A Machine Learning Approach for Continuous Development

Daniel Russo[1,2,3], Vincenzo Lomonaco[1], and Paolo Ciancarini[1,2,3]

[1] University of Bologna, Department of Computer Science & Engineering
Mura Anteo Zamboni, 7. 40126 Bologna, Italy.
[daniel.russo;vincenzo.lomonaco;paolo.ciancarini]@unibo.it,
[2] Italian National Research Council, Institute of Cognitive Sciences and Technologies, CNR–ISTC.
[3] Consorzio Italiano Nazionale per l'Informatica.

Abstract. Complex and ephemeral software requirements, short time-to-market plans and fast changing information technologies have a deep impact on the design of software architectures, especially in Agile/DevOps projects where micro-services are integrated rapidly and incrementally. In this context, the ability to analyze new software requirements and understand very quickly and effectively their impact on the software architecture design becomes quite crucial. In this work we propose a novel and flexible approach for applying machine learning techniques to assist and speed-up the continuous development process, specifically within the mission-critical domain, where requirements are quite difficult to manage. More specifically, we introduce an Intelligent Software Assistant, designed as an open and plug-in based architecture powered by Machine Learning techniques and present a possible instantiation of this architecture in order to prove the viability of our solution.

1 Introduction

Software design can be partially considered as a decision making process, where the architect translates the requirements into an architecture [4]. Therefore, the elicitation and formulation of the "User Requirements" is well known to be one of the most critical phases in an engineered software system. Before design, indeed, we need to fully understand the users' point of view, aiming at satisfying their needs and the expected quality of user experience (UX). At the end, software design is not as much about building a system which is technically perfect as one which is fully compliant with the customer's expectations [24]. Even though, during the past, automated frameworks which allow architectural languages [16] and decision-centric architecture design methods [10] have been extensively studied, very little has been done for practically assisting the continuous development and design processes. Generally speaking, we support epistemological innovation to pursue research goals in software engineering, like [7], [8].

In this work we propose novel approach for assisting the continuous development process through algorithmic methods which are able to *learn* from experience, that is according to previous Agile/DevOps iterations as described in [18]. At the best of our knowledge we are not aware of any relevant research in this direction. Indeed, even

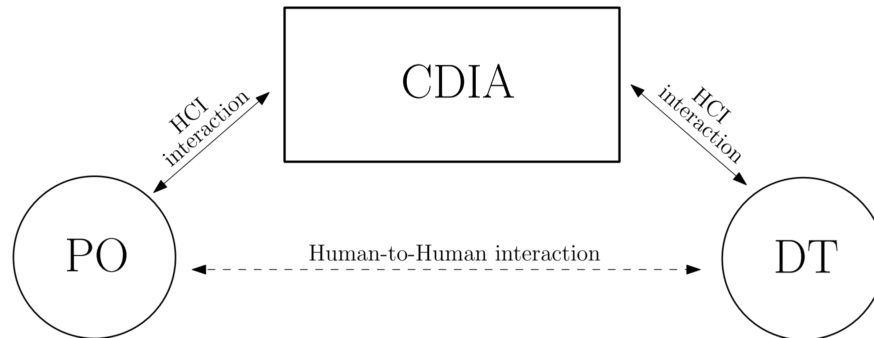

Fig. 1. CDIA: the Continuous Development Intelligent Assistant

if previous scholars already explored assistance frameworks (like [2]), none of them employed Machine Learning techniques aimed to automatize them. Other task-focused approaches (e.g., requirements prioritization) have been carried out [1], [20] but without a comprehensive approach with respect to the continuous development process or considering third party integration [6] and their data quality [9]. Our goal instead, is to improve developers' productivity, and increase software artifacts value (in terms of how much functionality they deliver) by automatizing the requirements analysis and assisting the continuous development process in a comprehensive way.

Velocity is also a key issue for the mission–critical domain which has the urgency to deliver fast safety–critical functionalities. The use of Machine Learning techniques for predicting and summarizing useful information regarding the architectural design and the impact of new requirements on the software code base is here essential to accelerate the entire process and allowing the Agile/DevOps team to rapidly transform the model into code.

Software architecture in the *Digital Age* and the role of the architect is undergoing a deep rethinking [14], [11]. The evolution and challenges of software architecture opened the door to Agile/DevOps methodologies as crucial asset to leverage continuous development and architecting [17]. In fact, the urgency to continuously modify systems designs leads to new approaches. The aim of this article is to show how a new Machine Learning approaches in Agile/DevOps development can also support the continuous development (providing useful hints to the Developer Team) along with the analysis of systems requirements.

