapping process

In ARDocS, the structuring of agile documentation is achieved through a mapping process that translates a set of

agile artefacts into a structured document in line with the traditional format. To avoid mapping for each agile project, we have opted to establish it at a meta-level between the concepts of M<sup>3</sup>A and M<sup>3</sup>T. This not only standardizes the process but also makes it easier, as the meta-level significantly reduces the number of concepts involved. In this strategy, the mapping is done in four stages, as described in Figure 2.

(1) Instantiating the agile requirements documentation meta-metamodel (M<sup>3</sup>A) creates the agile requirements documentation metamodel (M<sup>2</sup>A). (M<sup>3</sup>A) represents the concepts from which we can instantiate any agile metamodel. This instantiated metamodel can represent one of the agile methods among which we can cite Scrum, Kanban or XP metamodel.

(2) Mapping the agile documentation concepts to the ones of traditional documentation. This mapping is also represented using a metamodel.

(3) The instantiation of the traditional meta-metamodel (M<sup>3</sup>T) provides a metamodel that describes a specific template (M<sup>2</sup>T). This metamodel specifies all the structural elements of the template and the relationships between them.

(4) The final step consists of choosing the concepts of (M<sup>2</sup>T) that correspond to those of (M<sup>2</sup>A). In other words, selecting the right elements of the traditional template that reflect the elements of the chosen agile method.

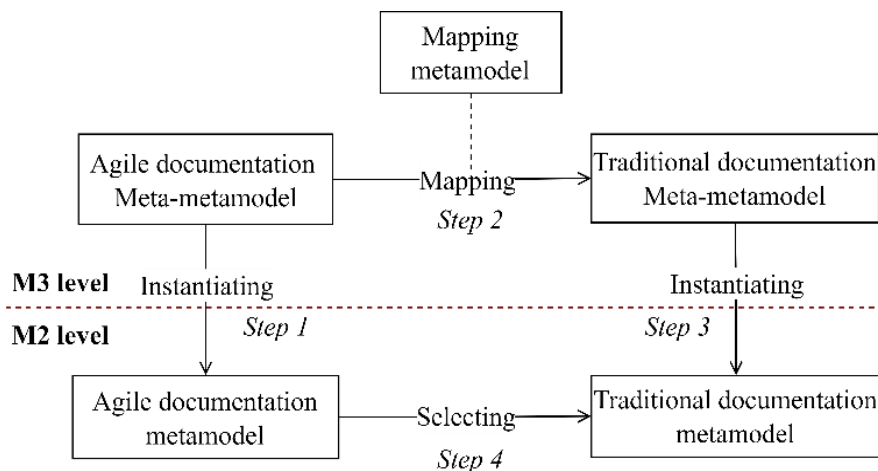
#### 4.5 Structuring documentation process

In Figure 3 we use the three levels of abstraction to describe the process of structuring documentation.

At M1 level, when the agile team members participate in an ARE activity, they automatically produce information that must be documented. These collected elements of information are identified as “chunks”. The chunks are grouped in a structure that represents the source model of ARDocS.

The target model on the other side represents a traditional documentation template. At this stage, an existing traditional documentation template will be used.

Then, in order to align the agile documentation structure (source model) with the traditional one (target model), we carry out a mapping between their respective terms and concepts using the process of metamodeling explained in section 4.4. In this study, we assume the role of agile and traditional methodologists in modelling meta-models and meta-metamodels at the M3 and M2 levels. As researchers, we propose these two models as part of our research work.



**Figure 2.** Overview of the mapping process

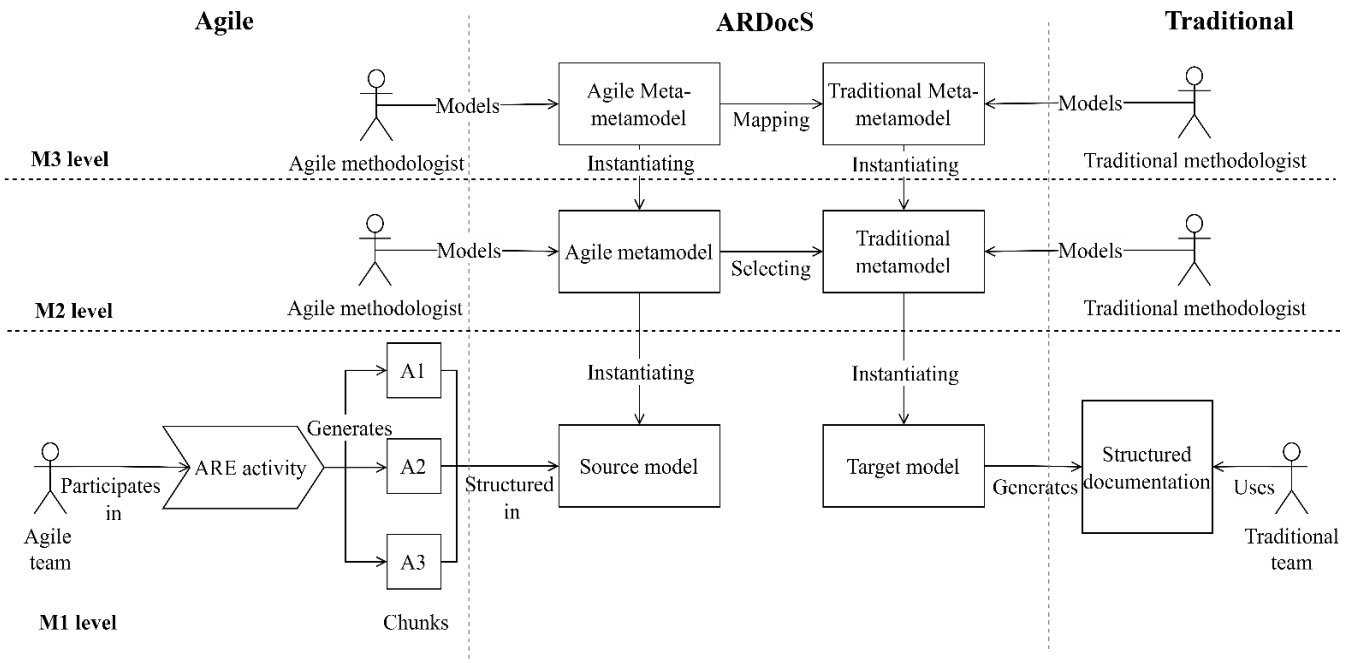


Figure 3. ARDocS architecture

## 5. THE MAPPING PROCESS

In this section, we will explore in detail the four steps described in Figure 2. The first subsection describes ARDocS at the M3 level, which includes the modeling of both agile and traditional meta-metamodels and the mapping between their concepts. The second subsection describes ARDocS at the M2 level, which involves the instantiation of M<sup>3</sup>A and M<sup>3</sup>T, namely the modeling of agile and traditional metamodels, and finally the selection of the appropriate concepts.

