World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:15, No:1, 2021

Methodologies, Systems Development Life Cycle and Modeling Languages in Agile Software Development

I. D. Arroyo

***Abstract*—**This article seeks to integrate different concepts from contemporary software engineering with an agile development approach. We seek to clarify some definitions and uses, we make a difference between the Systems Development Life Cycle (SDLC) and the methodologies, we differentiate the types of frameworks such as methodological, philosophical and behavioral, standards and documentation. We define relationships based on the documentation of the development process through formal and ad hoc models, and we define the usefulness of using DevOps and Agile Modeling as integrative methodologies of principles and best practices.

***Keywords*—**Methodologies, SDLC, modeling languages, agile modeling, DevOps, UML, agile software development.

I. Introduction

THIS article presents some advances of a research plan for the doctorate in informatics technologies from the University of Malaga. The research plan called: Agile design patterns to reduce the complexity of the models taking the experiences of the software industry. It seeks to identify, classify, and formalize the non-formal notations or ad-hoc models [36] that software developers are using to model agile software. The study started by applying a survey to a sample of 8000 emails from software development companies from 82 countries from January 2020, but the response was not as expected (only 138 developers have responded so far). However, important participations were received that identified the need to redefine some concepts to give relevance to the use of models in agile development.

The appearance of such a wide variety of methods, models, frameworks, programming languages, and diverse and new emerging concepts, have made it difficult to reach a coherent integration that facilitates agile development, or that presents improvements that can be incorporated as good practices that allow developers to orient themselves to achieve an integration of possibilities that could be useful to their development teams [15]. The main objective is to clarify the concepts of Methodologies, SDLC and modeling languages, identifying their functional differences and their possibilities of integration in an agile development process.

The documentation of the agile development process is identified as an important point, making use of different types of formal and ad-hoc models based on the integration of Agile Modeling a documentation methodology that facilitates the creation of documentary witnesses in different formal notations or ad hoc [15]. Likewise, the appearance of DevOps is identified as an important milestone as a philosophical or behavioral framework that must be understood as a comprehensive approach to operation that not only includes automation, but also facilitates communication between members of the development team as well as with operators, allowing different frameworks and approaches to be integrated into a global, customer-oriented and adaptable system [8].

The integration of all these concepts and possibilities constitutes a current vision of what contemporary software engineering is forming, integrating not only academic concepts, but also those produced in the industry.

I. D. Arroyo is a PhD student at the University of Málaga, Boulevar Luis Pasteur, Teatinos Campus 29071, 29010 Málaga, Spain (e-mail: ivan.arroyo@uma.es).

II. Methodologies and Life Cycle

A. Overview

From the origins of software engineering, SDLC was defined as a structured sequence of the “stages” of the development process; these stages were later thought to define the entire life of the software from its construction, operation, obsolescence and disuse (death) as Pressman illustrates in his graph: "Idealized and actual failure curves for software" [30]. There are many discussions regarding what activities should be developed in each "stage", which has led to the appearance of different frameworks to describe methodologies. These frameworks are being adapted by each development team and there can be as many different adaptations as there are developers or teams in the industry, so choosing a software development methodology can be a daunting and extremely difficult task [15].

The experiences of use of the frameworks that have been proposed over the years demonstrate a difference between SDLC and methodology. For example, Scrum was proposed to be suitable for an iterative SDLC [2]; however, there are adaptations of Scrum in an SDLC in cascade [16], [23] or using a prototype SDLC [18], [35], [37]. These adaptations show an independence between the SDLC and the methodology. To clarify this point, the SDLC should be understood as the logical sequence of phases in time, while the methodology defines the phases that must exist and the logical sequence of activities that must be carried out in each phase; that is, the methodology proposes the phases, but the SDLC proposes the time in which these phases are executed.

B. Usefulness of Differentiating between Methodology and SDLC in Agile Development

Choosing a methodology can be difficult for a development team, especially when looking to move from traditional development to agile development [15]. However, some

documented experiences regarding the adoption of agile methodologies, analyzed in more depth, show that in reality, much of the improvement obtained consisted of going from a Waterfall or traditional SDLC to an iterative or prototyping SDLC [16], [17], [25], [29], [34]. This improvement does not necessarily imply implementing agile software development; for example, the Rational Unified Process (RUP), already used an iterative SDLC and it was not in an agile development methodology [26], and although RUP became Agile UP later, and continued using the iterative SDLC, this does not mean that the iterative SDLC is necessarily agile, because it was used with the RUP framework before agile frameworks emerged.