In this paper, we present the approach developed in a real working case study within a governmental Agency (from now on "Agency") which develops mission-critical applications, where an intelligent software assistant has been designed for (i) the requirements comprehension and analysis; (ii) providing useful information with respect to the software design; (iii) predicting the impact of new requirements on the development process and the code-base within an Agile/DevOps customized methodology.

The paper is structured as follows. In Section 2 we explain the context in which we are developing our approach and motivate why solving this issue is crucial. Moreover, the problem and the solutions are outlined along with an abstract representation of our

working solution. In Section 3 the formal model is presented: the architecture is designed to be open and incremental, in order to add new machine learning models and refine their interactions. To convince the reader about the viability of our approach, we show a possible instantiation in Section 4. Finally, in Section 5 we summarize our work and discuss some extensions we plan to add in the near future.

2 Problem definition

Continuous software engineering is more than adopting continuous delivery and continuous deployment: the goal is to take an holistic view of a software production entity [12]. Empowering developers with an Intelligent Assistant is considered by the Agency as a viable solution to manage the fast-changing scenario of its daily operations. The Agency has strict constraints to develop and deploy mission-critical software in a fast way, since the operational scenarios it has to face change rapidly. Security and resilience is also a great issue, this is the reason why they are experimenting new antifragile frameworks [27], [25]. Satisfying changing users' needs is one of the top priorities of the IT department, and optimizing the continuous development processes is vital for the fulfillment of its mission. A major problem repeatedly observed during this phase is the inability of the development team (DT) to understand the language and the context in which some requirements are described by the user and to follow good architectural patterns along with the fast system evolution. A lot of effort and a number of different approaches have emerged in order to deal with RE within the Scrum process. At the beginning, an effective technique to understand requirements was to to write down user stories in order to fix the scope of the requirements. One of the most important devices supporting agile developments has been achieved by persuading the users to define their requirements by a number of "user stories" which become a sort of domain specific jargon that can be understood by both parties. However, users (Product Owners) tend to use the same "jargon", due to organizational routines [19]. The Agency refined the traditional user story structure into a customized one: `As <role> I want to <functionality description> in order to <goal to pursue>.`

Nevertheless, misunderstandings are still very common during Agile/DevOps and mission critical development, especially during the first cycles, where developers are usually unaware of the application context [27].

During the last years Knowledge Management Systems (KMS) and Data Mining techniques have made their appearance in this context in order to extract and relate semantic knowledge from user stories, hence facilitating the requirements engineering phase through disambiguation [28]. However, we argue that these techniques are still very unripe and uncorrelated, without a clear understanding of their improvement directions and future applicability. Furthermore, we remark that requirements disambiguation is just a single aspect of the continuous development process, which we try to improve with a uniform but flexible solution.

We envision a single software system that can take part in the continuous development process acting as a proficient assistant and interpreter who speaks the languages of both the users and the developers (see Fig.1). The disruptive idea is that this complex piece of software would not be a simple tool to analyse and correlate user stories, but

it would offer useful *predictions* learning continuously from previous interaction cycles as shown to be fruitful in many other application contexts [21], [22], [23].

The key factor here is the ability to *learn* from the past, exactly like a human software engineer would do and offers great insights during the continuous development processes that are *specific of the software which has been developing*. A software envisioned in this way, not only offers direct insights on what and how disambiguate some requirements, but can also make faithful predictions about the design and development processes (e.g., micro-services dependences, work/hours to commit, the price to pay, the number of code lines to change). Indeed, if we assume that there is an recognizable pattern among some requirements topics or typologies and the amount of work or services dependencies which can satisfy these requirements, then a statistical model would probably be able to capture it and such information would result in an extremely valuable asset for planning the development cycle ahead.

3 Model formalization

In this section we provide a formal model architecture which defines the structural properties and the operational modalities of a Continuous Development Intelligent Assistant (CDIA). We propose an open model extensible in a plug-in fashion along with a possible instantiation.

First of all, let us define the time factor as a variable T where we indicate a specific point in time as t_i with $i \in [0, \dots \infty)$ (zero stands for the starting time of the development).