### 5.1 The mapping process at M3 level

In this section we define the meta-metamodels for both agile and traditional requirements engineering, referred to as M<sup>3</sup>A and M<sup>3</sup>T, respectively. Then we define the mapping between their concepts. The development of these two meta-metamodels enabled the identification of similarities and related concepts, along with their integration into one.

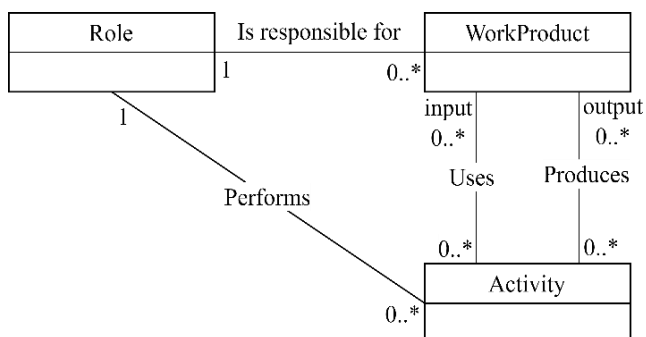


Figure 4. Conceptual model of software process engineering metamodel SPEM

Here, we adopt the Software Process Engineering Metamodel (SPEM) introduced by the OMG [39], which is used to describe software development processes. There are two versions of SPEM, each comprising several concepts.

However, we will focus only on the three main concepts: The role who is responsible for a product and performs activities that use and produce products. As shown in Figure 4, these three components are integrated into the SPEM conceptual model, providing an overview of how SPEM works in general.

#### 5.1.1 Meta-metamodel for ARE documentation: M<sup>3</sup>A

As shown in Figure 5, M<sup>3</sup>A defines requirements documentation in agile context at a high level of abstraction. It is built around the three SPEM concepts with an agile specification using the A annotation: A.Role, A.Activity and A.Documentation (Corresponds to work product in SPEM); these are considered from a documentation perspective.

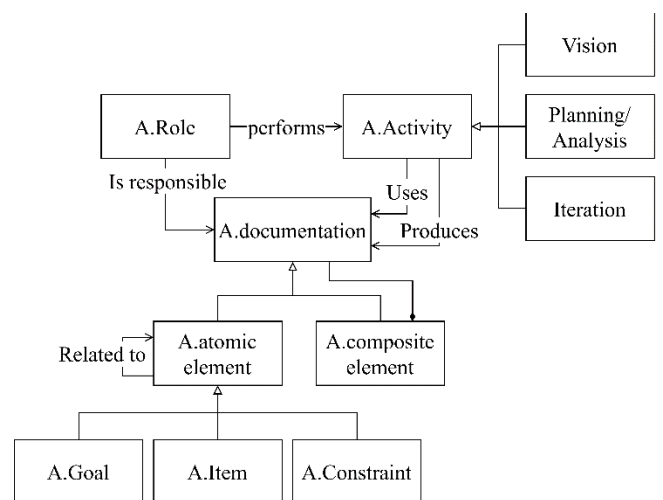


Figure 5. ARE documentation Meta-metamodel (M<sup>3</sup>A)

The M<sup>3</sup>A allows for describing documentation across two dimensions: process and product. (i) The process dimension is described through the concepts of role and activity. It describes who is responsible for each documentation chunk and what activities produce or use them. In addition, agile activities are divided into three main types found in the most

common agile methods: The vision is the stage of exploring and understanding the problem, followed by planning and analysis, which consists of deciding what to do to solve the stated problem. The last stage is the iteration, which represents managed intervals of time, of short or long duration depending on the agile method chosen, which bring together a set of

activities designed to produce value. (ii) The product dimension describes documentation product concept and its components. At the atomic level, components can be of several types, including goal, constraint and item. The M<sup>3</sup>A concepts are described individually in Table 4.

**Table 4.** M3A concepts description

| M <sup>3</sup> A Concepts | Description   |
|---------------------------|---|
| A.Activity                | Agile activities related to the whole development process, allowing to use and produce documentation chunks.  |
| Vision                    | This phase involves establishing a clear understanding of the project vision and goals.   |
| Planning/Analysis         | In this phase, the team plans and analyses the work required to achieve the project vision.   |
| Iteration                 | During this phase, the team executes the work in iterative cycles, focusing on delivering value incrementally. At least one iteration is completed during this phase. |
| A.Role                    | Represents both the agile team and the stakeholders involved in the project.  |
| A.Documentation           | It groups together the essential elements found in requirements documentation (The chunks).   |
| A.atomic element          | Represents an individual piece of information that describes either the final product or the process of creating it.  |
| A.Composite element       | Integrate several atomic elements.  |
| A.Goal                    | Objectives established by involved roles in the project, which must be accomplished to deliver the desired product.   |
| A.Item                    | Elements necessary to describe the final product to be developed.   |
| A.Constraint              | All restrictions on the way the product is produced.  |

5.1.2 Meta-metamodel for TRE documentation: M<sup>3</sup>T

As shown in Figure 6 and based on the same logic as in agile, two dimensions have been used to describe traditional documentation: process and product. (i) The process dimension is described through T.role and T.activity. The activity concept in M<sup>3</sup>T refers to activities of TRE process. Batra and Bhatnagar [40] conducted a comparative analysis of various requirements engineering process models. Of particular interest was the study by Loucopoulos and Karakostas [41], which categorizes the process into three key activities: Elicitation, Specification, and Validation. These activities encompass tasks performed across multiple traditional processes simultaneously. (ii) The product aspect represented by T.requirement specification document (Which corresponds to the work product in SPEM) gathers together the elements of the documentation and their components. The most frequently used documentation elements are Purpose or Goal, Requirement and Scenario, which are also in accordance with the concepts defined in the SWORE ontology - SoftWiki Ontology for Requirements Engineering [42]. The addition of the concept of constraint was considered important, since it is included in the majority of existing models. The M<sup>3</sup>T concepts are described individually in Table 5.

