Representation of some principles of the Agile Manifesto in the Semat Essence Kernel

Representación de algunos principios del Manifiesto Ágil en el núcleo de la Esencia de *Semat*

C. M. Zapata-Jaramillo ; D. E. Yepes-Palacio
DOI: 10.22517/23447214.24241
Article of scientific and technological research

***Abstract*— The Agile Manifesto (AM) provides principles guiding agile software development as an alternative to traditional software development processes. While attempts have been made to adapt processes to the agile context, representation of AM principles remains underdeveloped and subjective. The Semat (Software Engineering Method and Theory) Essence kernel offers a common ground for representing software development endeavors. In this paper, we represent some AM principles using the language provided by the Semat Essence kernel to establish a common ground. Additionally, we define constraints in OCL (Object Constraint Language) to enhance the Semat Essence kernel, enabling the introduction of time management in our representation. Such a representation will allow us for adapting and assessing different processes in an agile context.**

***Index Terms*—Agile Manifesto, Agile Processes, Representation, Semat, Software Development Process**

***Resumen*— El Manifiesto Ágil tiene un conjunto de principios que guían el desarrollo ágil de software, el cual se presenta como una alternativa a los procesos de desarrollo tradicionales. En la revisión de la literatura hay varios intentos para tratar de adaptar diferentes procesos en contextos ágiles, pero esta adaptación se dificulta debido a que no existe una representación del Manifiesto Ágil en un terreno común y, además, la interpretación de esos principios puede tener subjetividad. El núcleo de *Semat* (*Software Engineering Method and Theory*) tiene un terreno común para representar cualquier esfuerzo de ingeniería de software. En este artículo se representan algunos principios del Manifiesto Ágil en un terreno común usando el lenguaje que provee el núcleo de la Esencia de *Semat*. Además, se introduce el manejo del tiempo en la representación de dicho núcleo mediante el lenguaje OCL (*Object Constraint Language*). La representación de los principios del Manifiesto Ágil en un terreno común permitirá adaptar y evaluar diferentes procesos en contextos ágiles.**

***Palabras claves*—Manifiesto Ágil, Proceso de desarrollo de Software, Procesos Ágiles, Representación, *Semat*.**

I. Introduction

Agile methods have been adopted in software development companies as an alternative to the traditional software development process and they are intended to improve the software development processes [1]. Agile Manifesto helps to guide the agile methods in their purpose by means of four values and twelve principles directed to customer satisfaction, fast responses to change, and quick software delivery.

Some attempts for adapting processes to the agile context can be found in the state of the art. Some authors attempt to replicate, use, and adapt the Agile Manifesto principles and values in several processes like development of embedded systems [2], software product lines (SPL) [3], and translation of traditional methods into agile methods [4]. However, the adaptation among different processes is difficult, since comparison among agile and traditional processes is still underdeveloped. In fact, the Agile Manifesto lacks a common ground representation and consequently the agile manifesto principles can be subjectively interpreted.

Semat (Software Engineering Method and Theory) is an initiative founded by Ivar Jacobson, Bertrand Meyer, and Richard Soley for creating a common ground for software engineering [5]. Semat promotes a kernel with elements—*e.g.*, work products, activity spaces, and practices—for representing any software development endeavor.

In this paper we develop a representation of some principles of the Agile Manifesto by using some elements of the Semat Essence kernel—*e.g.*, alphas, states, work products, patterns and activity spaces. Also, we introduce a new syntax based on OCL (Object Constraint Language) statements in order to incorporate temporal constraints in the representation. Such representation allows us for adapting processes like embedded system development and software products lines to the agile context. We can also assess agility of several processes and methods.

This paper is organized as follows: in Section II we present

This manuscript was submitted on May 18, 2020, accepted on November 16, 2023 and published on March , 2023.. C. M. Zapata-Jaramillo is Full Professor of the Computer and Decision Sciences, Faculty of Mines, Universidad Nacional de Colombia, Sede Medellín (e-mail: cmzapata@unal.edu.co).

D. E. Yepes-Palacio is Professor of the Systems Engineering Department, Universidad de Antioquia (e-mail: desteban.yepes@udea.edu.co).

This work has been supported by the Research Group in Computing Languages of the Universidad Nacional de Colombia

the theoretical framework of this research; in Section III we discuss the state of the art related to the agile manifesto representations; in Section IV we propose the representation of some principles of the agile manifesto; conclusions and future work are discussed in Section V.

II. Theoretical framework

A. *Agile Manifesto*

Agile Manifesto covers better ways of developing software [6]. 17 people have met in 2001—called by Kent Beck—for talking about better ways to develop software applications and they reach an agreement called "agile methods" to describe a set of lighter methods compared to traditional methods [6].

Agile Manifesto creators establish four values focused on individuals and interactions, working software, customer collaboration, and answers to change. They also define twelve principles related to agile methods, including frequent delivery of valuable software, management of changing requirements, simplicity, self-organizing teams, motivation, and sustainable development [6].

