Vázquez-Ingelmo, A., García- Holgado, A., & García-Peñalvo, F. J. (2020). C4 model in a Software Engineering subject to ease the comprehension of UML and the software development process. In 2020 IEEE Global Engineering Education Conference (EDUCON), (27-30 April 2020, Porto, Portugal) (pp. 919-924). IEEE.

C4 model in a Software Engineering subject to ease the comprehension of UML and the software development process (*)

Andrea Vázquez-Ingelmo
GRIAL Research Group
University of Salamanca
Salamanca, Spain
andreavazquez@usal.es

Alicia García-Holgado
GRIAL Research Group
University of Salamanca
Salamanca, Spain
aliciagh @usal.es

Francisco J. García-Peñalvo
GRIAL Research Group
University of Salamanca
Salamanca, Spain
fgarcia@usal.es

***Abstract*—Software engineering provides the competences and skills to design and develop robust, secure and efficient applications that solve real problems. Students have to develop their abstract thinking to find solutions taking into account not only technical development, but economic and social impact. In previous years, different changes have been introduced in the teaching methods with significant outcomes. However, students are still facing difficulties with one of the core contents of the subject, UML. For this reason, the present work aims to introduce C4 model as a complement of the existing UML diagrams. This proposal uses the two first levels of the C4 model to complement the requirements elicitation process, traditionally based only on use cases, to let students start the design of their systems without going into greater technical details.**

Keywords—UML, software architecture, software documentation, C4 model, software engineering, abstraction.

I. INTRODUCTION

Software Engineering I is a mandatory subject of the Degree in Computer Science at the University of Salamanca (Spain). This subject addresses the first activities of the software development process. Through its contents, students learn how the early stages of the life cycle of information systems are carried out, focusing on their definition, planning and analysis.

Software Engineering I is the students' first look in the degree at systematic approaches to manage and develop information systems. Students take the course in the second semester of the second year. At this point, they have acquired programming and computational skills. However, when students take this subject, they face a different way of approaching software development, because software engineering needs higher levels of abstraction. For these reasons, the subject is perceived as difficult and hard to understand by the students.

This discipline is also a challenge for teachers, as they need to introduce new methods to engage students with the subject and to foster knowledge acquisition.

In previous years, different changes have been introduced in the teaching methods, like Project-Based Learning (PBL) and active methodologies with significant outcomes [1, 2]. However, based on the learning outcomes and feedback collected during the last 4 years, students are still facing difficulties with one of the core contents of the subject: The Unified Modeling Language (UML).

UML is a modeling language that supports the specification, visualization, construction and documentation of information systems [3, 4]. Although UML syntax is divided between two subjects, the wide syntax of this language, the variety of diagrams, the programming background of the students, among other factors, makes its learning a challenge [5, 6]. Besides, through informal conversations, year by year, the students express that software engineering a useless subject, something that fancy startups do not use. Most of them think that UML is something old, not necessary to define and develop software.

The purpose of UML is to rely on a unified syntax for communicating from requirements to design decisions with the goal of being understood by the involved stakeholders. One of the main difficulties faced by students when studying this language is the gap that they perceive between the systems they model throughout the subject and the systems they have actually coded throughout past subjects.

George Fairbanks named this issue the "model-code gap" [7]. In [7], this gap is described through the differences among different aspects present in models and code: the vocabulary employed in models (modules, components, protocols, associations, etc.) in contrast with the vocabulary employed in code (packages, classes, variables, functions, etc.), the high levels of abstraction in models (especially at the analysis phase of the software development process) in contrast with the concrete and specific nature of source code, among other factors that enlarge this gap.

The model-code gap threatens the understanding of UML because students need to shift their minds from concrete code towards abstract models, and the main problem is that they often do not see the relationship (and thus, the utility) between these low-level and high-level perspectives of information systems.

A variety of works in the literature have tried to address this issue through different methodologies; for example, by using a combination of UML and OCL [8], by adapting materials to the audience [5], by employing interactive exercises to visually understand the implications of UML diagrams [9], or even by teaching modeling languages before programming languages [10].

On the other hand, some active methodologies, such as project-based learning (PBL), try to ease the comprehension of this subject by fostering the involvement of students through the development of projects that are close to real-world context. These methodologies have the goal of raising the motivation of students regarding such a complex subject,

(*) This work has been partially funded by the Spanish Government Ministry of Economy and Competitiveness throughout the DEFINES project (Ref. TIN2016-80172-R)

and to increase the utility perception of software engineering in real-life scenarios [11, 12].