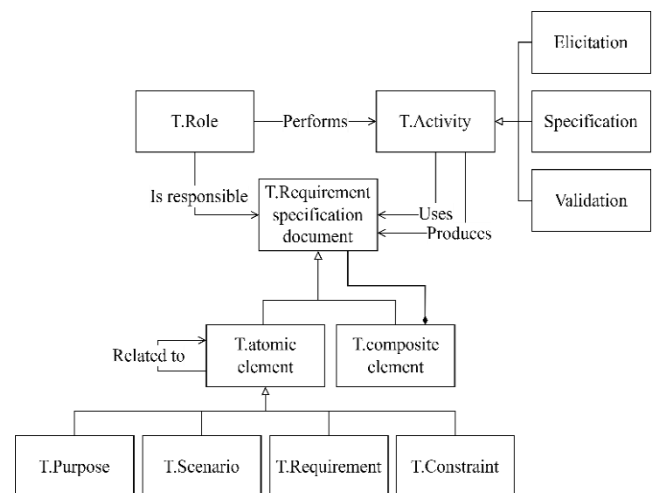
performed differently, and in M<sup>3</sup>A, the entire development process is represented, unlike in M<sup>3</sup>T, where only the RE activities are included.

The selection of concepts in the mapping was made as follows:

- *A.Goal corresponds to T.Purpose:* Both concepts define the "why" of a project or iteration. A goal in agile and a purpose in traditional methods represent what guides the work to achieve the business objectives.
- *A.Item corresponds to T.Requirement/T.Scenario:* Agile items represent achievable work focused on value creation. They are aligned with requirements or scenarios, as both articulate what needs to be accomplished.
- *A.Constraint corresponds to T.Constraint:* In both methods, constraints establish the limits that the project or product must respect. These limits are essential for decision-making and planning.
- *A.Role corresponds to T.Role:* Roles are essential for defining responsibilities in any methodology. Although the implementation differs, both methods rely on roles to ensure the successful execution of tasks.

5.1.3 Mapping agile to traditional concepts: Step 2

After modeling the two meta-metamodels: M<sup>3</sup>A and M<sup>3</sup>T, the next step (which corresponds to step 2 in Figure 2) is to establish a mapping between their concepts, which will allow to identify the equivalent of each agile documentation element in the traditional template. This mapping is modeled in the following format: "agile concept: traditional concept" as illustrated in Figure 7. It should be noted that, even if there is a correspondence between all agile and traditional concepts there is still a difference in the level of detail handled and documented between the two methods. Traditional methods generally involve more extensive detail, aligning with their characteristic plan-driven development approach. In contrast, agile methodologies, in line with the third value of the agile manifesto, prioritise working software over comprehensive documentation [6]. Furthermore, there is no correspondence between the activities because the two processes are

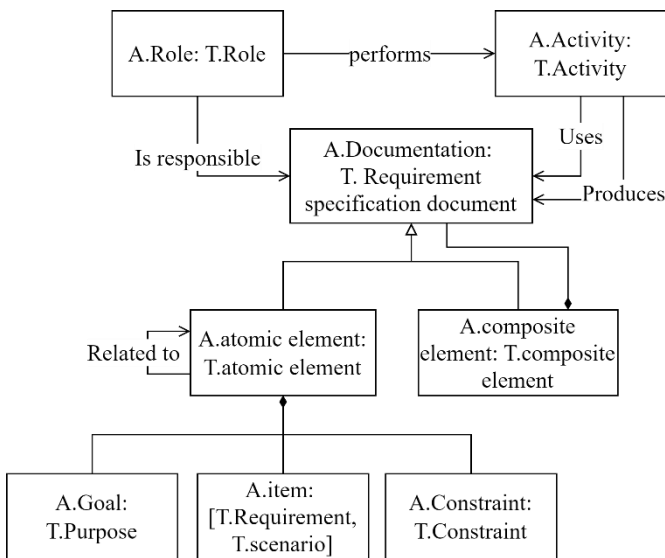


**Figure 6.** TRE documentation Meta-metamodel (M<sup>3</sup>T)



**Table 5.** M3T concepts description

| M <sup>3</sup> T Concepts            | Description  |
|--------------------------------------|--|
| T.Activity                           | TRE activities that produce or use requirements.   |
| Elicitation                          | Refers to gathering the requirements of the system from different stakeholders.                                      |
| Specification                        | The activity of documenting requirements.  |
| Validation                           | Checks that the requirements accurately represent the needs of the system.   |
| T.Role                               | Individuals with an interest in the product, who either have requirements for it or contribute to its development.   |
| T.Requirement specification document | A complete collection of requirements knowledge for a specific project.  |
| T.atomic element                     | Represents an individual piece of information that describes either the final product or the process of creating it. |
| T.composite element                  | Integrate several atomic items.  |
| T.Purpose                            | Intended result to be achieved by the system.  |
| T.Scenario                           | Descriptions of the usage of the planned system to reach a defined goal.   |
| T.Requirement                        | What the product must do, or a property that the product must have.  |
| T.Constraint                         | Restrictions on the product or the way it is produced.   |



**Figure 7.** Mapping metamodel: M3 level

**5.2 The mapping process at M2 level**

In this section, we see the connection between the concepts at M3 and M2 levels, examining how they were instantiated and modeled at the M2 level for both the agile and traditional sides. This corresponds to step 1 and step 3 in Figure 2.

**5.2.1 ARE process**

By instantiating the M<sup>3</sup>A, we obtain the agile metamodel M<sup>2</sup>A. This instantiation is based on a single chosen agile method. The M<sup>2</sup>A is presented in two parts: the first shows how each concept was derived from the M<sup>3</sup>A and the second explains the different concepts and how they are related to each other.