B. *Semat (Software Engineering Method and Theory).*

Semat is an initiative created by Ivar Jacobson, Bertrand Meyer, and Richard Soley in 2009, after recognizing that software engineering practices suffer certain specific maturity problems [5]. Semat has two fundamental goals related to software engineering [5]:

- Finding a kernel of widely-agreed elements.
- Defining a solid theoretical basis.

The first goal is achieved by establishing a common ground for all software engineering endeavors. In the Semat Essence kernel three main components are defined: “things we always work with,” represented by means of alpha elements; “things we always do,” represented by means of activity space elements; and “skills we always need to have,” represented by means of competency elements. We present the key elements of the Semat Essence Kernel in Table I.

Alphas represent the key concepts involved in software engineering; they provide a common ground for assessing the progress and health of any software engineering endeavor [5].

Activity spaces represent the things teams and stakeholders always do in a software engineering endeavor—*e.g.*, use the system, understand the requirements, deploy the system, and track progress.

Competencies are skills we always need to have in any software engineering endeavor.

The Semat Essence kernel has a set of main elements for representing the practices:

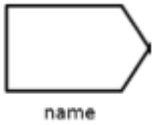
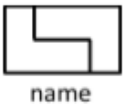
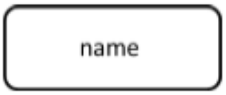
- Alphas
- Alpha states
- Activity spaces
- Competencies
- Work Product
- Activity
- Pattern

OCL (Object Constraint Language) is used in the Semat Essence kernel as a language specification for establishing invariants/constraints inside the meta-model and elements. An example of such constraints in the alpha states is the following:

self.states -> forAll(s1, s2 | s1 <> s2 implies s1.name <> s2.name) [7]

According to this OCL expression, alpha states have always a different name from each other. In addition, OCL has an extension to support temporal constraints by means of patterns and events [8].

TABLE I
SOME ELEMENTS OF THE SEMAT ESSENCE KERNEL

Element	Representation	Description
Alpha		Representations of the essential things to work with.
Activity Space		Representations of the essential things to do. An activity space is visualized by the dashed-outline symbol, either containing the name of the activity space or with the name of the Activity Space placed below the symbol.
Activity		An activity defines one or more kinds of work items and gives guidance on how to perform them.
Competency		Representations of the key capabilities required to carry out the work of software engineering. A competency is visualized by a 5-point star symbol with the name of the Competency placed below the symbol.
Patterns		Generic mechanism for naming complex concepts that are made up of several Essence elements.
Alpha State		A specification of the state of progress of an alpha.
Work Product		Artifact of value and relevance for a software engineering endeavor.
Work product manifest		A work product manifest is visualized by a solid line connecting an alpha and a work product.
Activity Association (“part-of” kind)		An activity association of the “part-of” kind is visualized by a solid line connecting an activity space and an activity.
Pattern Association		A pattern association is visualized by one or more solid lines originating from a circle that connects each associated element within the pattern

Source: The authors based on Jacobson *et al.* [5]

III. State-of-the-art review

Even though some graphical representations of the agile manifesto are underdeveloped, several authors try to map the agile manifesto into some processes like software product lines, system development (hardware development), and traditional software development, among others.

Da Silva *et al.* [3] try to understand how to associate software product lines and the Agile Manifesto principles. They make a case study, a round with expert judgment, and a mapping for discovering one approach is inadequate to relate the software product lines to the Agile Manifesto principles. Also, they discover analytical and empirical evidence could be subjective. In their research, they collect evidence from SPL and map it into different agile manifesto principles. For instance, according to Da Silva *et al.* [3], in software product lines the "time-box is very long (months), there is no meeting to reflect about self-adaptations," and, as a result, they said the first, third, and twelfth principles are related; the rationale for supporting this assertion is Agile Manifesto "fosters that is not necessary to define everything up front before the team can start building software," and this is a justification for the iterative approach. The second agile manifesto principle is related to SPL evidence "meetings were used to communicate and update changes in the requirements," and the rationale for such relation is "if the team wants to be more Agile, it is necessary to treat the requirements as a prioritized stack which is allowed to vary over time;" that is mapped to changes in requirements. The 10th agile manifesto principle is mapped to "...a lot of documentation effort for a small return on investment (value)..." with the rationale that "focuses on high value features and strives to increase the value to the stakeholders;" similarly, the rationale is "...prioritization should be by requirements, features, and/or use cases instead of modules...;" as a result of the previous statement, they related to 10th principle because it is "aimed at the use of prioritization." Consequently, we need a common ground for comparing SPL evidences and relating them to the Agile Manifesto principles in a more objective way.