For these reasons, this paper proposes the introduction of a new approach for documenting software as a complement of UML in the subject of Software Engineering I: the C4 model [13]. This model is already being employed to describe software architectures in real contexts [14-16].

The C4 model proposes four levels of abstraction: context, containers, components and code. Each information system is composed by these elements and persons (i.e., users that interact with the software system). With this model, information systems are divided in more manageable parts that can be better analyzed. Its simplified syntax and abstraction could be a benefit for students to understand how software systems are designed and built.

The rest of the paper is organized as follows. Section II explains how the subject introduces and works with UML and the main issues students face with this language. Section III outlines the syntax of the C4 model. Section IV describes the methodology followed in the subject. Section V presents the actions proposed to introduce the C4 model, while section VI illustrates different proposals of the C4 model in the context of the subject. Finally, Section VII summarizes the main conclusions of the proposal.

II. Context

Software Engineering I covers the definition, planning and analysis of information systems. The whole course is driven by milestones following the software development process; in this particular case, milestones are requirements elicitation, domain model and use-case realization.

The sessions in which UML is explained are structured following this model. First, use-case diagrams are presented to lay the foundations of requirement elicitation. Then, class diagrams are explained at analysis level, meaning that classes should represent domain entities and their relationships at a high-level, omitting implementation-level details. Finally, sequence and communication diagrams are introduced.

These sessions are followed by workshops in which real-world problems are tackled and documented through the presented UML diagrams, putting special emphasis in class diagrams to foster abstract thinking.

Moreover, students must apply the acquired knowledge to analyze and document their own information system, which constitutes their final project. This project covers only two phases, requirement elicitation and requirement analysis (domain model, interaction view and architectural proposal). The other phases of the software engineering process are studied in Software Engineering II during the third year of the degree.

For this project, the teachers propose a generic goal that the system must accomplish, and students have the freedom to design any type of system that lets users reaching the goal, following the software development process and using UML diagrams to document each phase.

One of the main issues students face is the conceptualization of the system. The project's requirements give the students freedom to choose how to address the stated problem, but they must think carefully about them because these requirements are the drivers of the whole project. If this first conceptualization phase is poorly carried out, it could affect the rest of the phases, given the incremental nature of the project deliveries and the relationships among the different milestones.

In fact, the majority of problems and mistakes increase during the second phase of the project, in which students must develop the domain model of their solution. Switching from use-case diagrams (at the requirements elicitation phase, the first milestone of the project) to class diagrams (at the domain model development phase, the second milestone of the project) seems to be a complex step for students.

The most common mistakes found during this phase are related to the required abstraction levels of the domain model. Students usually focus on functionality instead of on the structure of the domain, introducing several modeling errors in their UML class diagrams.

During the first milestone students are focused on use-cases, which are more related to what they have learning through programming subjects (i.e., offering services or some functionality to end users). However, there are also problems at this stage, because students usually end up modeling an interface for their solutions (including buttons, text inputs, etc.) more than the functionality pieces of their systems.

The C4 model offers a syntax that could make the conceptualization and abstraction of a system's functionality an easier and less convoluted task.

III. C4 Model

The C4 model defines software systems through four views. Each of these levels of the C4 model is focused on a certain perspective of the system. The syntax is simple and flexible. In this section, the notation of this model is outlined based on the work of Simon Brown [13], in order to contextualize the next sections.

The first level (context) focuses on framing the software system to be modeled by representing the different persons or actors involved and external systems that provide any kind of service to our system. The interactions among components are represented by dashed arrow lines annotated by a description withe the role of the relationship. External or already implemented services are colored with different from the target software system. At this level, further details regarding the involved software systems are omitted, providing a high-level view of the context in which the target system will be framed.

The second level decomposes the software system in containers. This view outlines the necessary components to provide the services that the system would offer. Again, further details such as the internal structure of the components are omitted, providing only a functional description of each container and the relationships among them.

Containers are in turn formed by components, which is the perspective in which the C4 model focuses on the third level. The collaboration between the different components must provide the service or functionality of the container described at the second level of the C4 model. A legend that outlines the main notation elements of the C4 model can be consulted in Fig. 1.