Some of the best known SDLCs are, waterfall or traditional, iterative, spiral, prototyping, rapid iterative production prototyping (RIPP) and incremental SDLC.

Some of the best-known methodologies are:

- 6D-BUM
- Acceptance test driven development (ATDD)
- Adaptive software development (ASD)
- Agile Unified Process (AUP)
- Constructionist design methodology (CDM)
- Crystal Clear
- Dynamic systems development method (DSDM)
- Enterprise release management (ERM)
- Enterprise Unified Process
- Extreme programming (XP)
- Feature-driven development (FDD)
- Freedom
- G300
- Joint application development (JAD)
- Kanban
- Open Unified Process (OpenUP)
- Personal Software Process PSP
- PMI Agile
- Rapid-application development (RAD)
- RUP
- Scrum
- Scrumban
- Site Reliability Engineering (SRE)
- Team software process (TSP)
- Top-down and bottom-up
- Unified process

Each of these methodologies can execute its phases in any of the SDLC described above For example, using Kanban, the general phases are defined: Backlog, Planned, In Progress, Developed, Tested and Completed, the time in which the phases are executed in the SDLC: Cascade or traditional, iterative and prototyping can be seen in Fig. 1.

In general, the planned phase includes requirements taking, analysis and design, the *In Progress* phase executes development or programming, the *Tested* phase checks that everything planned has been built and that it works, and the *Completed* phase involves other implementation processes or implantation. In Fig. 1, the SDLC Waterfall is seen as a sequence similar to a Kanban board, and since each task is organized as a card, each card can be located in each phase of the SDLC Waterfall. In the prototyping SDLC, it is sought to arrive at a prototype that is evaluated and is improved each time the *Completed* phase is reached; this would imply that each card must move back to the *Planned* phase each time this prototype is evaluated. This happens frequently in Agile development when each version delivered needs improvement. In the Iterative SDLC, the *In Progress* phase is divided into several iterations of *Planned, In Progress, Developed, Tested* and *Completed* phases; each card goes through each iteration until development is finally completed.

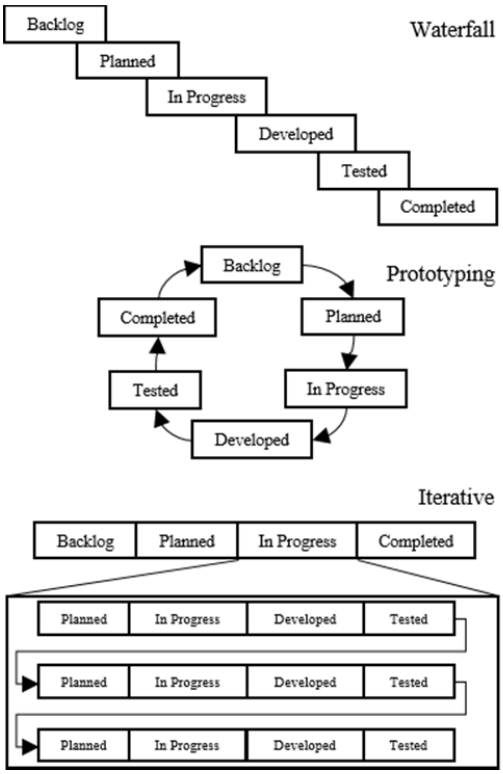

Fig. 1 Kanban methodology using three different SDLCs

In methodologies such as scrum, some "Sprints" (a time-box of one month or less during which a "Done") are programmed, however, it does not necessarily mean that in a sprint, the software or any of its modules is delivered, but an advance of the project is presented [23]. These Sprints are generally tested in "Agile Testers" with a mini-waterfall model that has the same structure as the waterfall SDLC [23], but this does not mean that the Sprint is an SDLC or that Scrum has its own SDLC.

C. Frameworks and Standards in Agile Development

Frameworks are the formal guide for implementing a methodology, according to which, some institutions certify competencies through exams. For example, the Scrum framework implies the use or implementation guide that professionals can study to become certified as a Product Owner or Scrum Master among others. A methodological framework is a non-mandatory standard that provides some good practices that can be partially or fully implemented.