Kaisti *et al.* [2] propose the definition of an agile system development process. They say the agile manifesto is focused on software development instead of hardware development and mechanical engineering activities. In the case study, they emphasize and challenge each one of the agile manifesto principles for defining an agile system development process. In particular, Kaisti *et al.* [2] emphasize the first principle is focused on "customer satisfaction, continuous delivery, value, and early deliveries;" in this case, the challenge according to Kaisti *et al.* [2] is "definition of deliverable, long development cycles." The second principle of the agile manifesto is associated with adaptability, competitiveness, and customer benefit; according to Kaisti *et al.* [2] the challenge in agile system development is "high cost of change late in development." The third agile manifesto principle is associated with frequent deliveries; Kaisti *et al.* [2] say the challenge is "definition of deliverable, the cost of delivering the whole system, long development cycles." The 10th agile manifesto principle is related to simplicity and optimizing work, and the challenge in agile manifesto principles is face long cycles. Also in this case the lack of a common ground can help to avoid subjective interpretation of the emphasis and challenges.

A model for mapping the traditional software development process approach into agile software development process is based on several traditional software development processes—*i.e.*, spiral and waterfall—and the agile approach [4]. Such a model is shown in (1) and (2) as follows:

$$CE = \text{Implicit Factors} + \text{Explicit Factors.} \quad (1)$$

$$MF = (T, J, I, F, D, M, TG, MO, E, B, CE) \quad (2)$$

Where

MF: mapping function.
T: Large equipment to small teams.
J: Major tasks to small stories.
I: Long iterations to small sprints.
F: long feedback cycles to instant feedback.
D: late deliveries to small and quick deliveries.
M: Long meetings to daily and short meetings.
TG: Testing conducted late to early evaluation with testing.
MO: Two monitors to a terminal for pair programming.
E: Estimation with lines of code to estimation with story points.
B: Project manager to head off approach.
CE: Effective coordination.

According to Popli *et al.* [4], the model resembled by (1) and (2) is based on expert judgment and therefore some degree of subjectivity can arise. If we express the agile manifesto principles on a common ground, we can make a mapping between traditional and agile methods, and we can find parameters for making a function/model for mapping traditional methods into agile methods.

IV. Representation of some principles of agile manifesto in the Semat Essence Kernel

In this Section we propose the representation of some principles of the Agile Manifesto in a common ground. This representation can help to reduce subjectivity in the Agile Manifesto by using a software engineering standard. Furthermore, we can help to assess several processes and methods related to the principles of the Agile Manifesto. Here, we develop the representation of principles 1, 2, 3, and 10 because they represent change responses, working software, and simplicity, essential features in Agile Software Development according to the values of the Agile Manifesto.

The first principle is described in the Agile Manifesto as follows: "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software." In this principle, the main idea is the early and continuous delivery of valuable software. Here, we involve the *software system* alpha in the *usable* state, since according to the Semat Essence kernel, such a state is accomplished when "the system is usable and demonstrates all of the quality characteristics required of an

operational system" [7]. However, the *software system* alpha can be in a superior state to *usable*, so we need to define a constraint in order to assure that (see fig. 1, the constraint linked to the *software system* alpha). We also involve the *opportunity* alpha in the *value established* state, since according to the Semat Essence kernel, such a state is accomplished when "the value to the customers and other stakeholders of a successful solution that addresses the need is established" [7]. Again, we can admit the *opportunity* alpha in any other superior states, as we establish with the OCL constraint linked to such an alpha. We propose an OCL notation—outside the Semat Essence kernel standard—in the *software system* and the *opportunity* alphas in order to show minimum state to accomplish the first principle in agile manifesto. The work product represents valuable software supporting the realization of the states linked to *software system* and *opportunity* states, but we need to create a constraint related to the adjectives *early* and *continuous* in OCL for the work product (see fig. 1, the constraint linked to the work product). The OCL notation in work product suggests the early (for t>0) and continuous (0<self.period < N, where N is a variable in days) delivery, with a frequency defined as (self.frequency=f). The work product is created/updated during any activity belonging to the *deploy the system* activity space.