Then, let us denote a user story as s and a set of user stories as S. In our model we assume for simplicity that the requirements are defined by user stories and at each development iteration they came together as single set (or batch) of arbitrary size. More formally, we can enumerate the set using S_j with $j \in [0, \dots, \infty)$, where 0 is the first batch of requirements commissioned by the user. Note that each set can be of different size. For simplicity, as often performed discrete-event simulation models (DES), we represent time as a discrete variable which varies only when a new batch of user stories arrives i.e., $i = j$.

For each story that has been proposed by the user $s \in S$ we should also keep track of the final and agreed user story that has been refined after a few feedback from the software assistant or external consultations. We will refer to them as s^r where r stands for *refined*. Note that we have a one to one connection for each s and s^r even if the story hasn't been changed or has been dismissed (in this case $s = s^r$).

Each story s is defined by a series of attributes: let us use a function named $attr()$ that given a story s return its attribute. Note that $|attr(s)| = k$ with $k \in \mathbb{N}$, and k is the same for each s. We need also a number of attribute which can describe the state of the software at each development iteration (let us name it D_j). We can use the same function $attr()$ defined before but in this case it accept as input the software state D_j at time j, where $|attr(D_j)| = z$ with $z \in \mathbb{N}$. Note that the more attributes $attr(s)$ and $attr(D)$ we insert in the model the more accurate may be the prediction.

As for the last essential step we can not bypass in our CDIA formalization, we need to keep track of the inter-dependences among services and micro-services which

constitutes the functionalities of the developed software. We define the set of services V_i with i varying with the development iterations. Let us also use a function named $dep()$ that given a set of services V_i return the dependences among them.

Now that we have all formal environment in place we can formulate the main CDIA system as a series of plugins whose results combination can produce two different evaluation feedbacks, one for the Product Owner (PO) and one for the Developer Team (DT) in order to assist the continuous development process as depicted in Fig.2. In this work, we describe three main plugins (defined as Machine Learning models):

– **Services Dependences Tracking Plugin (SDTP)**: This machine learning model learns the relationships between services and requirements (in this case the user stories and services $\{(S_0,\ldots,S_{i-1}),\ (V_0,\ldots,V_{i-1})\}$). Then, at iteration i (i.e time t_i), given a new batch S_i returns a feedback to the development team (DT) regarding the suggested changes among the services inter-dependences or the eventual insertion of new services, actually guiding the continuous development process. More formally, we would like to learn a set of parameters θ of a function d, such that:

$$n, dep(V_i) = d(\theta, attr(S_i)) \tag{1}$$

  That is predicting all the new dependences among the services after the implementation of the requirements S_i and eventually suggesting the introduction of a number of n new services. Note that in this case we apply the function $attr$ to the entire batch of user stories S_i meaning that we compute $attr(s)$ for each s and then we aggregate the results.

– **Development Changes Impact Plugin (DCIP)**: This machine learning model at iteration i (i.e time t_i) learns the impact on the development phase of accepted user stories $\{(S_0,\ldots,S_{i-1}),\ (D_0,\ldots,D_{i-1})\}$ and, given a new batch S_i returns a feedback to the development team (DT) regarding the predicted changes impact on the software (a more general introduction to this approach can be found in [15], [3], [29]). More formally, we would like to learn a set of parameters θ of a function c, such that:

$$attr(D_i) = c(\theta, attr(S_i)) \tag{2}$$

  That is predicting all the attributes that we expect the software to have after the implementation of the requirements S_i. Note that even in this case we apply the function $attr$ to the entire batch of user stories S_i.

– **User Stories Disambiguation Plugin (USDP)**: This machine learning model at the development iteration i (i.e time t_i) learns from previous proposed and accepted user stories $\{(S_0,\ldots,S_{i-1}),\ (S^r_0,\ldots,S^r_{i-1})\}$ and, given a new proposed set S_i, returns a feedback to the customer (Product Owner or PO) regarding the possible changes to apply it in order to minimize its ambiguity (for this plugin we took inspiration from [13]). The more development iterations the software goes, the more accurate the software assistant becomes. So, more formally, we want to learn a set of parameters θ of a function f, such that:

$$s^r = f(\theta, s) \tag{3}$$

  In this way we can then predict the corresponding s^r given a new s which may have never been seen before. Another possibility, more naive but still powerful would be

to learn a set of parameters θ of a function g, such that:

$$p(A|s) = g(\theta, attr(s)) \quad (4)$$

that is returning the acceptance probability $p(A)$ given the submitted user story, along with some hints about the motivation (hidden in the structure of f).