Each software development methodology has its framework and its objectives are strictly technical. However, experience has caused DevOps to emerge with a philosophical framework [8]; this is a very interesting milestone, since it is the first time that it is considered necessary to document good practices that, far from leading to fulfill technical objectives, lead to fulfill objectives of behavior [8]. In the context of software development, it has been difficult to understand this topic, so there are studies that seek to trap DevOps within the technical framework, claiming that it has a "conceptual deficit" [3]; however, DevOps has been well received in organizations since it is related to the organizational culture [12] and allows communication and collaboration between operators and developers [24]. This collaboration and communication involve the adoption of an ethics rather than the adoption of technical skills. DevOps should be used as an "integrating philosophy" and understood as an opportunity for developers and operators [8]. Philosophical or behavioral frameworks are an emerging concept, which arises with DevOps and can be used to incorporate non-technical skills in work teams, which can facilitate the implementation of methodological frameworks or standards frameworks, because behavior needs to be changed to reduce resistance to change, or facilitate collaboration between different areas.

Software standards such as ISO, IEEE, and other standards imposed by governments, implementing organizations, or others must also be studied and implemented by developers and companies. This implies a challenge, because to demonstrate compliance with these standard frameworks it is necessary to continuously create standardized documents. The time it takes to create these documents can cause delays in software delivery, directly affecting agile methodologies [15]. Within these frameworks, CMMI (Capability Maturity Model Integration) is also counted as a regulatory framework that allows the improvement and evaluation of processes for the development, maintenance and operation of software systems, of which, there are already investigations that show some benefit when using agile methodologies with CMMI [6], [9], [33].

According to the above, software development formally requires teams to use a methodology framework that uses an appropriate SDLC. It is observed that iterative SDLC and prototype SDLC work better with agile frameworks [2], [16], [18], [23], [35], [37]. Also, regulatory bodies require compliance with standards [15], [33], and therefore, companies need to implement standards' frameworks. Furthermore, the relationship between team members, resistance to change, and other behavioral aspects can be addressed by incorporating philosophical or behavioral frameworks, which are actually an emerging topic that arises with DevOps [8], [12], [24]. Likewise, it is important to identify Agile Modeling as a documentation methodology that can be used in parallel to facilitate the creation of documentary witnesses in different formal or ad-hoc notations [15].

III. Modeling Languages in Agile Development

The three types of frameworks described above (methodology, behavior and standard) require documentary witnesses that demonstrate compliance both in development and in operation, so it is useful to rely on a documentation methodology such as Agile Modeling [15].

The technical documentation of the software includes architecture, design and API documentation [7], [11], particularly for this study, it is important to analyze the documentary evidence of architecture and design, and these elements are documented with models [10]. A more technical way to refer to documentary evidence obtained during software development is as artifacts [4], [5]. An artifact is a document (text) or diagram (drawing) [5]. These documents or diagrams represent the models in an abstract way, for example, a "user story" is a document, while a "Data Flow Diagram" (DFD) is a diagram, both are ways of representing models [5].

Interestingly, Gorschek et al. [36] surveyed more than 3,900 software developers and found that over 70% of them do not use formal models. They also report that eight other investigations also show the same results on a smaller scale. This research mainly sought to find information about the use of models in practice, mainly of UML and it was found that UML is not used as expected. It was also discovered that developers are using their own "ad-hoc models" (whiteboard drawings, tables, documents, etc.). These "ad-hoc models" are artifacts designed by companies, development teams, or developers [36]. It is also claimed that it is possible that the complexity of UML has caused developers to create their own "ad-hoc models", this could be related to the research by Fernández-Sáez et al. [1] that found an "arbitrary use" of the UML syntax, that is, the developers adapted the models in such a way that their complexity was reduced, although UML does not allow this syntax.

In agile development, some authors consider UML too complex and heavy to be really useful in an agile environment [13], [14], [19]-[22]. Although, there are also other authors who say that UML is useful in agile development making use of specific design patterns [31], [32]. Talking about the use of models is controversial, as there are many opinions about it; however, Agile Modeling has achieved the coexistence of different formal and ad-hoc models, with a successful implementation in agile development [15]. Agile Modeling includes ad-hoc models such as "Free-Form Diagrams" and "Mind Map".