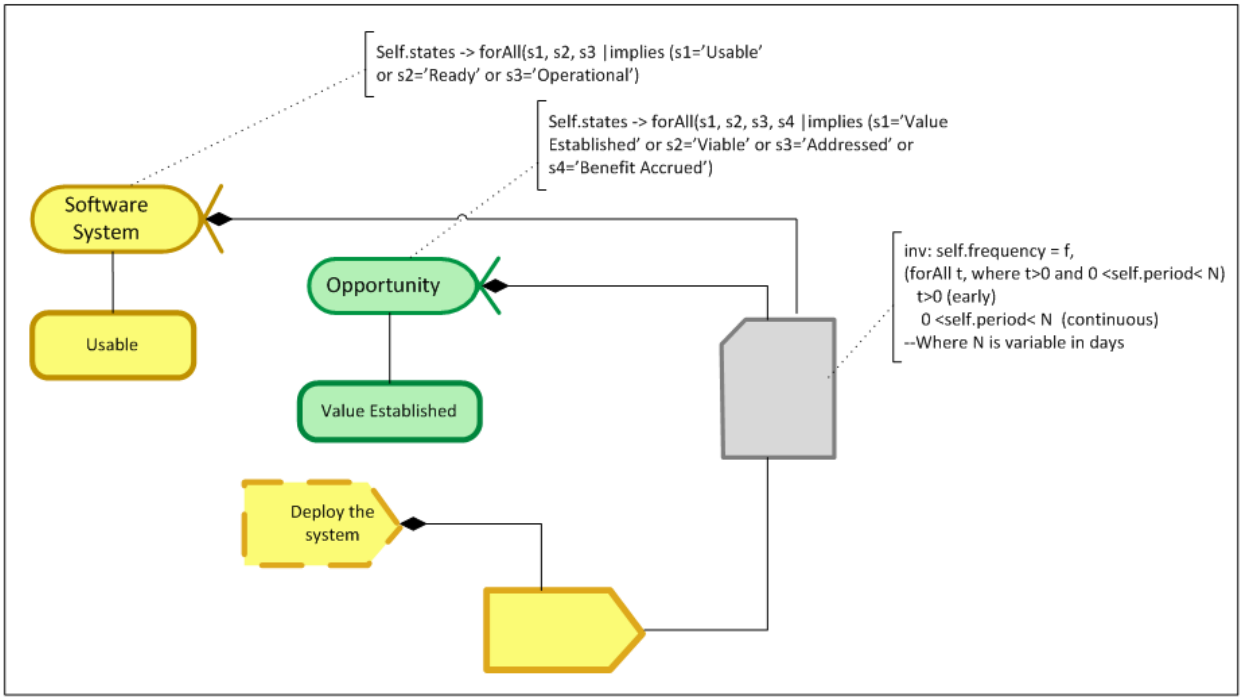


Fig. 1. Representation of the first Agile Manifesto principle

The second principle is described in the Agile Manifesto as follows: "welcome changing requirements, even late in development; agile processes harness change for the customer's competitive advantage." Adapting to change is a common feature to agile methods. Adapting requirements is established in this principle, even late in the software development process. For this reason, the *requirements* alpha is identified in the principle linked to a work product resulting from an activity belonging to the *understand requirements* activity space. Changing requirements is represented by using an OCL expression linked to the work product (see fig. 2). Such an expression has keywords such as *temp*, *eventually*, and *globally* [8]. Similar to the previous representation, *temp* keyword is related to a temporal OCL expression, *eventually* keyword indicates anytime is called for updating the work product, and *globally* keyword indicates the usage for all time in agile methods. Consequently, competitive advantage of the customer can be achieved with this principle. The full representation of second principle of the Agile Manifesto is shown in fig. 2.

The third principle is described in the Agile Manifesto as follows: "deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." The main idea in third principle is the frequent delivery of working software. At this point, the *software system* alpha is identified as the center of the principle with at least the *usable* state (See fig. 3). The principle statement includes *software* concerning the *software system* alpha since the software (the source code) is a deliverable—*i.e.*, a work product—representing a part of the *software system* alpha. According to the Semat Essence kernel specification, the alpha states can be reached by the progress of work products. *Working* in the principle can be referred to at least the *usable* state. The work product should be created/modified by using an activity belonging to the *deploy the system* activity space. This activity is commonly performed by a role belonging to the *endeavor* area of concern—*e.g.*, the Scrum master if we are working in Scrum. We fully represent the third principle of the Agile Manifesto in fig. 3.