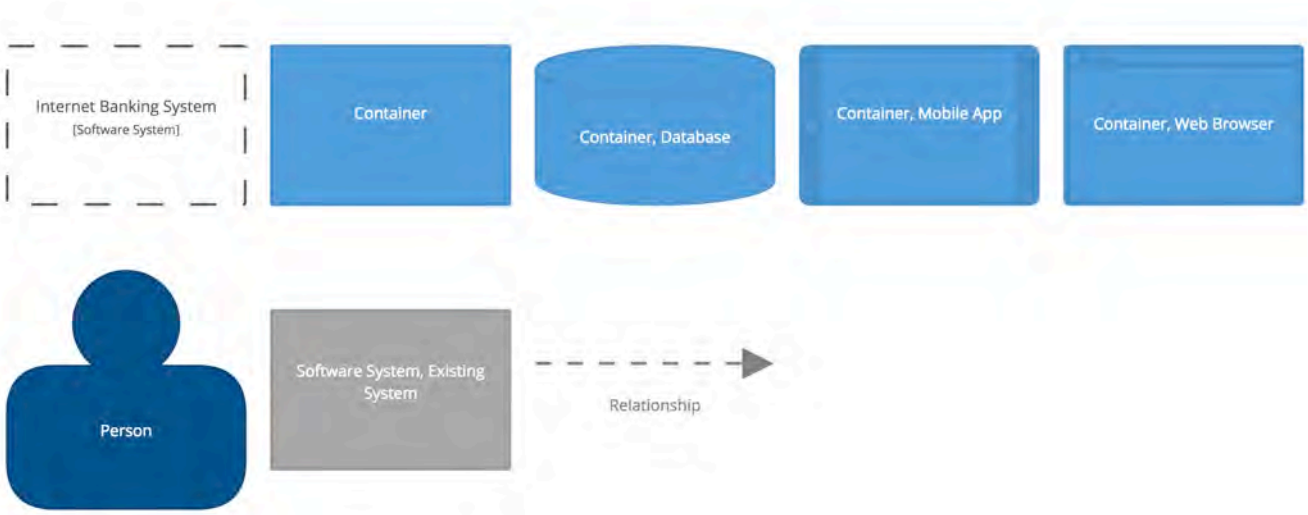

Fig. 1. Notation of the elements that compose the C4 model. This specific notation is used by Structurizr (https://structurizr.com/), a documentation tool by Simon Brown that supports the creation of C4 model diagrams. Source: Simon Brown [17].

Finally, the fourth level (code-level) can be specified through UML diagrams, such as class diagrams, sequence diagrams, components diagrams, etc.

One of the strengths of the C4 model is the continuous use of explicit descriptions in every involved element within the diagrams, avoiding the ambiguity that can be introduced through UML given its elaborated syntax.

The notation of the C4 model is not restricted to a set of strict rules. In fact, the notation is more concerned in dividing the system in the mentioned four levels and in developing comprehensible diagrams than in standardizing a syntax. If any element or notation needs to be adapted to the domain of application, as long as it is well defined, there isn't any restriction to do it [17].

IV. Methodology

The methodology is based on the active learning methodology implemented through cooperative and collaborative learning, an approach of flipped classroom [18, 19] for the theorical contents, and Project-Based Learning (PBL) [20-22] that were already introduced in the Software Engineering I subject during the 2017-18 course [23]. The proposed actions of the current work will be framed within the active learning methodology. In particular, as part of the activities associated to the final project in which students work in teams from the beginning of the course in order to achieve a set of milestones related to the software engineering process.

Regarding the study case monitoring and impact, a satisfaction questionnaire to get the students' opinion about the implemented measures is employed. Authors have used a satisfaction questionnaire published as an annex in the doctoral dissertation "Evaluation of the impact of a teaching methodology, based on active student learning, in computing in engineering" by González Rogado [24]. This questionnaire was modified to hold specific questions regarding the introduction of the C4 model.

V. Actions

The introduction of the C4 model approach for documenting software architectures is carried out at the beginning of the course (i.e., at the requirements phase).

At the requirements elicitation phase, students are asked to identify the main requirements and users of their final project through use-case diagrams. This phase is a challenge for them, because they need to understand the goal and formalize a set of features to reach it. Moreover, they try to define a set of requirements instead a holistic overview of the system that they want to develop to solve the proposed problem.

The simplified syntax of the C4 model is set to help them identify main actors, external systems and internal components, and as a support to perform brainstorming regarding their system's requirements and goals.