**Instantiation of M<sup>3</sup>A (Step 1)** In step 1, to instantiate M<sup>3</sup>A, we opted for a single agile method, selecting Scrum due to its widespread popularity. We mapped the relevant Scrum concepts to each M<sup>3</sup>A concept, using the latest version of the Scrum Guide [43]. The instantiation process of M<sup>3</sup>A was carried out based on the semantics of the concepts. Due to the generic nature of M<sup>3</sup>A, the instantiation was a smooth and straightforward process, leading to a logical and coherent outcome. The instantiation of M<sup>3</sup>A is presented in Table 6, and below, we explain how it is applied to Scrum artefacts:

- Agile Goal is instantiated into two key goals in Scrum:

- Product Goal and Sprint Goal, reflecting both long-term product vision and short-term sprint objectives.
- Agile Item is instantiated into Theme, Epic, Functionality, Benefit, and Task, representing different levels of work, from high-level strategic objectives to specific tasks that enable daily progress.
- Agile Constraint is instantiated into Project Estimation, Story Points, Acceptance Criteria, and Definition of Done, providing measurable limits and conditions that ensure work quality.

**Agile metamodel: (Scrum)** The artefacts included in this metamodel are those most commonly found in an agile project, particularly when using the Scrum framework. The metamodel shown in Figure 8 brings together the scrum concepts, focusing on those related to documentation. The Concepts in yellow represent A.Role, those in blue are A.Activity and those in green are A.Documentation.

**Table 6.** Instantiating M<sup>3</sup>A concepts

| M <sup>3</sup> A Concept | Scrum Concept  |
|--------------------------|--|
| A.Role                   | Scrum team, Stakeholders, Persona  |
| Vision                   | Envisioning  |
| Planning                 | Product backlog refinement, Release planning   |
| Iteration                | Sprint   |
| A.Composite item         | Product backlog (PB), Sprint backlog (SB), User story (US)                               |
| A.Goal                   | Product goal, Sprint goal  |
| A.Item                   | Theme, Epic, Functionality, Benefit, Task  |
| A.Constraint             | Project estimation, Story point (SP), Acceptance criteria (AC), Definition of done (DOD) |

**5.2.2 TRE process**

By instantiating the M<sup>3</sup>T, we obtain the traditional metamodel M<sup>2</sup>T. As with the work done in agile, this instantiation is based on a single traditional documentation model. Also, the M<sup>2</sup>T is presented in two parts: the first illustrates how each concept was derived from the M<sup>3</sup>T and the second explains the different concepts and how they are related to each other.

**Instantiation of M3T (Step 3):** In step 3, we have chosen the VOLERE Requirements Specification Template to instantiate the M<sup>3</sup>T [44]. VOLERE was established by Suzanne and James Robertson and is described in their book “Mastering the Requirements Process: Getting Requirements Right,” which was first published in 1999 [45]. While the book outlines a process for successfully discovering, verifying, and

documenting requirements, our focus in this research is solely on the *VOLERE* template itself. The *VOLERE* template is included in the book as a result of the *VOLERE* process. In order to respect intellectual property rights and use the template appropriately, we reached out to the book's authors, who kindly provided us with the most recent version of the template (Edition 20-2020). The template is protected by copyright, and our use of it is restricted to academic purposes as per the authors' guidelines. We selected the *VOLERE* template because it is rich and comprehensive, covering a wide range of requirements concepts, including those defined in the IEEE 830 standard. The instantiation is shown in Table 7, and below, we explain how it is applied to the *VOLERE* template elements:

- T.Purpose is instantiated as the Project Goal, representing the reasons for doing the project.
- T.Scenario is instantiated into Business Scenario and Product Scenario, capturing the different contexts or situations the project aims to address, from business needs to product-specific scenarios.
- T.Requirement is instantiated into Functional Requirement and Non-functional Requirement, reflecting the functionalities the system must perform and the quality attributes it must satisfy.
- T.Constraint is instantiated into Solution Constraint and Project Constraint, defining both the limitations imposed on the solution itself and the boundaries within which the project must operate.

**Traditional metamodel:** *VOLERE* The M<sup>2</sup>T presented in Figure 9 consolidates the key concepts of *VOLERE*. It is

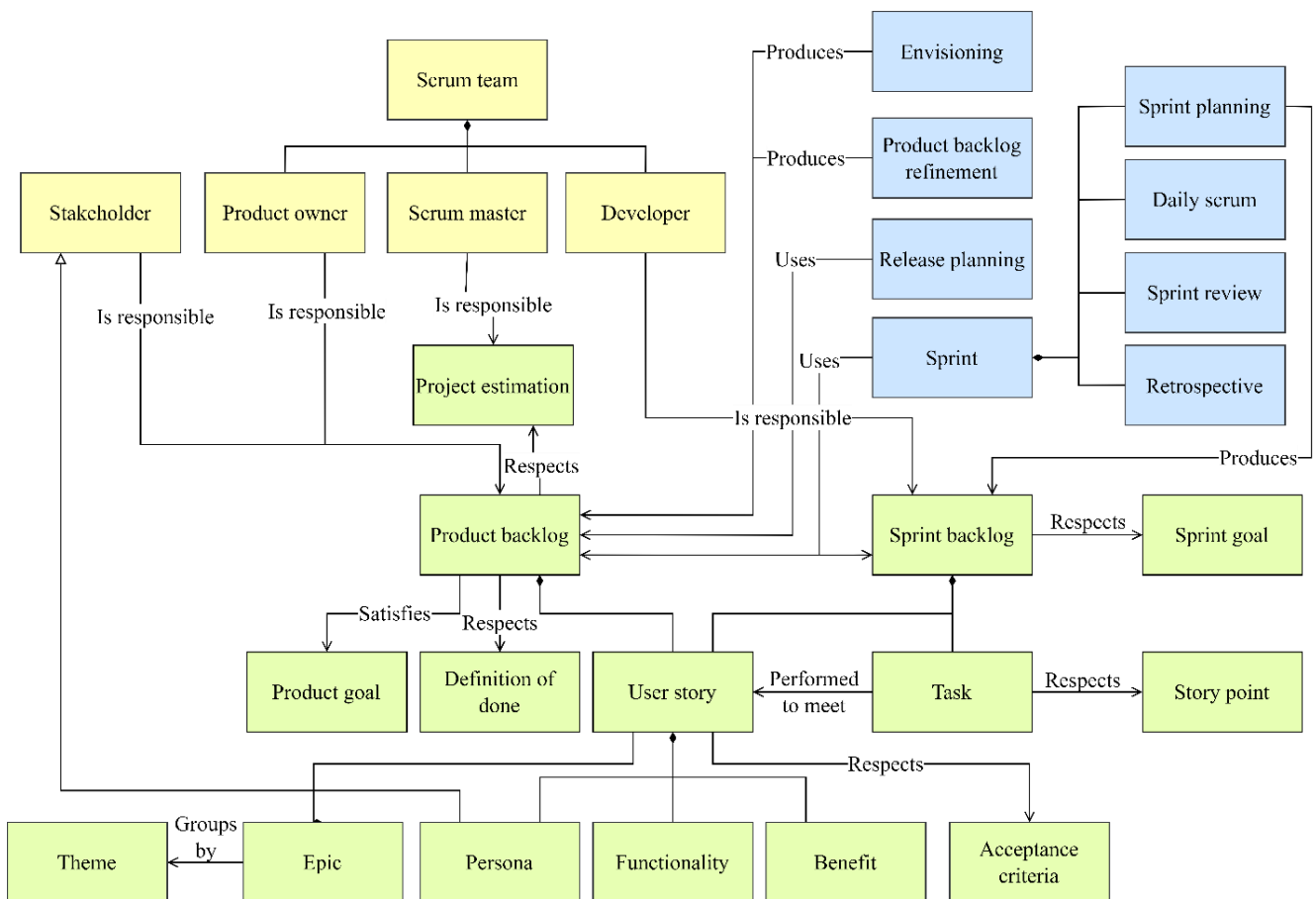
evident that traditional documentation entails more details compared to agile documentation. Consequently, as mentioned before, during the mapping process, some *VOLERE* concepts may not be utilized. The concepts in yellow represent T.Role, those in blue are T.Activities and those in green are T.Documentation.