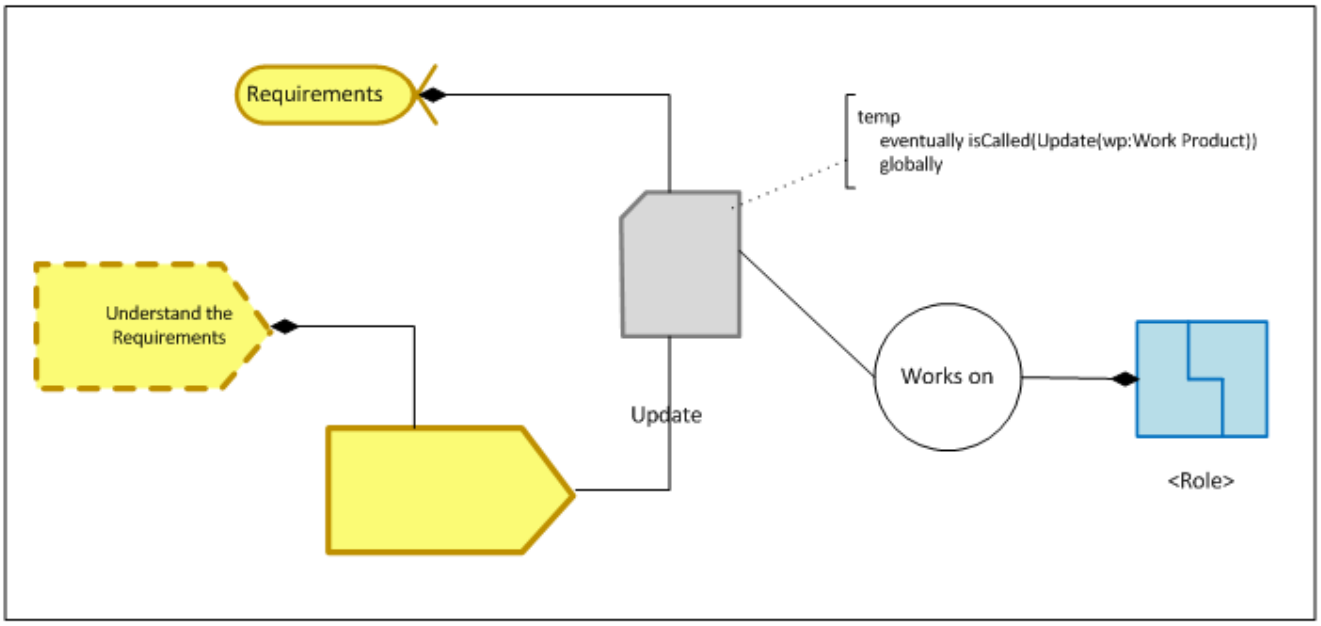


Fig. 2. Representation of the second Agile Manifesto principle

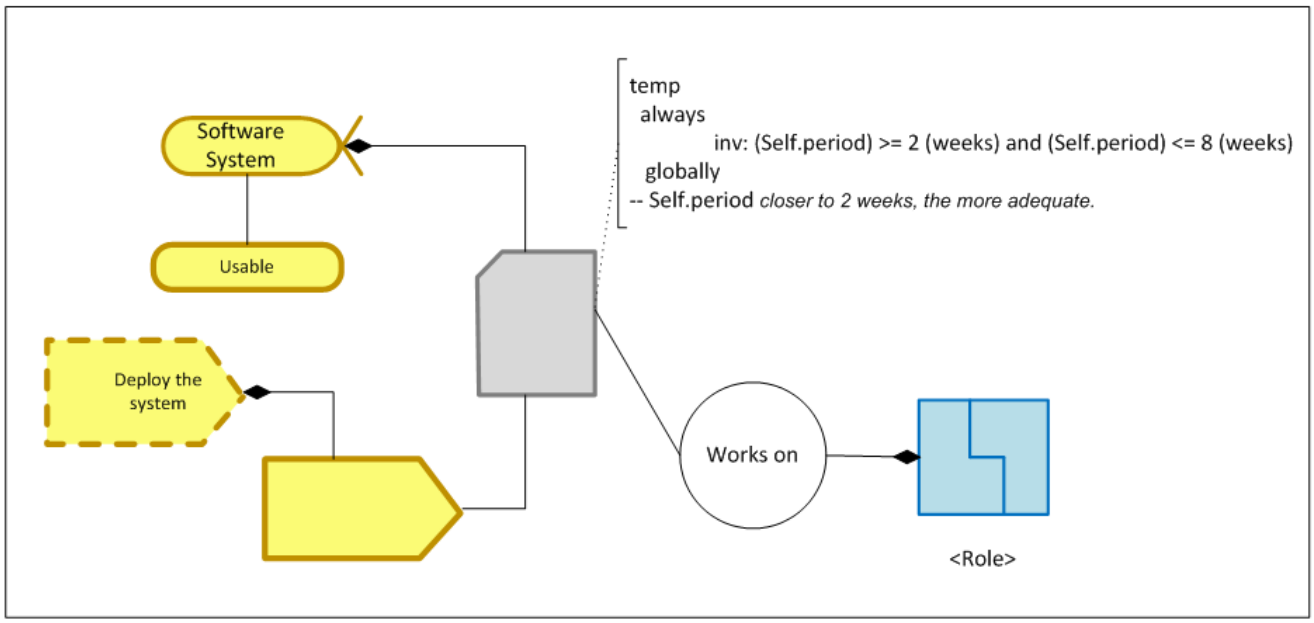


Fig. 3. Representation of the third Agile Manifesto principle

Delivery of working software has a timescale represented by using temporal constraints in the OCL notation. The *temp*, *always*, and *globally* keywords are temporal constraints [8]; *temp* keyword is the header of a temporal OCL expression, *always* keyword indicates the full applicability of the timescale, and *globally* keyword indicates implementation throughout the time within the agile framework. The OCL expression linked to the work product in fig. 3 shows the work product has an invariant to be always applied invariant in a period from couple of weeks to a couple of months (eight weeks) during an agile framework.

The 10th principle is described in the Agile Manifesto as follows: “simplicity—the art of maximizing the amount of work not done—is essential.” According to this principle, simplicity is an essential feature of agile methods. The method components such as method specification, work products, and processes should be simple; for this reason, we involve all alphas in this representation, besides, as Kent beck says "there's a strong taste of minimalism in all the agile methods (…) include only what everybody needs rather than what anybody needs…", *i.e.*, all components in agile methods should match minimalism. Similarly, alphas in agile methods should have

few and simple work products, along the three areas of concern. As a result, we express the 10th principle in the Semat Essence kernel in fig. 4.