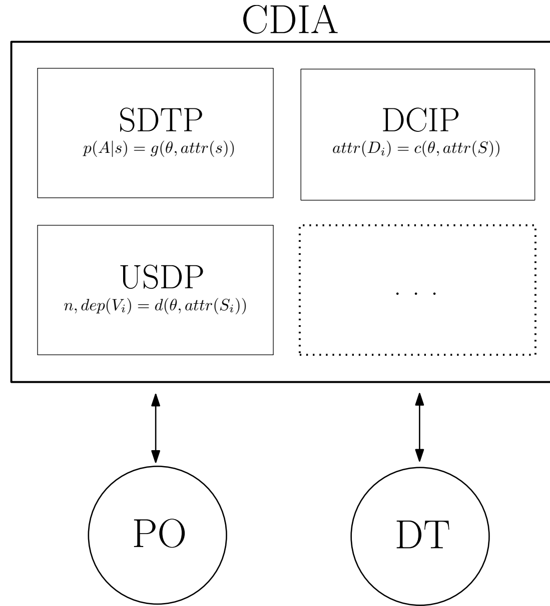

Fig. 2. The Continuous Development Intelligent Assistant Design

The USDP and DCIP plugins are instrumental to the SDTP pluging, which can suggest useful insights regarding the architectural changes (in terms of micro-services) based on a new set of requirements committed by the Product Owner. Indeed, even though the SDTP pluging, seems to be the most valuable in terms of assistance to the continuous development process, a good prediction can not came without a profound understanding about the actual requirements and how they effect the code which run these micro-services.

4 Instantiation

In order to convince the reader about the feasibility of this approach, let us now formulate a possible instantiation of the formal model.

For the user story attribute we can define $attr(s)$ as:

- target user class

- length of the story (number of characters)
- number of atypical words

Then, for the STDP plugin we can choose to represent the services in V_i and their relationships as a directed graph in which the nodes constitutes the services and the directed edges the dependencies of one service to another. So that, if node (i.e. service) a as a directed edge towards b we can say that the service a depends from b. With this formulation the function $dep(V_i)$ can be instantiated simply as the connection matrix (also called *adjacency matrix*) of the directed graph. Then, we can instantiate d as a multivariate regression function, using a two layers (or more) artificial neural network (ANN), where the output nodes are n^2+1 with $|V_i| = n$, meaning that we are trying to predict the value of each edge in the current connection matrix, plus one real number which is the expected number of new services to be introduced at time i. However, we are aware that a larger number of (possibly semantic) attributes may be needed especially in the case of more complex projects.

For the DCIP plugin we need first to define a set of aggregate attribute with can best represent an entire batch of user stories. For simplicity, based on the only three attribute we have defined for each user story, a possible instantiation for $attr(S)$ could be:

- Number of user stories
- Number of different User Classes
- Average length of the user stories
- Average number of atypical words

Regarding the instantiation of $attr(D)$ (which defines the impact on the development phase of the new batch of submitted user stories), we may define three main attributes:

- number of new code lines to write
- number of new classes to implement in the code
- person-hours to allocate

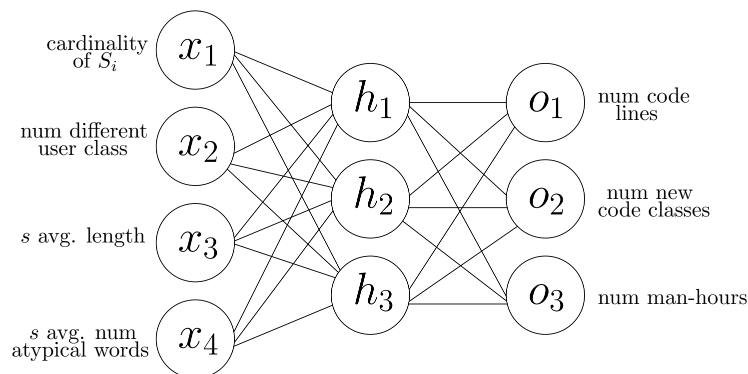

Fig. 3. Artificial neural network for the DCIP plugin

Also in case of the function c an Artificial Neural Network can be employed. Using neural networks for predicting future changes in the software is not new and, if the architecture is properly tuned, this approach can lead to substantially improved results [15], [3]. In Fig. 3, the ANN architecture designed for our problem instantiation is illustrated. It is a common two-layers neural network (also called Multi-Layer Perceptron) where the x_i neurons represent the *input units* and the h_i the *hidden* ones (which are the non-visible computational units, indispensable for learning an high-level representation of the input data). Lastly, the output units o_i, constitute the variables we would like to predict.