### 5.2.3 Concepts selection: Step 4

The final step of ARDocS approach (which corresponds to step 4 in Figure 2) is selecting concepts from the traditional template (i.e. *VOLERE*) that match those of agile framework (i.e. Scrum). This selection is guided by the mapping conducted earlier, reducing the options for each concept in agile documentation. However, we have always had to make a choice since Scrum and *VOLERE* are both conceptually rich. Furthermore, this selection is a singular event and will serve as a reference for all practitioners employing Scrum, wanting to structure their documentation artefacts.

**Table 7.** Instantiating M<sup>3</sup>T concepts

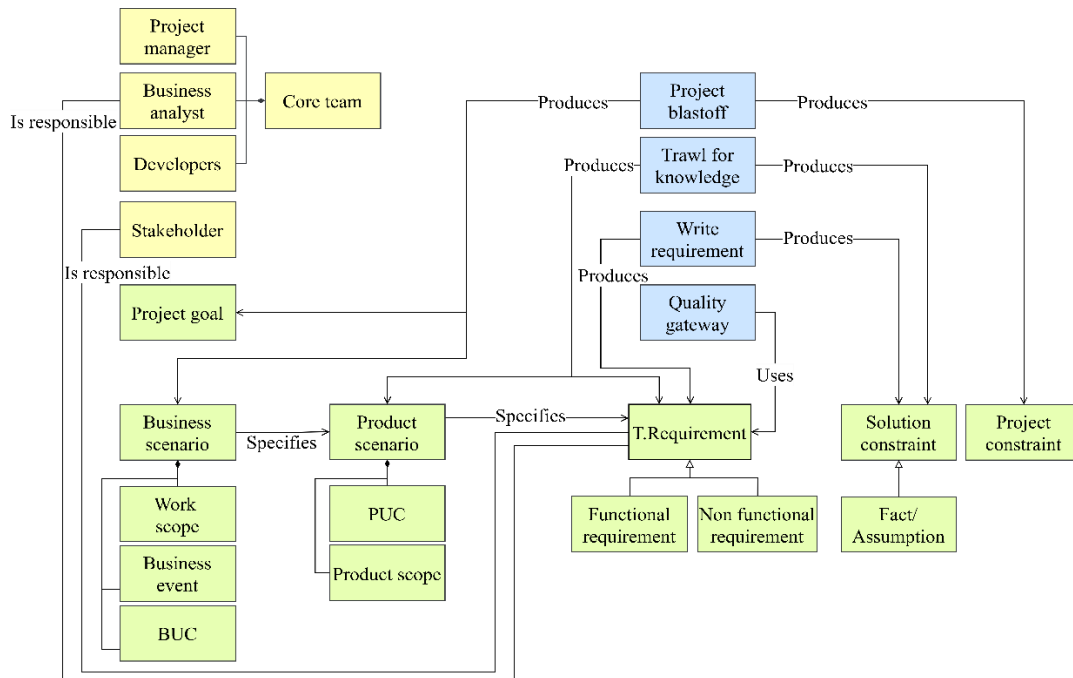
| M <sup>3</sup> T Concept | VOLERE Concept                                     |
|--------------------------|--|
| T.Role                   | Core team, Stakeholders, Persona                   |
| Elicitation              | Project blast off, Trawl for knowledge             |
| Specification            | Write requirements                                 |
| Validation               | Quality gateway                                    |
| T.Purpose                | Project goal                                       |
| T.Scenario               | Business scenario, Product scenario                |
| T.Requirement            | Functional requirement, Non-functional requirement |
| T.Constraint             | Solution constraint, Project constraint            |



**Figure 8.** Scrum metamodel (M2A)

**Table 8.** Selection step of ARDocS

| N° | SCRUM  | VOLERE   |
|----|--|--|
| 1  | Scrum team   | Core team  |
| 2  | Stakeholder  | Stakeholder  |
| 3  | Product goal   | Project goal                                       |
| 4  | Project estimation                                   | Project constraint                                 |
| 5  | Sprint goal  | Project goal                                       |
| 6  | Theme  | Business use case (BUC)                            |
| 7  | Epic   | Product use case (PUC)                             |
| 8  | Persona  | Stakeholder (Persona)                              |
| 9  | Functionality  | Functional requirement/ Non-functional requirement |
| 10 | Benefit  | Functional requirement (Rationale)                 |
| 11 | Definition of done, Acceptance criteria, Story point | Solution constraint                                |
| 12 | Task   | Tasks  |



**Figure 9.** VOLERE metamodel (M2T)

As shown in Table 8, during the selection process, we encountered three scenarios. The first scenario is when there is a direct link between the concepts, in which case no choice is imposed on us (see lines 3-5). The second scenario is when we have a choice to make, but it is obvious because the concepts in Scrum and *VOLERE* are similar (see lines 1-2-8-9-10-12). The third scenario is when the choice is unclear and the semantics of both sides differ. In this case, the choice is made through careful consideration and a thorough understanding of *VOLERE*, allowing us to select the concepts that are the most representative and the closest semantically to make our selection meaningful (see lines 4-6-7-11). It is worth noting that certain Scrum concepts, being generic, may not have direct equivalents; however, the projection is made for their components, as is the case with the product backlog.