There are many other formal models besides UML, such as DFDs, CRCs, etc. However, these models are not part of the same modeling language, UML unifies the object-oriented models that existed before the OOPSLA 95 workshop [4], [5]; later, modeling languages were created based on Meta-Object Facility (MOF). UML is a general-purpose language, while other MOF-based languages are domain-specific languages that fulfill the domain-specific functions they were designed for [5]. Agile Modeling has compiled these models that make sense in agile development, but have also allowed the use of ad-hoc models; apparently the Agile Modeling community

seeks to organize a catalog of models that can be used for agile development. Although, the initial result is a possible unification of formal and ad-hoc models, the next step is the formalization of models, defining a lexicon, syntax and semantics; however, this can also be controversial, because UML did it previously, although it only unified OO models.

Generally speaking, in any development process, artifacts should be used to model six general characteristics of the software, which include [5]:

- Context and Requirements: define the requirements and their initial abstraction.
- Interaction: defines the way in which the parts of the system are described in the context and requirements interact.
- Structure: defines how the parts of the system are composed, according to the context and their interaction.
- Behavior: defines how the structure of the system behaves or the behavior that the structure implies.
- Logical architecture: defines how the structural elements are organized to allow the required behavior.
- Physical architecture: defines, in general, the characteristics that make up the system and allow its execution.

TABLE I
SOFTWARE MODELS AND FEATURES

Artifact	Context and Requirements	Interaction	Structure	Behavior	Logical architecture	Physical architecture
Acceptance Test	X					
Business Rule	X		X	X		
Change Case	X					
Class Responsibility Collaborator (CRC) model			X	X		
Constraint	X					
Contract model	X			X		
Data Flow Diagram (DFD)	X	X	X	X	X	X
Domain Model			X	X	X	X
Essential/Abstract Use Case	X					
Essential/Abstract User Interface Prototype	X	X	X	X	X	X
Feature	X					
Free-Form Diagrams	X	X	X	X	X	X
Flow Chart	X	X	X	X		
Glossary	X					
Logical Data Model (LDM)		X	X	X	X	X
Mind Map	X	X	X	X	X	X
Network Diagram						X
Object Role Model (ORM) Diagram			X	X		
Personas	X	X				
Physical Data Model (PDM)		X	X	X	X	
Robustness Diagram	X	X	X			
Security Threat Model	X				X	X
System Use Case	X					
Technical Requirement	X					
UML Activity Diagram			X	X		
UML Class Diagram			X			
UML Communication/Collaboration Diagram		X				
UML Component Diagram					X	X
UML Composite Structure Diagram					X	X
UML Deployment Diagram				X	X	X
UML Interaction Overview Diagram		X				
UML Object Diagram	X		X			
UML Package Diagram				X	X	
UML Sequence Diagram		X	X			
UML State Machine Diagram				X		
UML Timing Diagram				X		
UML Use Case Diagram	X					
Usage Scenario	X					
User Interface Flow Diagram (Storyboard)	X					
User Interface Prototype	X					
User Story	X					
Value Stream Map	X	X		X		

The Agile Modeling models list can be classified as in Table I. On the other hand, in some informal sources and in the requirements of some job offers, it is shown that the industry talks about backend development and frontend development, considering that developers tend to specialize in these two areas. The frontend specializes in programming software at the presentation layer of the software architecture, while the backends specialize in producing software at the application logical layer, and some other positions require an integration architect to integrate the frontend job with the backend. This vision is not new; for example, Larman [5] describes "real use cases" in which a graphical user interface is defined, and its elements are related to the actions of the extended use case [5]. In the same way, Lutowski [28] also defines the interface and relates it to black box encapsulated processes [28]. These relationships of the graphical interface with the logical layer have a further development of the backend model but do not develop a frontend model; although, Martin [27] presents frontend models in UML.