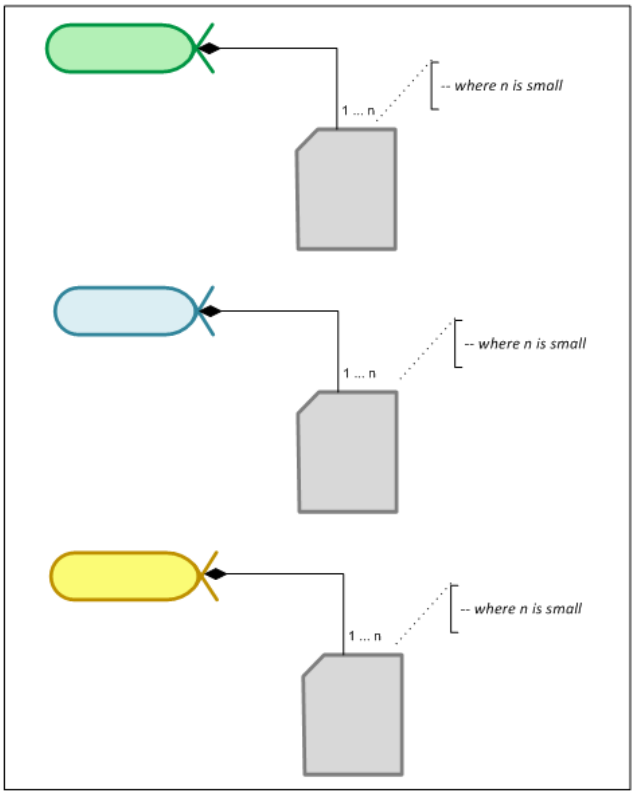


Fig. 4. Representation of the 10th Agile Manifesto principle

The represented principles can be used for comparing with other Semat-Essense-kernel-based representations in order to determine the applicability of the principles belonging to the Agile Manifesto into agile methods. For example, Jacobson *et al.* [9] represent the so-called *Scrum lite* practice in the Essence language, as we show in fig. 5. By comparing the fig. 1 to the fig. 4 with fig. 5, we can identify some similarities: *requirements* and *software system* alphas are linked to work products in the *Scrum lite* practice (see fig. 5) in a similar way to the four principles represented. Also, we can identify *Scrum master* as one of the roles linked to the process in a similar way to both the second and third principles. However, the way work products are used is unclear in fig. 5, since Jacobson *et al.* [9] avoid the usage of OCL expressions; some other elements like cardinality of the work products and additional information about time of frequency of usage are also out from the representation. In absence of such information, we can only recognize the terminology about Scrum (*e.g.*, Scrum team, daily Scrum, sprint backlog, etc.) in order to characterize a practice about Scrum.

Also, accomplishment of the principles of the Agile Manifesto is limited in absence of either OCL constraints or additional information for evaluating such constraints. For example, González-Pérez *et al.* [10] represent some practices related to Rational Unified Process (RUP) [11] by using the Semat Essence kernel, as we show in fig. 6 and fig. 7. If we compare fig. 6 and fig. 7 with fig. 1 and fig. 4, we can see the *software system* and *opportunity* alphas linked to work products in a way similar to the first and tenth principles of the Agile Manifesto. However, we lack enough information in order to validate the constraints. In this case, RUP is a plan-based method instead of an agile method, so we can expect accomplishment of the Agile Manifesto principles in agile methods. As a summary, we need additional information for verifying the restrictions we have in the representations of the Agile Manifesto principles.

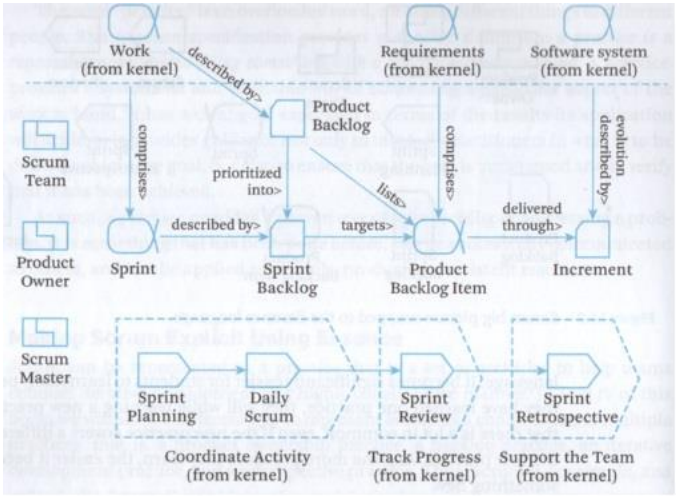


Fig.5. Representation of the *Scrum lite* practice [9]

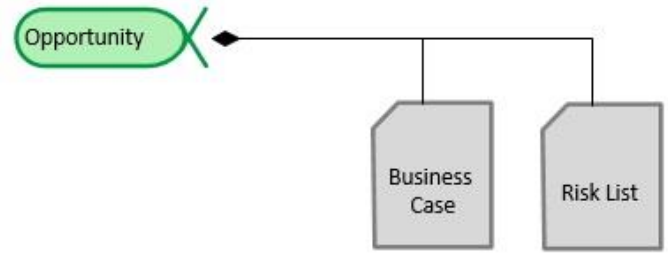


Fig.6. Representation of the RUP practice related to the *opportunity* alpha [10]