The two first levels of the C4 model (i.e., system context and container levels) are employed, as they let students design their systems without going into greater technical details. Besides, although these models have a high abstraction level, are pretty near to real world, in which students can identify even mobile apps or websites as elements connected with other tools and different users.

At the end of the semester, when students have developed their final project, they are asked to do another C4 model of the final system. This C4 model should be similar to the one developed at the beginning, but with the specific components that are part of their solution.

VI. C4 Model Application Example

As introduced in the methodology section, the Software Engineering I subject previously implemented an active learning methodology based using Project-Based Learning (PBL). Students that chose this active modality are involved in the final project since the beginning of the semester.

During the 2018-2019 course, the final project consisted in designing a system that accomplish the goal of promoting diversity in business contexts. Students needed to identify a set of requirements the system should have to reach the goal and model them to obtain an analysis model.

A great variety of solutions were proposed; from hiring apps that help employers to avoid bias when seeking for employees to information repositories to foster awareness.

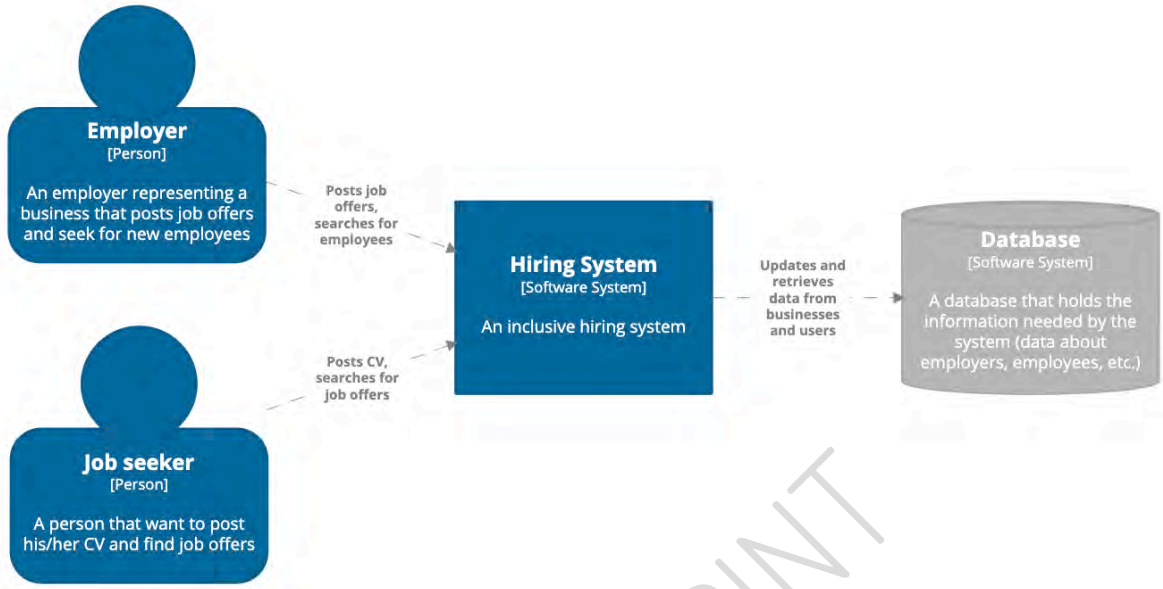

Fig. 2. Context view of a proposed inclusive hiring system using the C4 model.