Agile development teams typically have a wide variety of professionals, including frontend or backend developers and integration architects. Normally a single model is used, and the frontend or backend developers encode a front layer and a back layer separately, later the integration architect allows these two codes to be integrated. If instead of using a single model, a front model and a back model are used, the integration is likely to be more efficient. On the other hand, the use of models is very relative, since a person with experience and studies in UML will be more motivated to use UML than a person without experience and without studies. Similarly, ad-hoc models such as "Freeform Diagrams" or "Mind Map" or some formal models, such as DFDs, might be more useful for someone who does not know UML. This represents an advantage when using Agile Modeling, because there is a wide list of models of greater or lesser complexity that can be used in agile development with any methodology framework. Also, this extensive list of models can be used to demonstrate compliance with the required standard frameworks.

Finally, they can use behavioral or philosophical frameworks such as DevOps to improve communication between operators, backend or frontend developers, and integration architects while fostering an appropriate collaboration culture to compliance the standards frameworks.

Arbitrary notations have also been found in the use of UML. These forms of adaptation of UML models could be caused by the lack of knowledge of the syntax of this modeling language; however, these adaptations were clear enough to be understood by the team members. This would demonstrate that UML adaptations are being made to simplify models and reduce time during an agile development process [1].

The integration of these concepts, as observed in Fig. 2, allows different good practices to be implemented. The maturity of the software or the software development teams does not necessarily imply the incorporation of frameworks exclusively or rigidly; however, the integration of these allows the possibilities to be coupled to the experience of the team, allowing collaboration and even automation of some processes through DevOps [8]. Also, compliance with standards such as ISO or CMMI can be more effective if, together with collaboration, appropriate documentation strategies are established [6], [9], [15], [33], where Agile Modeling provides better benefits, not only because of the way as it is complemented by other frameworks [15], but by the open documentation possibilities, which not only involve formal notations but also models that can be adapted or proposed by the teams or the developers.

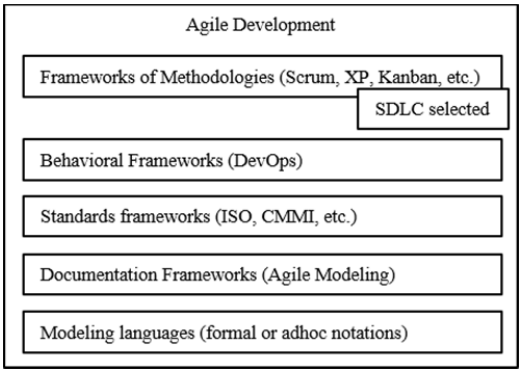

Fig. 2 Integration of software engineering concepts

IV. Conclusion

The proposal of software development methodologies such as RUP, as well as the formal definition of UML as an OO modeling language [4], [5], [14], [26], allowed to make a difference between three concepts: SDLC, Methodology and modeling language. With the advent of agile development, various methodologies have emerged that can be used with different SDLCs. For example, Scrum that was proposed with an iterative SDLC has been tested with other SDLCs such as: cascade and prototypes [2], [16], [18], [23], [35], [37]. Differentiating between these three concepts allows the organization of development teams to choose and combine the frameworks that best suit their needs.

There are four different types of frameworks:

- The standard frameworks, which are those proposed by governments and regulatory institutions such as ISO, IEEE, CMMI, etc. that allow to implement quality, safety, production standards, etc.
- The frameworks of methodologies that allow the organization of software development processes based on organized and auditable phases that are executed according to an SDLC, among these are Kanban, Scrup, UP, XP, etc.
- Philosophical or behavioral frameworks, which are an emerging concept and allow establishing guidelines that facilitate collaboration between team members, decrease resistance to change or promote appropriate behavior to implement tasks required by organizations, this concept arises with DevOps.
- Documentation frameworks, which allow defining clear

processes to generate supporting documents for each activity developed in a methodology, this concept arises with Agile Modeling.

DevOps is a philosophical or communication framework, which is essentially intended to facilitate communication between operators and developers. It is a milestone in the field, since it proposes behavioral skills to a greater extent than technical skills for its implementation, which allows facing challenges such as continuous delivery that generates frequent changes in production environments [8], the implementation of different types frameworks, communication between team members, and feedback with operators among others.

Agile Modeling is a documentation framework that unifies different formal and ad-hoc notations. This provides a large set of modeling possibilities that serve as documentary witnesses that can be used to demonstrate compliance of both standard and methodological frameworks. The advantage of allowing ad-hoc notations within the modeling capabilities of Agile Modeling is that it allows developers and teams the alternative of proposing their own models and relating them to other formal models to present more complete models. It also opens the gap to the formalization of ad-hoc models specific to developers or teams that apparently are more used than formal models such as UML [36].