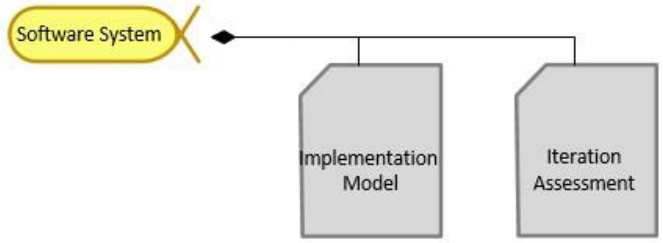


Fig.7. Representation of the RUP practice related to the *software system* alpha [10]

Notwithstanding the aforementioned problems regarding the accomplishment of the constraints we discover for the principles of the Agile Manifesto, we can use the common ground we defined for making objective comparisons about the topics we have in the state-of-the-art review as follows.

Da Silva *et al.* [3] advocates the first and third principles are co-related. A single look to fig. 1 and fig. 3 let us discover that the representations have in common the *software system* alpha,

the *usable* state, the work product and the activity belonging to the *deploy the system* activity space. In this way we can objectively declare they correlate with each other. We can also use the Semat Essence kernel for representing the rationale, if we want to make more explicit the relationship between the principles and SPL, but we need the OCL expressions for dealing with expressions like "agile," "up front," "before," "over time," and "high value".

We can also use our representation for analyzing the work of Kaisti *et al.* [2]. As a matter of fact, we can now recognize the elements they are emphasizing about the first principle (see fig. 1), since customer satisfaction and value are linked to the *client* area of concern (the green elements) and the other elements (continuous and early deliveries) are linked to the *solution* area of concern (the yellow elements). Also, we represented the adjectives *early* and *continuous* by using the OCL constraint attached to the work product. Regarding the second principle, we can establish the emphasis identified by Kaisti *et al.* [2] seem to be very subjective, since adaptability, competitiveness, and customer benefit are abstract themes difficult to relate to the second principle. The emphasis defined by Kaisti *et al.* [2] related to the third principle is objectively mapped into fig. 3, by recognizing the work product and the activity belonging to the *deploy the system* activity space, in addition to the OCL expression for defining the frequent delivery of software systems. The emphasis related to the 10th principle is also very abstract to be mapped into our representation.

On the other hand, the model defined by Popli *et al.* [4] for mapping the transition from traditional to agile development can be objectively analyzed in terms of the elements of the Semat Essence kernel, as we show in Table II. The main difference in this case is the possibility to generate OCL constraints for defining the main qualifiers of the Popli *et al.* [4] mapping model: large, small, major, long, instant, late, quick, long, daily, short, early, and effective. If we can effectively represent the mapping model from traditional to agile software development by including the OCL constraints related to the qualifiers of the factors involved, we can decide the *agility* trend of a practice. Otherwise, we can only decide on a subjective manner. For example, if we review the presence of "agile" elements in the *Scrum lite* practice of the fig. 5 [9], we can see only one element of the right side of the mapping model of Popli *et al.* [4], *i.e.*, the *sprint* sub-alpha, and some other ones similar—*e.g.*, *Scrum team* <role> pattern vs. *team* alpha and *daily scrum* activity vs. *meeting* work product. However, we lack objective evidence for assuring whether the *Scrum lite* practice belongs to an agile development or not. We need more information to objectively define whether a practice belong to an agile development method or not, and such information is related to the way things are linked to the principles of the Agile Manifesto. The common ground we are using to represent the principles of the Agile Manifesto—including the OCL constraints—is the initial resource for evaluating agility, but we need to establish the well formedness of such principles in the realm of a software engineering theory, the one provided by the Semat Essence kernel.

TABLE II
SOME ELEMENTS OF THE SEMAT ESSENCE KERNEL

Mapping factors [4]	Semat Essence kernel elements
Large equipment to small teams	*Team* alpha
Major tasks to small stories	*Task* and *User story* work product
Long iteration to small sprints	*Iteration* work product and *Sprint* sub-alpha
Long feedback cycles to instant feedback	*Feedback* work product
Late deliveries to small and quick deliveries	*Delivery* work product
Long meetings to daily and short meetings	*Meeting* work product
Testing conducted late to early evaluation with testing	*Conducting tests* activity
Two monitors to a terminal for pair programming	*Frequent pair programming* practice
Estimation with lines of code to estimation with story points	*Lines of code* and *User story* work product
Project manager to head off approach	*Project manager* <role> pattern
Effective coordination	*Leadership* competency

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed the usage of the Semat Essence kernel for representing some principles of the Agile Manifesto in a common ground. We also proposed a way to represent some qualifiers—*e.g.*, timing and sizing—used for the elements of the principles by using OCL constraints outside the Semat Essence standard. We validated our proposed representation by assessing an allegedly agile practice with our representation. Finally, we tested the state-of-the-art studies about the principles of the Agile Manifesto by translating them into our representation. We discover the principles of the Agile Manifesto are still immature in their statements and we need a more formal and well-formed way to represent them in order to be used as a proof of concept about the agility of a statement. This study will help to avoid subjectivity, provide a common ground inside team work, and enable the use of the principles of the Agile Manifesto for different purposes, such as assessing processes, assessing frameworks, and assessing practices, whether they are agile or not.