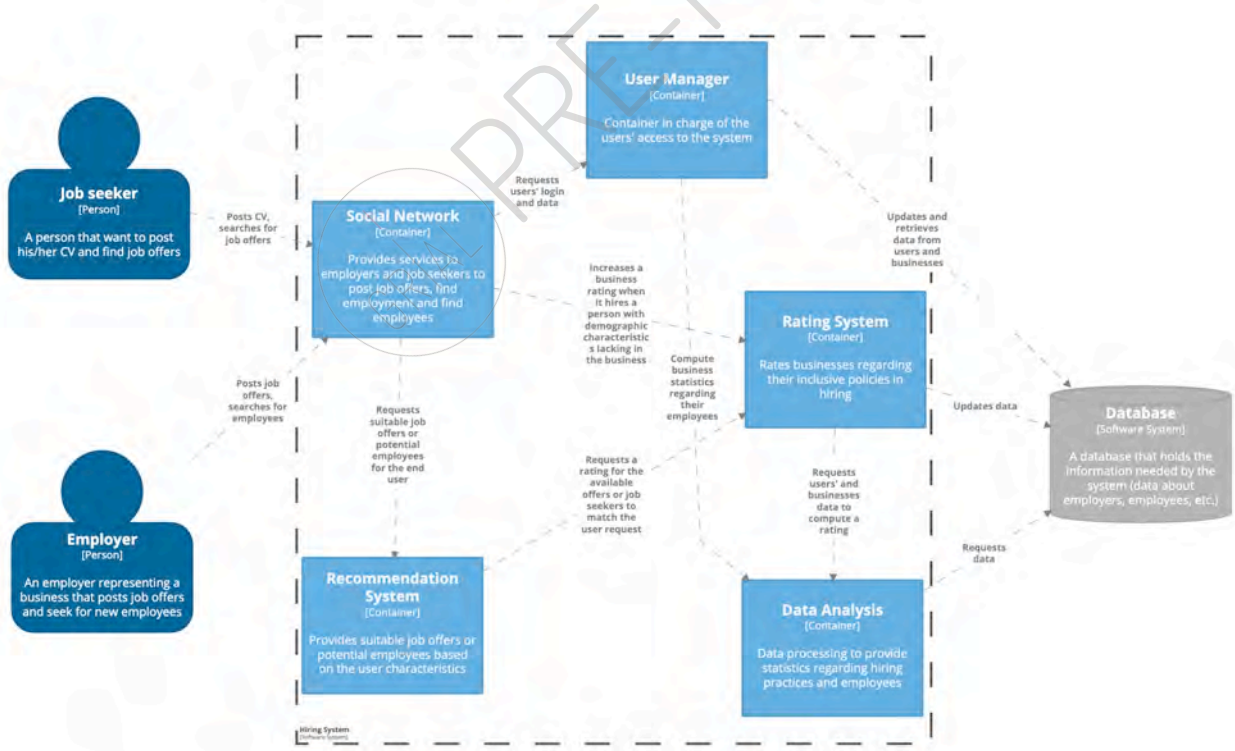

Fig. 3. Container view of a proposed inclusive hiring system using the C4 model.

However, although the ideas fulfilled the project requirements, students had trouble documenting them with UML, as detailed in Section II.

Students were not totally able to recognize the conceptual classes of their domains and relate them to their previously identified functional requirements through use-case diagrams.

Given that the C4 model combine the static structure of the system with its functionality through its different hierarchical levels, it might ease these difficulties encountered at the first phases of the project development.

Figures 2 and 3 shows one of the proposed solutions for the final project. Students designed a system that rates businesses based on the proportions of employees that belong to some demographic group. The rating increases if the business shows balanced proportions, so it increases if the business hires people that belong to minorities or discriminated groups.

The conceptualization phase is complex, and students encounter a lot of barriers that are drawn across the rest of the milestones if conceptualization issues are not solved at the beginning of the course.

The C4 model can assist students with this task and help them to identify the main functionalities each container must have, thus making the process of developing the use-case diagrams more straightforward.

For example, this model could help students with the organization of functionalities at the requirements elicitation phase. In this case, the use-case diagram shown in Figure 4 details the functionality of the user management container outlined in Figure 3.

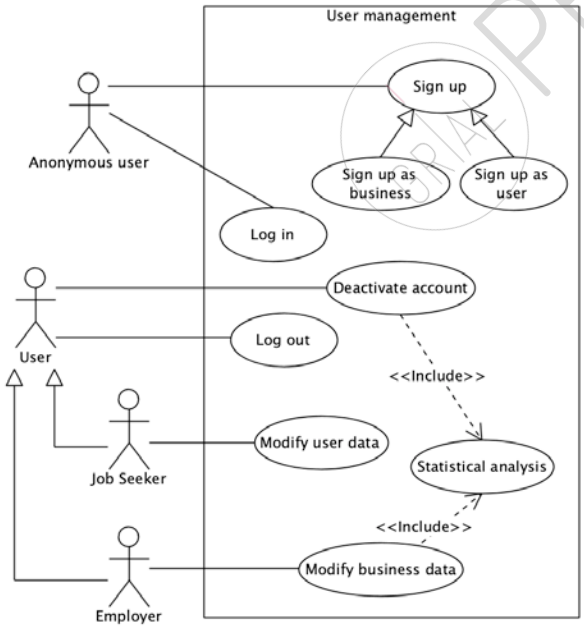

Fig. 4. Use-case diagram for the proposed hiring system's user management.