Finally, let us consider for the USDP plug-in the strategy defined in eq. 4, where g could be a *classification tree*. After the training we can obtain the acceptance probability as described in [5] and understand why the user story has been classified in a certain way by looking a the structure of the classification tree. Indeed, despite their simplicity, classification trees are still one of the most used algorithms in machine learning due to their efficiency and interpretability. An example of such learnable classification tree can be found in Fig. 4.

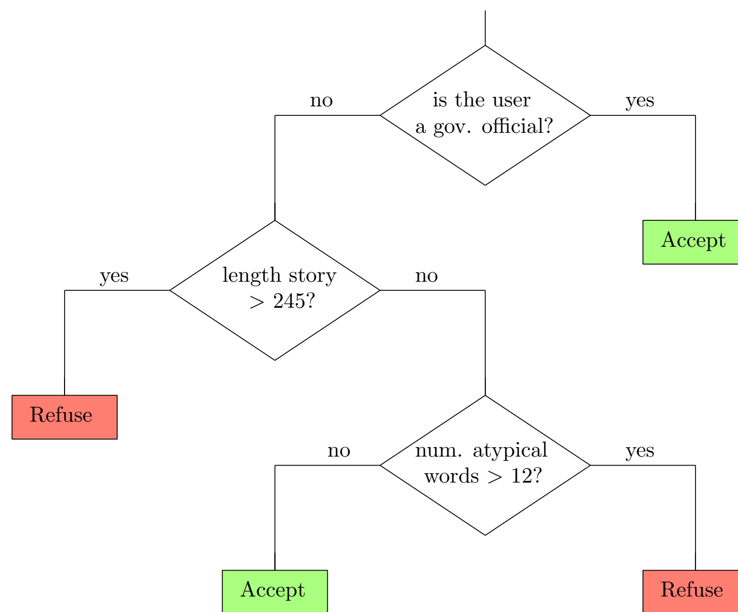

Fig. 4. Learned classification tree for the USDP plugin

5 Conclusions and Future Works

In this paper we proposed a novel machine learning approach for automatically assisting the continuous development. Along with the CDIA Intelligent Assistant formalization

we detailed a possible instantiation of the same in order to show the viability and potential of our approach.

Even though the use of Machine Learning techniques is not novel in this field, we believe this is the first study which proposes a theoretical framework and a systematic approach for the deployment of an automated tool specifically designed for the continuous development context.

We plan to extend this work releasing the extensive experimental evaluation we are currently undergoing in order to show the potential of such a system in a real-world mission-critical application.

Another interesting research direction we are planning to follow in the near future, is to further extend our design infrastructure. The first step would be adding more plugins (like explicit requisites prioritization as in [1], [20]) to the system.

The ideal development of CDIA would then proceed towards a fully comprehensive and refined architecture in charge of the requirements automation and the entire continuous development process: understanding relations and dependences of old functionalities and new ones and help planning their interactions based on past Agile/DevOps iterations or previous developed software which are similar to the one being developed.

6 Acknowledgments

The Authors wish to thank the Consorzio Interuniversitario Nazionale per l'Informatica (CINI) and the Italian National Research Council (ISTC–CNR) for the partial financial support.

References

1. Avesani P., Bazzanella C., Perini A., Susi A.: Facing scalability issues in requirements prioritization with machine learning techniques. *13th IEEE International Conference on Requirements Engineering (RE'05)*, 297–305, 2005.
2. Bachmann F., Bass L., Klein M.: (2003). Preliminary design of ArchE: A software architecture design assistant *CMU/SEI Technical Report*, 21, 2003.
3. Boetticher G.: Using machine learning to predict project effort: Empirical case studies in data-starved domains. *1st International Workshop on Model-Based Requirements Engineering*, 2001.
4. Bosch, J.: Software architecture: The next step. *European Workshop on Software Architecture*, 2004.
5. Buntine W.: Learning classification trees. *Statistics and computing*, 2(2), 63–73, 1992.
6. Ciancarini P., Messina A., Poggi F., Russo D.: Agile Knowledge Engineering for Mission Critical Software Requirements. In *Synergies Between Knowledge Engineering and Software Engineering*. Springer, pp. 151–171, 2018.
7. Ciancarini P., Russo D., Sillitti A., Succi G.: A Guided Tour of the Legal Implications of Software Cloning. *38th International Conference on Software Engineering (ICSE '16)*, 563–572, 2016.
8. Ciancarini P., Russo D., Sillitti A., Succi G.: Reverse Engineering: a Legal Perspective. *31st Annual ACM Symposium on Applied Computing (SAC '16)*, 1498–1503, 2016.