## 6. VALIDATION: CASE STUDY

Requirements documentation is considered trade secrets by companies and is therefore confidential [46]. Consequently, finding User Stories and backlogs becomes challenging for the validation stage. To validate ARDocS approach we used an

existing case study which is an adaptation of a case from Yourdon and Argilla [47]. The case study is realistic and typical of medium-sized web application development projects, which means it can be generalised to similar cases. Bolloju et al. described their own version of the example which they used to validate their method [48]. They provide a description of the case study's project and a set of user stories. We completed the example by extracting other important agile artefacts from the project description to obtain documentation that is as concrete and realistic as that of a real project and that could be structured using ARDocS. Table 9, in the appendix, presents the agile documentation derived from the case study. Due to space constraints, we could not provide full descriptions of all documentation elements; but only extracts are shown. The selected artefacts are among the most common in agile documentation and have been chosen for their importance in the development process, but also for their diversity in terms of nature and level of abstraction. In this way, they enable the method to be tested correctly by covering the main concepts of the ARDocS mapping process shown in Figure 7. We're going to run through ARDocS approach on US2 and the elements linked to it: SP2, Theme2, Epic3, AC2, as shown in Table 10.

**Table 9.** Agile documentation artefacts

| Artefact               | Description  |
|------------------------|--|
| Product goal           | Build a web-based software system for efficiently managing the subscriptions, reviews, and publications of various journals.   |
| Project estimation     | <b>Sprint Length:</b> 2 weeks. <b>Total Time:</b> 26 weeks (13 Sprints). <b>Total Budget:</b> \$260,000  |
| Theme                  | <b>Theme1:</b> Subscription Management. <b>Theme2:</b> Article Review and Publication.   |
| Epic                   | <b>Epic1:</b> Manage Subscriptions. <b>Epic2:</b> Manage Article Submissions and Reviews. <b>Epic3:</b> Manage Editorial Workflow.   |
| User story+Story point | <b>US1:</b> As a Subscriber I want to Subscribe to the one or more journals so that I can receive the journal issues. (SP1: 5 points). <b>US2:</b> As an Editor I want to Reject an article without any further reviews so that irrelevant articles need not be assigned for review. (SP2: 3 points) |
| Definition of done     | All acceptance criteria of the user stories are met.   |
| Acceptance criteria    | <b>AC1:</b> The subscriber can select one or more journals to subscribe to. <b>AC2:</b> Upon rejection, the article’s status is updated to “Rejected” and no further review steps are triggered.   |

**Table 10.** ARDocS application

| Levels/Chunks | M <sup>2</sup> A    | M <sup>3</sup> A | M <sup>3</sup> T | M <sup>2</sup> T                       |
|---------------|---------------------|------------------|------------------|--|
| <b>Theme2</b> | Theme               | A.item           | T.scenario       | BUC                                    |
| <b>Epic3</b>  | Epic                | A.item           | T.scenario       | PUC                                    |
|               | Persona             | A.role           | T.role           | Persona                                |
| <b>US2</b>    | functionality       | A.item           | T.requirement    | Functional/ Non-functional requirement |
|               | Benefit             | A.item           | T.requirement    | Rationale                              |
| <b>SP2</b>    | Story point         | A.constraint     | T.constraint     | Solution constraint                    |
| <b>AC2</b>    | Acceptance criteria | A.constraint     | T.constraint     | Solution constraint                    |

**7. DISCUSSION AND CONCLUSION**

The case study used here is just a demonstration of the application of ARDocS. It’s clear that the benefits of ARDocS will be seen more clearly when applied to complete documentation. Currently, most agile teams use digital tools to manage and document their projects. The most popular tool is Confluence by Atlassian, which is dedicated to documentation and offers a wide choice of templates such as the Product Requirements Document (PRD) for user stories, the Project Kickoff for project objectives and estimations, to-do lists for tasks and so on.

According to a recent study by Atlassian, there are currently over 70 templates for defining all kinds of information, including software requirements. ARDocS would bring together all the information dispersed in the various agile templates to create a structured documentation that could be used by both traditional and agile teams. This would reduce the complexity of use and the repetition of concepts. However, when applying ARDocS approach, teams are faced with two scenarios due to the iterative nature of agile: 1) Generating a single document at the end of the project from the artefacts produced during the SRE activities, specifying the iterations. 2) Generating documentation increments at the end of each iteration. These scenarios can be adapted to the needs of the team using them.

In conclusion, the aim of ARDocS is to conceptualize agile and traditional documentation at different levels of abstraction. It also serves as a means of structuring agile documentation, which is particularly useful in the context of a project developed in hybrid approach. ARDocS can therefore be seen as a tool of communication between agile and traditional teams, or among multiple agile teams using different methods with different documentation artefacts. This approach can be enhanced by integrating it with tools used by agile teams, such as Confluence, which offers the possibility of extending it by creating new plugins. To achieve this, we propose developing

a Confluence plugin that will retrieve relevant content from Jira (for example, user stories and priorities) and fill in predefined sections of a custom Confluence template based on VOLERE. An important challenge is ensuring real-time synchronization between agile artefacts and the generated document, as agile teams frequently update artefacts that must be immediately reflected in the traditional document. To address this, we suggest that the plugin will incorporate a trigger that periodically checks for changes in agile artefacts and updates the Confluence document accordingly. Automating ARDocS in this way would streamline its use, making the documentation process more efficient and accessible.