We define some lines of future work as follows:

- Representing all of the principles of the Agile Manifesto by using our proposal and then assessing some agile methods—*e.g.*, Scrum, Extreme Programming, and Feature Drive Development—and some traditional methods—*e.g.*, Rational Unified Process, the UNC-Method, and Custom Development Method—regarding

the Agile Manifesto. We want to establish their compliance to the principles of the Agile Manifesto.

- Defining other OCL constraints for representing other qualifiers than timing and sizing—*e.g.*, effective and high-value.
- Representing the whole mapping model of Popli *et al.* [4] by using our proposal as a way to get closer to a formal and well-formed equation for transforming traditional software development methods into agile ones.

REFERENCES

[1] A. Singh, K. Singh, and N. Sharma, "Agile knowledge management: a survey of Indian perceptions," *Innov. Syst. Softw. Eng.*, vol. 10, no. 4, pp. 297–315, 2014. DOI: https://doi.org/10.1007/s11334-014-0237-z

[2] M. Kaisti, T. Mujunen, T. Mäkilä, V. Rantala, and T. Lehtonen, "Agile principles in the embedded system development," *Agile Processes in Software Engineering and Extreme Programming*, vol. 179, G. Cantone and M. Marchesi, Eds. Rome: Springer, 2014, pp. 16–31. DOI: https://doi.org/10.1007/978-3-319-06862-6_2

[3] I. F. Da Silva, P. A. da Mota Silveira Neto, P. O'Leary, E. S. de Almeida, and S. R. de Lemos Meira, "Using a Multi-Method Approach to Understand Agile Software Product Lines," *Inf. Softw. Technol.*, vol. 57, no. 1, pp. 527–542, 2014. DOI: https://doi.org/10.1016/j.infsof.2014.06.004

[4] R. Popli, R. Anita, and N. Chauhan, "A mapping model for trans-forming traditional software development methods to agile-methodology," *Int. J. Softw. Eng. Appl.*, vol. 4, no. 4, pp. 53–64, 2013. DOI: https://doi.org/10.5121/ijsea.2013.4405

[5] I. Jacobson, P. Ng, P. E. Mcmahon, and C. (Traductor) Zapata, "La Esencia de la Ingeniería de Software: El Núcleo de Semat," *Rev. Latinoam. Ing. Softw*, vol. 1, no. 3, pp. 71–78, 2013. DOI: https://doi.org/10.18294/relais.2013.71-78

[6] M. Fowler and J. Highsmith, "The agile manifesto," *Softw. Dev.*, vol. 9, no. 8, pp. 28–35, 2001.

[7] Object Management Group, "Essence–Kernel and Language for Software Engineering Methods, version 1.2," 2018.

[8] B. Kanso and S. Tala, "Temporal constraint support for OCL." In International Conference on Software Language Engineering, pp. 83–103, 2012. DOI: https://doi.org/10.1007/978-3-642-36089-3_6

[9] I. Jacobson, H. Lawson, P.-W. Ng, P. McMahon, and M. Goedicke. *The essentials of modern software engineering: free the practices from the method prisons!,"* Milton Keynes, UK: ACM Books, 2019. DOI: https://doi.org/10.1145/3277669.3277673

[10] M. González-Pérez, C. M. Zapata-Jaramillo, L. González-Palacio. "Toward a standardized representation of RUP best practices of project management in the Semat kernel," in *Software engineering: methods, modeling, and teaching*, vol. 3, chapter 7, Zapata, C. M. & Castro, L. (Ed.). Medellín, Colombia: Universidad Nacional de Colombia, 2014.

[11] Rational. "Rational Unified Process: Best Practices for Software Development Teams" Rational Software White Paper TP026B, 1998.

Carlos Mario Zapata-Jaramillo received the M.S. and Ph.D. degrees in systems and software engineering from the Universidad Nacional de Colombia and he is currently serving as full Professor in the Computing and Decision Science Department at the Medellin Headquarters of the same institution. He is also president of the Executive Committee of the Latin American Chapter of Semat and also is one of the official translators of the book "The Essence of Software Engineering—Applying the SEMAT Kernel".
ORCID: https://orcid.org/0000-0002-0628-4097

Daniel Esteban Yepes-Palacio received a degree in Systems Engineering from the University of Antioquia and a M.S. degree in Systems and Software Engineering from the Universidad Nacional de Colombia. Since 2013, he has worked in the field of software engineering at several companies in the telecommunications and finance industries. Additionally, since 2019, he has served as a professor of Systems Engineering at the University of Antioquia, teaching courses in Software Engineering, where his areas of specialty and research include Software Quality and Software Testing.
ORCID: https://orcid.org/0000-0003-0635-1541