Also, by detecting the relationships among containers at the first stage of the project, students can rely on these associations and the information requirements of each container to build their domain model.

Finally, as explained before, the third milestone of the final project is focused on the development of interaction diagrams (specifically, sequence diagrams). These can be easily added to the C4 model at its last level (code-level), relating them to the specific use-cases that a container holds and providing a whole and organized view of the system.

For example, Figure 5 shows a sequence diagram for the use-case named "modify business data".

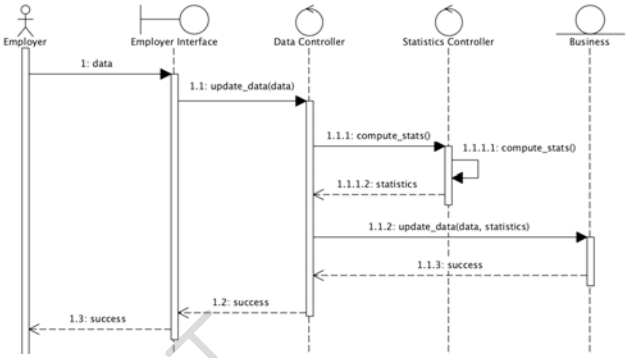

Fig. 5. Sequence diagram that represents the modification of business data within the proposed hiring system.

The C4 model and its different hierarchical levels, promote the comprehension of the software architecture and reduces the model-code gap. Moreover, the hierarchical structure of the C4 model allow "zoom-in and zoom-out" on systems' details.

Teaching how to employ this model along with UML could enhance the learning outcomes and increase the motivation regarding the subject.

VII. Discusion and Conclusions

The main goal of this case study is to raise the students' perception regarding software documentation and to ease the comprehension of UML.

With this approach, students work with a methodology closer to businesses contexts, that can be understood by the stakeholders [25] by outlining the whole system as sets of containers and components with specific functionalities and goals.

This methodology is also closer to the reality of how systems are being currently developed (e.g., distributed components, communication through API calls, services held in different servers, etc.) [26, 27], providing a high-level overview of the organization and information flows of the different system's components.

However, the C4 model does not replace the UML language, it wraps and refactors it in more manageable pieces. In fact, as mentioned in others sections of this work, the fourth level of the C4 model (i.e., code-level), can be represented through UML class diagrams or similar (https://c4model.com/#coreDiagrams).

So, in this case, the C4 model can be seen as a complement to ease the comprehension of UML diagrams and the software development process.

This proposal will be applied in the forthcoming years. The analysis of the learning experience will be carried out in the long-term in order to rely on meaningful samples from different academic years.

The satisfaction questionnaire is employed to validate the utility and acceptation of the C4 model as a support to understand and ease the comprehension of the first phases of the software process.

REFERENCES

[1] A. García-Holgado, F. J. García-Peñalvo, and M. J. Rodríguez Conde, "Pilot experience applying an active learning methodology in a Software Engineering classroom," in *2018 IEEE Global Engineering Education Conference (EDUCON), (17-20 April 2018, Santa Cruz de Tenerife, Canary Islands, Spain)*USA: IEEE, 2018, pp. 940-947.

[2] A. García-Holgado, F. J. García-Peñalvo, M. J. Rodríguez-Conde, and A. Vázquez-Ingelmo, "El campus virtual como soporte para implementar una metodología activa para mejorar la tasa de éxito en la materia de Ingeniería del Software," in *Actas IX Jornadas Internacionales de Campus Virtuales, 11-13 Septiembre 2019*Popayán, Colombia: Asociación Red Universitaria de Campus Virtuales (RUCV), 2019.

[3] P. Stevens and R. Pooley, *Using UML: software engineering with objects and components*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[4] A. Van Lamsweerde, *Requirements engineering: From system goals to UML models to software*. Chichester, UK: John Wiley & Sons, 2009.

[5] S. Moisan and J.-P. Rigault, "Teaching object-oriented modeling and UML to various audiences," in *International Conference on Model Driven Engineering Languages and Systems*, 2009, pp. 40-54: Springer.

[6] K. Siau and P.-P. Loo, "Identifying difficulties in learning UML," *Information Systems Management,* vol. 23, no. 3, pp. 43-51, 2006.

[7] G. Fairbanks, *Just enough software architecture: a risk-driven approach*. Marshall & Brainerd, 2010.