**REFERENCES**

- [1] Garscha, P. (2021). From sustainability in requirements engineering to a sustainability-aware scrum framework. In 2021 IEEE 29th International Requirements Engineering Conference (RE), Notre Dame, IN, USA, pp. 462-467. <http://doi.org/10.1109/RE51729.2021.00069>
- [2] Martins, L.E.G., Gorschek, T. (2016). Requirements engineering for safety-critical systems: A systematic literature review. Information and Software Technology, 75: 71-89. <https://doi.org/10.1016/j.infsof.2016.04.002>
- [3] Pang, C.Y. (2020). Product backlog and requirements engineering for enterprise application development. In: Software Engineering for Agile Application Development. IGI Global, pp. 1-29. <http://dx.doi.org/10.4018/978-1-7998-2531-9.ch001>
- [4] Hull, E., Jackson, K., Dick, J. (2011). Requirements Engineering (3rd ed.). Springer. <https://doi.org/10.1007/978-1-84996-405-0>
- [5] Alhazmi, A., Huang, S. (2020). Survey on differences of requirements engineering for traditional and agile development processes. Proc. 2020 SoutheastCon, IEEE,

- Raleigh, NC, USA, pp. 1-9. <https://doi.org/10.1109/SoutheastCon44009.2020.9397492>
- [6] Fowler, M., Highsmith, J. (2001). The agile manifesto. *Software Development*, 9(8): 28-35.
- [7] Hadar, I., Sherman, S., Hadar, E., Harrison, J.J. (2013). Less is more: Architecture documentation for agile development. In 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), San Francisco, CA, USA, pp. 121-124. <http://doi.org/10.1109/CHASE.2013.6614746>
- [8] Stettina, C.J., Heijstek, W. (2011). Necessary and neglected? An empirical study of internal documentation in agile software development teams. In Proceedings of the 29th ACM International Conference on Design of Communication, Italy, pp. 159-166. <https://doi.org/10.1145/2038476.2038509>
- [9] Wagenaar, G., Overbeek, S., Lucassen, G., Brinkkemper, S., Schneider, K. (2018). Working software over comprehensive documentation-Rationales of agile teams for artefacts usage. *Journal of Software Engineering Research and Development*, 6: 1-23. <http://dx.doi.org/10.1186/s40411-018-0051-7>
- [10] Rigby, D., Sutherland, J., Takeuchi, H. (2016). Embracing Agile. *Harvard Business Review*. <https://hbr.org/2016/05/embracing-agile>.
- [11] Brennan, K. (2009). A guide to the business analysis body of knowledge. IIBA. [https://www.academia.edu/6555031/A\\_Guide\\_to\\_the\\_Business\\_Analysis\\_Body\\_of\\_Knowledge\\_BABOK\\_Guide\\_Version\\_2\\_0](https://www.academia.edu/6555031/A_Guide_to_the_Business_Analysis_Body_of_Knowledge_BABOK_Guide_Version_2_0).
- [12] Sommerville, I., Sawyer, P. (1997). Requirements engineering: A good practice guide. John Wiley and Sons, Inc. <https://dl.acm.org/doi/10.5555/549198>.
- [13] Paetsch, F., Eberlein, A., Maurer, F. (2003). Requirements engineering and agile software development. In WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Linz, Austria, pp. 308-313. <https://doi.org/10.1109/ENABL.2003.1231428>
- [14] Kotonya, G., Sommerville, I. (1996). Requirements engineering with viewpoints. *Software Engineering Journal*, 11(1): 5-18. <http://doi.org/10.1049/sej.1996.0002>
- [15] Alsayed, A.O., Bilgrami, A.L., Foster, W.A. (2017). Improving software quality management: testing, review, inspection and walkthrough. *International Journal of Latest Research in Science and Technology*, 6(1): 1-12.
- [16] Atoum, I., Baklizi, M.K., Alsmadi, I., Otoom, A.A., Alhersh, T., Ababneh, J., Almalki, J., Alshahrani, S.M. (2021). Challenges of software requirements quality assurance and validation: A systematic literature review. *Proc. 2021 IEEE Access*, 9: 137613-137634. <http://dx.doi.org/10.1109/ACCESS.2021.3117989>
- [17] Scukanec, S.J., van Gaasbeek, J.R. (2010). A day in the life of a verification requirement. In *INCOSE International Symposium*, 20(1): 2524-2542. <https://doi.org/10.1002/j.2334-5837.2010.tb01180.x>
- [18] Elshandidy, H., Mazen, S. (2013). Agile and traditional requirements engineering: A survey. *Proc. Int. Journal of Scientific & Engineering Research*, 4(9): 473-482. <http://dx.doi.org/10.14299/ijser.2013.09.002>
- [19] Inayat, I., Salim, S.S., Marczak, S., Daneva, M., Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*, 51: 915-929. <http://dx.doi.org/10.1016/j.chb.2014.10.046>
- [20] Goetz, R. (2002). How agile processes can help in time-constrained requirements engineering. In Proceedings of the International Workshop on Time Constrained Requirements Engineering. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=aff0a3151a7c742d94fad1bbcbe04139af8509e0>.
- [21] Ramesh, B., Cao, L., Baskerville, R. (2010). Agile requirements engineering practices and challenges: An empirical study. *Information Systems Journal*, 20(5): 449-480. <http://dx.doi.org/10.1111/j.1365-2575.2007.00259.x>
- [22] Sutherland, J., Sutherland, J.J. (2014). Scrum: The art of doing twice the work in half the time. Crown Currency. [https://www.agileleanhouse.com/lib/lib/News/More\\_Praise\\_for\\_Scrum\\_The\\_Art\\_of\\_Doing\\_T.pdf](https://www.agileleanhouse.com/lib/lib/News/More_Praise_for_Scrum_The_Art_of_Doing_T.pdf).
- [23] Schön, E.M., Sedeño López, J., Mejías Risoto, M., Thomaschewski, J., Escalona Cuaresma, M.J. (2019). A metamodel for agile requirements engineering. *Journal of Computer and Communications*, 7(2): 1-22. <http://dx.doi.org/10.4236/jcc.2019.72001>
- [24] Júnior, P.S.S., Barcellos, M.P., Falbo, R. de A., Almeida, J.P.A. (2021). From a Scrum reference ontology to the integration of applications for data-driven software development. *IST*, 136: 106570. <https://doi.org/10.1016/j.infsof.2021.106570>
- [25] Batool, A., Motla, Y.H., Hamid, B., Asghar, S., Riaz, M., Mukhtar, M. (2013). Comparative study of traditional requirement engineering and agile requirement engineering. In 2013 15th International Conference on Advanced Communications Technology (ICACT), PyeongChang, Korea (South), pp. 1006-1014. <https://ieeexplore.ieee.org/abstract/document/6488350>.
- [26] Prenner, N., Unger-Windeler, C., Schneider, K. (2021). Goals and challenges in hybrid software development approaches. *Journal of Software: Evolution and Process*, 33(11): e2382. <http://doi.org/10.1002/smr.2382>
- [27] Hess, A., Diebold, P., Seyff, N. (2019). Understanding information needs of agile teams to improve requirements communication. *Journal of Industrial Information Integration*, 14: 3-15. <http://doi.org/10.1016/j.jii.2018.04.002>
- [28] Boehm, B., Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*, 22(5): 30-39. <http://doi.org/10.1109/MS.2005.129>
- [29] Gupta, A., Poels, G., Bera, P. (2022). Using conceptual models in agile software development: A possible solution to requirements engineering challenges in agile projects. *IEEE Access*, 10: 119745-119766. <http://doi.org/10.1109/ACCESS.2022.3221428>
- [30] Franch, X., Palomares, C., Quer, C., Chatzipetrou, P., Gorschek, T. (2023). The state-of-practice in requirements specification: an extended interview study at 12 companies. *Requirements Engineering*, 28(3): 377-409. <http://doi.org/10.1007/s00766-023-00399-7>
- [31] Mahmud, I., Veneziano, V. (2011). Mind-mapping: An effective technique to facilitate requirements engineering in agile software development. In 14th International Conference on Computer and Information Technology