Considering that the industry requires for its development teams frontend or backend developers, it is possible that Front or back layers need to be modeled separately, not only for relating the elements of a GUI with specific processes in the logical application layer [5], [28], but also to facilitate the work of the integration architect. Providing frontend and backend models could ease the task of integration, as it has already been found in previous work [27]. This specialization of modeling in the visible (frontend) and the invisible (backend) model can offer a better perspective in the development of software with high-level programming languages such as Rubi, for which modeling seems unnecessary; however, a well-specified frontend model is sufficient to demonstrate compliance with a standard or to understand the improvements or progress obtained in each iteration, "Split" or "done".

In general, modeling in agile development is especially useful for documenting what was planned, designed and happened throughout the development process, as well as making decisions when testing or feedback each "done". In the context of Agile Modeling, it allows establishing a historical memory that is useful for demonstrating compliance with standards but also for evaluating the performance of development teams and find experiences for future decision making.

REFERENCES

[1] A. M. Fernández-Sáez, M. R. V. Chaudron and M. Genero, An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles. Empirical Software Engineering. 2018

[2] A. Mundra, S. Misra and C. A. Dhawale, "Practical Scrum-Scrum Team: Way to Produce Successful and Quality Software," 2013 13th International Conference on Computational Science and Its Applications, Ho Chi Minh City, 2013, pp. 119-123.

[3] A. Wahaballa, O. Wahballa, M. Abdellatief, H. Xiong and Z. Qin, "Toward unified DevOps model," 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, 2015, pp. 211-214.

[4] C. Larman, "Tutorial: mastering design patterns," Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, Orlando, FL, USA, 2002, pp. 704-.

[5] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd Edition), Prentice Hall PTRUpper Saddle River, NJUnited States, 2004

[6] C. R. Jakobsen and K. A. Johnson, "Mature Agile with a Twist of CMMI," Agile 2008 Conference, Toronto, ON, 2008, pp. 212-217.

[7] D. L. Parnas, "Document based rational software development", Knowledge-Based Systems, vol. 22, no. 3, pp. 132-141, 2009

[8] D. Söllner, DevOps in der Praxis – Handlungsfelder für eine erfolgreiche Zusammenarbeit von Entwicklung und Betrieb. HMD Praxis Der Wirtschaftsinformatik, 54(2), 189–204. 2017. doi:10.1365/s40702-017-0303-8

[9] F. S. F. Soares and S. R. de Lemos Meira, An agile strategy for implementing CMMI project management practices in software organizations, 2015 10th Iberian Conference on Information Systems and Technologies (CISTI), 2015.

[10] G. Buchgeher, C. Klammer, B. Dorninger and A. Kern, "Providing Technical Software Documentation as a Service - An Industrial Experience Report," 2018 25th Asia-Pacific Software Engineering Conference (APSEC), Nara, Japan, 2018, pp. 581-590.

[11] G. Garousi, V. Garousi-Yusifoglu, G. Ruhe, J. Zhi, M. Moussavi, B. Smith, "Usage and usefulness of technical software documentation: An industrial case study", Information and Software Technology, vol. 57, pp. 664-682, 2015

[12] H. A. Mehairi, "Empowering Knowledge Sharing Behaviours through Means Oriented vs. Goal Oriented Cultures: The Impact of Organizational Culture on Knowledge Sharing," 2013 10th International Conference on Information Technology: New Generations, Las Vegas, NV, 2013, pp. 702-705.

[13] J. Erickson and K. Siau, Theoretical and practical complexity of unified modeling language: A Delphi study and metrical analyses. In Proceedings of the International Conference on Information Systems, (pp. 183-194). 2004.

[14] J. Erickson and K. Siau, UML complexity. In Proceedings of the Systems Analysis and Design Symposium, Miami, FL. 2003.

[15] J. Erickson, K. Lyytinen and K. Siau, Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research." JDM 16.4 (2005): 88-100. Web. 7 Jun. 2019.

[16] J. Lewis and K. Neher, Over the Waterfall in a Barrel - MSIT Adventures in Scrum. AGILE 2007 (AGILE 2007). 2007.