[8] L. Burgueño, A. Vallecillo, and M. Gogolla, "Teaching UML and OCL models and their validation to software engineering students: an experience report," *Computer Science Education,* vol. 28, no. 1, pp. 23-41, 2018.

[9] S. Frezza and W. Andersen, "Interactive Exercises To Support Effective Learning of UML Structural Modeling," in *Proceedings. Frontiers in Education. 36th Annual Conference*, 2006, pp. 7-12: IEEE.

[10] C. Starrett, "Teaching UML modeling before programming at the high school level," in *Seventh IEEE International Conference on Advanced Learning Technologies (ICALT 2007)*, 2007, pp. 713-714: IEEE.

[11] M. Daun, A. Salmon, T. Weyer, K. Pohl, and B. Tenbergen, "Project-Based Learning with Examples from Industry in University Courses: An Experience Report from an Undergraduate Requirements Engineering Course," in *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, 2016, pp. 184-193.

[12] F. Llopis and F. G. Guerrero, "Introducing competitiveness and industry involvement as learning tools," in *2018 IEEE Global Engineering Education Conference (EDUCON)*, 2018, pp. 298-307.

[13] S. Brown, "Software Architecture for Developers—Volume 2: Visualise, document and explore your software architecture," ed: Ebook, 2017.

[14] T. Masood and J. Egger, "Adopting augmented reality in the age of industrial digitalisation," 2019.

[15] J. Silva *et al.*, "An Online Platform For Real-Time Air Quality Monitoring," in *2019 5th Experiment International Conference (exp. at'19)*, 2019, pp. 320-325: IEEE.

[16] A. Vázquez-Ingelmo, A. García-Holgado, F. J. García-Peñalvo, and R. Therón, "Dashboard Meta-Model for Knowledge Management in Technological Ecosystem: A Case Study in Healthcare," in *UCAmI 2019*, Toledo, Castilla-La Mancha, Spain, 2019, vol. 31, no. 44: MDPI.

[17] S. Brown. (2018). *The C4 Model for Software Architecture*. Available: http://c4model.com/

[18] E. M. Choi, "Applying Inverted Classroom to Software Engineering Education," *International Journal of e-Education, e-Business, e-Management and e-Learning,* vol. 3, no. 2, pp. 121-125, 2013.

[19] G. C. Gannod, J. E. Burge, and M. T. Helmick, "Using the Inverted Classroom to Teach Software Engineering," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 777-786.

[20] V. Estruch and J. Silva, "Aprendizaje basado en proyectos en la carrera de Ingeniería Informática," presented at the Actas de las XII Jornadas de la Enseñanza Universitaria de la Informática, JENUI 2006, Deusto, Bilbao, 2006.

[21] L. Helle, P. Tynjälä, and E. Olkinuora, "Project-based learning in post-secondary education–theory, practice and rubber sling shots," *Higher Education,* vol. 51, no. 2, pp. 287-314, 2006.

[22] J. A. Macias, "Enhancing Project-Based Learning in Software Engineering Lab Teaching Through an E-Portfolio Approach," *IEEE Transactions on Education,* vol. 55, no. 4, pp. 502-507, 2012.

[23] A. García-Holgado, F. J. García-Peñalvo, and M. J. Rodríguez-Conde, "Pilot experience applying an active learning methodology in a Software Engineering classroom," in *2018 IEEE Global Engineering Education Conference (EDUCON), (17-20 April 2018, Santa Cruz de Tenerife, Canary Islands, Spain)*USA: IEEE, 2018, pp. 940-947.

[24] A. B. González Rogado, "Evaluación del impacto de una metodología docente, basada en el aprendizaje activo del estudiante, en computación en ingenierías," Departamento de Didáctica, Organización y Métodos de Investigación, Universidad de Salamanca, Salamanca, España, 2012.

[25] B. Dobing and J. Parsons, "How UML is used," *Communications of the ACM,* vol. 49, no. 5, pp. 109-113, 2006.

[26] M. Kassab, M. Mazzara, J. Lee, and G. Succi, "Software architectural patterns in practice: an empirical study," *Innovations in Systems and Software Engineering,* vol. 14, no. 4, pp. 263-271, 2018.

[27] C. Pahl, I. Fronza, N. El Ioini, and H. Barzegar, "A Review of Architectural Principles and Patterns for Distributed Mobile Information Systems."