- (ICCIT 2011), Dhaka, Bangladesh, pp. 157-162. <http://doi.org/10.1109/ICCITech.2011.6164775>
- [32] Koutsopoulos, G., Kjellvard, N., Magnusson, J., Zdravkovic, J. (2020). Towards an integrated meta-model for requirements engineering. In: Proceedings of the Practice of Enterprise Modelling 2019 Conference Forum, RWTH Aachen University, pp. 40-53.
- [33] Kalfat, H., Oussalah, M., Chikh, A. (2023). ADM: An agile template for requirements documentation. In Proceedings of the 18th International Conference on Software Technologies (ICSOFT 2023), Rome, Italy, pp. 494-501. <http://doi.org/10.5220/0012122400003538>
- [34] Gill, A.Q., Henderson-Sellers, B., Niazi, M. (2018). Scaling for agility: A reference model for hybrid traditional-agile software development methodologies. *Information Systems Frontiers*, 20: 315-341. <https://doi.org/10.1007/s10796-016-9672-8>
- [35] Ortega-Ordoñez, W.A., Pardo-Calvache, C.J., Pino-Correa, F.J. (2019). OntoAgile: An ontology for agile software development processes. *Dyna*, 86(209): 79-90. <http://doi.org/10.15446/dyna.v86n209.76670>
- [36] Kuhrmann, M., Münch, J., Diebold, P., Linssen, O., Prause, C. (2016). On the use of hybrid development approaches in software and systems development: Construction and test of the HELENA survey. *Gesellschaft für Informatik*, pp. 59-68. <https://dl.gi.de/items/56845282-077d-41b7-ab86-a1af5847514d>.
- [37] Theocharis, G., Kuhrmann, M., Münch, J., Diebold, P. (2015). Is water-scrum-fall reality? On the use of agile and traditional development practices. In *Product-Focused Software Process Improvement: 16th International Conference, PROFES 2015, Bolzano, Italy*, 16: 149-166. [https://doi.org/10.1007/978-3-319-26844-6\\_11](https://doi.org/10.1007/978-3-319-26844-6_11)
- [38] Hoda, R., Noble, J., Marshall, S. (2012). Documentation strategies on agile software development projects. *International Journal of Agile and Extreme Software Development*, 1(1): 23-37. <http://doi.org/10.1504/IJAESD.2012.048308>
- [39] Object Management Group. (2008). *Software & Systems Process Engineering Metamodel. About the Software & Systems Process Engineering Metamodel Specification Version 2.0*. <https://www.omg.org/spec/SPEM/2.0/>.
- [40] Batra, M., Bhatnagar, A. (2017). A comparative study of requirements engineering process model. *International Journal of Advanced Research in Computer Science*, 8(3): 740-745. <https://doi.org/10.26483/ijarcs.v8i3.3088>
- [41] Loucopoulos, P., Karakostas, V. (1995). *System requirements engineering*. McGraw-Hill Book Co., Europe. <https://dl.acm.org/doi/abs/10.5555/545779>.
- [42] Riechert, T., Lauenroth, K., Lehmann, J., Auer, S. (2007). Towards semantic based requirements engineering. In *Proceedings of the 7th International Conference on Knowledge Management (I-KNOW)*. Springer Berlin Heidelberg. <http://jens-lehmann.org/files/2007/swore.pdf>.
- [43] Schwaber, K., Sutherland, J. (2011). *The scrum guide*. Scrum Alliance, 21(1): 1-38. <http://dx.doi.org/10.1002/9781119203278.app2>
- [44] Robertson, J., Robertson, S. (2000). *Volere requirements specification template*. <https://www.volere.org/templates/volere-requirements-specification-template/>.
- [45] Robertson, S., Robertson, J. (2012). *Mastering the requirements process: Getting requirements right*. Addison-Wesley. <https://www.volere.org/mastering-the-requirements-process-getting-requirements-right/>.
- [46] Mosser, S., Pulgar, C., Reinhar, V. (2022). Modelling agile backlogs as composable artifacts to support developers and product owners. *The Journal of Object Technology*, 21(3): 3-1. <http://doi.org/10.5381/jot.2022.21.3.a3>
- [47] Yourdon, E., Argila, C. (1996). *Case studies in object oriented analysis & design*. Prentice-Hall, Inc., United States. <https://dl.acm.org/doi/abs/10.5555/235257>
- [48] Bolloju, N., Alter, S., Gupta, A., Gupta, S., Jain, S. (2017). Improving Scrum user stories and product backlog using work system snapshots. *Systems Analysis and Design (SIGSAND)*, 7. <https://aisel.aisnet.org/amcis2017/SystemsAnalysis/Presentations/7/>.