9. Ciancarini P., Poggi F., Russo D.: Big Data Quality: a Roadmap for Open Data. *Proceedings of the 2nd IEEE International Conference on Big Data Service (BigDataService '16)*, 210–215, 2016
10. Cui X., Sun Y., Mei H.: Towards automated solution synthesis and rationale capture in decision-centric architecture design. *7th IEEE/IFIP Working conference on software architecture (WICSA'08)*, 221–230, 2008.
11. Erdogmus H.: Architecture meets agility. *IEEE Software*, 26(5), 2–4, 2009.
12. Fitzgerald B., Stol K-J.: Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123, 176–189, 2017.
13. Gazzerro S., Marsura R., Messina A., Rizzo S.: Capturing User Needs for Agile Software Development. *4th International Conference in Software Engineering for Defence Applications*, 307–319, 2016.
14. Hohpe G., Ozkaya I., Zdun U., Zimmermann O.: The Software Architect Role in the Digital Age. *IEEE Software*, 33(6), 30–39, 2016.
15. Mair C. et al.: An investigation of machine learning based prediction systems. *Journal of Systems and Software*, 53(1), 23–29, 2000.
16. Malavolta I., Muccini H., Pelliccione P., Tamburri D.: Providing architectural languages and tools interoperability through model transformation technologies. *IEEE Transactions on Software Engineering*, 36(1), 119–140, 2010.
17. Martini A., Bosch J.: A multiple case study of continuous architecting in large agile companies: current gaps and the caffea framework. *13th IEEE/IFIP Working conference on software architecture (WICSA'16)*, 1–10, 2016.
18. Messina A., Fiore F., Ruggiero M., Ciancarini P., Russo D.: A new Agile Paradigm for Mission Critical Software Development. *The Journal of Defence Software Engineering (CrossTalk)*, 29(6), 25–30, 2016.
19. Nelson R., Winter S.: *An evolutionary theory of economic change*. Harvard University Press, 1982.
20. Perini A., Susi A., Avesani P.: A machine learning approach to software requirements prioritization. *IEEE Transactions on Software Engineering*, 39(4), 445–461, 2013.
21. Giraud–Carrier, Christophe.: A note on the utility of incremental learning. *Ai Communications*, 13.4, 215-223, 2000.
22. Lomonaco V. and Maltoni D.: Comparing Incremental Learning Strategies for Convolutional Neural Networks. *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, 175–184, 2016.
23. Lomonaco V. and Maltoni D.: CORe50: a New Dataset and Benchmark for Continuous Object Recognition. *arXiv preprint arXiv:1705.03550*, 2017.
24. Russo D., Ciancarini P., Falasconi T., Tomasi M.: Software Quality Concerns in the Italian Bank Sector: the Emergence of a Meta-Quality Dimension. *39th International Conference on Software Engineering (ICSE '17)*, 63–72, 2017.
25. Russo D., Ciancarini P.: Towards Antifragile Software Architectures. *Procedia Computer Science*, 109, pp. 929–934, 2017.
26. D. Russo, P. Ciancarini.: A Proposal for an Antifragile Software Manifesto. *Procedia Computer Science*, 83 (1), 982–987, 2016.
27. Russo D.: Benefits of Open Source Software in Defense Environments. *4th International Conference in Software Engineering for Defence Applications (SEDA '15)*, pp. 123–131, 2016.
28. Yu, E. S.: Towards modelling and reasoning support for early-phase requirements engineering. *3rd IEEE International Symposium on Requirements Engineering (WICSA'16)*, 226–235, IEEE.
29. Zhang D., Tsai J. P.: Machine learning and Software Engineering. *Software Quality Journal*, 11(2), 87-119, 2003.