[17] J. W. Spence, "There has to be a better way! (software development)," Agile Development Conference (ADC'05), Denver, CO, USA, 2005, pp. 272-278.

[18] K. E. Harper and A. Dagnino, Agile Software Architecture in Advanced Data Analytics. 2014 IEEE/IFIP Conference on Software Architecture. 2014.

[19] K. Siau and L. Lee, Are use case and class diagrams complementary in requirements analysis? An experimental study on use case and class diagrams in UML. Requirements Engineering, 9(4), 229-237. 2004

[20] K. Siau and Q. Cao, Unified modeling language (UML): A complexity analysis. Journal of Database Management, 12(1), 26-34. 2001

[21] K. Siau, J. Erickson and L. Lee, Complexity of UML: Theoretical versus practical complexity. In Proceedings of the 12th Workshop on Information Technology and Systems (WITS), (pp. 13-18). 2002.

[22] K. Siau, J. Erickson and L. Lee, Theoretical versus practical complexity: The case of UML. Journal of Database Management, 16(3), 40-57. 2005

[23] K. V. Jeeva Padmini, P. S. Kankanamge, H. M. N. D. Bandara, and G. I. U. Perera, Challenges Faced by Agile Testers: A Case Study. 2018 Moratuwa Engineering Research Conference (MERCon), 2018.

[24] M. B. Kamuto and J. J. Langerman, "Factors inhibiting the adoption of DevOps in large organisations: South African context," 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, 2017, pp. 48-51.

[25] M. B. Snapp and D. Dagefoerde, The Accidental Agilists: One Team's Journey from Waterfall to Agile, Agile 2008 Conference, 2008

[26] P. Kruchten, "Tutorial: introduction to the Rational Unified Process/sup /spl reg//," Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, Orlando, FL, USA, 2002, pp. 703-.

[27] R. C. Martin, *UML for Java (TM) Programmers*, Prentice Hall, OL3778385M, 2003.
[28] R. Lutowski, *Software Requirements: Encapsulation, Quality, and Reuse*, Auerbach Publications, OL8259867M, 2005
[29] R. M. Haj Hamad and M. Al Fayoumi, "Scalable Agile Transformation Process (SATP) to Convert Waterfall Project Management Office into Agile Project Management Office," 2018 International Arab Conference on Information Technology (ACIT), Werdanye, Lebanon, 2018, pp. 1-8.
[30] R. S. Pressman, "Idealized and actual failure curves for software," In, *Software Engineering: A Practitioner's Approach.* New York: McGraw-Hill, 1987. Print. P.P. 8.
[31] R.C. Martin, Agile Principles, Patterns, and Practices in C#. Prentice Hall. 2006
[32] R.C. Martin, Agile software development: principles, patterns, and practices. Pearson. 2003
[33] S. Cohan and H. Glazer, An Agile Development Team's Quest for CMMI® Maturity Level 5, 2009 Agile Conference, 2009
[34] S. H. VanderLeest and A. Buter, "Escape the waterfall: Agile for aerospace," 2009 IEEE/AIAA 28th Digital Avionics Systems Conference, Orlando, FL, 2009, pp. 6.D.3-1-6.D.3-16.
[35] S. Hermanto, E. R. Kaburuan and N. Legowo, "Gamified SCRUM Design in Software Development Projects," 2018 International Conference on Orange Technologies (ICOT), Nusa Dua, BALI, Indonesia, 2018, pp. 1-8.
[36] T. Gorschek, E. Tempero, and L. Angelis, On the use of software design models in software development practice: An empirical investigation. Journal of Systems and Software, 95, 176–193. 2014.
[37] W. M. D. Ruchira Prasad, G. I. U. Perera, K. V. Jeeva Padmini, and H. M. N. Dilum Bandara, Adopting Design Thinking Practices to Satisfy Customer Expectations in Agile Practices: A Case from Sri Lankan Software Development Industry. 2018 Moratuwa Engineering Research Conference (MERCon). 2018.

I. D. Arroyo is a Master in software engineering and artificial intelligence and a PhD student in Informatics Technologies at the University of Malaga; he has worked as a project leader, technology department leader, and software architect in different public and private companies, with an experience of 13 years in the industry.