Analysis of the Effectiveness and Efficiency of Software Development Tools

Transmissia Semiawan[1] Muhammad Riza Alifi[1,*] Hashri Hayati[1] Djoko Cahyo Utomo Lieharyani[1]

[1] *Politeknik Negeri Bandung*
[*] *Corresponding author. Email: muhammad.riza@polban.ac.id*

ABSTRACT
Software tools, frameworks, and libraries are relatively crucial in software development tools. However, these tools cannot guarantee that a developer will produce good quality software applications. This study aims to investigate the efficiency and effectiveness of software tools, specifically frameworks, that support developers in producing good quality applications. Using an experimental approach in software development, the developer participants were given a case in which software needed to be developed with and without a framework. The experiment results were then evaluated to obtain facts from the participants' experiences during development, through some data sources. The results of the study show that the use of a framework does not significantly help developers effectively and efficiently develop software or applications. The success of the application that has been measured based on the level of completion, quality, and usability of the application does not show significant differences in value whether developed with or without tools. Furthermore, the study revealed that the software development skill that cannot be obtained through the use of framework was the general or detailed system conceptualization.

***Keywords**: software development tools, efficiency & effectiveness of software tools, developer skills.*

1. INTRODUCTION

Learning in vocational education requires a profound link between theoretical and practical learning. One of the important aspects of practical learning is hands-on learning experiences. This type of learning activity requires adequate and large amounts of supporting equipment.

One of the core study units in the Department of Computer Engineering and Informatics is software engineering, which is a software development process that requires core activities such as analyzing, system designing, coding, and unit testing, integrating, and system testing [1]. Practicing software development can be supported by development tools such as frameworks, tools, and libraries [2] to support every stage of software development [3, 4, 5, 6]. Several studies have proven that using tools can provide a variety of conveniences, including insight into the success and quality of the software that has been developed [1]. This practicality is expected to improve developers' skills in optimum software development.

1.1. Related Studies

Nowadays, a framework is a commonly used tool because of the increasing demands to accelerate software production. Framework's paradigm in building the architecture of reusable software is by partially developing the software [3]. This allows the framework to support the development of highly complex and scalable software that has a more structured sustainable development.

In addition, the current framework model has the capability to provide an API (Application Programming Interface) to accommodate cross-platform interoperabilities [3], to adapt to the software application domain with a framework construction that is based on its specific domain [4], and to support multi-language programming [5].

The usability of the framework and its appropriate application in developing software will provide advantages in terms of effectiveness and efficiency, such as:

1. Providing comfort in predicting the time and progress of the software development process [1];
2. Producing quality software that has the capability to minimize defects by detecting and correcting defects early [1][4];
3. Increasing developer productivity by diverting from activities that do not have added value, that usually tends to be repetitive (rework) in building general components [4][7];
4. Improving the usability of the produced software in terms of meeting its requirements, in accordance with the pre-defined software design specifications [4].

1.2. Our Contribution

Developers should not depend entirely on tools, because there are other factors that developers need to effectively and efficiently completely develop the software.

This study examined the effectiveness and efficiency of software tools, specifically the use of a framework for software development. This study examined how effectively framework provides support to developers and identified which factors cannot be dealt with by framework in producing effective and efficient software or application. The results of this study provide awareness to the Department of Computer Engineering and Informatics that there are skills that may be required for practical learnings of software development but cannot be replaced by a framework.

2. BACKGROUND

In a software development process, tools are significant for developers in successfully creating software that meets the specific requirements of the software being developed [3]. These tools help developers build the software more efficiently [3], such as reducing code lines and significantly saving time, from problem identification until the software is developed, to ensure that the software can function according to its requirements [2].

The following are types of software development tools that are often used by developers:

1. *Tools*: a technical aid in software programming that is commonly used by developers for editing, debugging programs, etc. There is a variety of software development tools, such as code editors, compilers, linkers, debuggers, GUI designers, assemblers, and performance analysis tools [2].
2. Framework*:* defined as a generic object or function [2] which is developed according to the problem domain [4], a framework is used to improve the productivity of a development process because the objects that are being developed can be used repeatedly, therefore able to reduce development time and improve the quality of the software application [4].
3. *Libraries*: a collection of objects or functions or support modules (helpers), which are required by the application for specific functionalities. Libraries have a much-limited scope because they are specific, such as providing IO, sockets, and strings [2].

Based on studies [1, 4, 7], this study measured the effectiveness and efficiency of a framework based on the following factors: a) completion time; b) thorough production of the software; c) quality of the software in terms of defects; d) rework of component constructions; e) usability in terms of accommodating the requirements. The focus of this study was on the development of front-end and back-end applications.

3. RESEARCH METHODS

This study used an experimental method to compare the results of application developments with and without a framework. The application is focused on the front-end and back-end types of application. In line with its objectives, this study did not highlight whether or not the application is correct, rather it identified the extent to which framework affects the application development process.

3.1. Development Stages

The experiment is divided into two stages of software development: (1) application/module development without a framework, in which the developer participants work on the project by understanding and solving problems using their thoughts; and (2) application/module development using a framework.

The following four phases of the software development lifecycle (SDLC) was applied for each of the development stages: (a) problem identification; (b) analysis based on user and system requirements; (c) design that included data design, functionality, and the user interface; and (d) implementation, including environmental settings and requirements.

To observe the effectiveness and efficiency of the tools on each of the above stages, the following four critical success factors were evaluated and analyzed:

a) *elapsed time* (T): the time required to produce the application;
b) *completeness of the application* (P): application completion stage according to their requirements;
c) *quality of the application* (Q): software quality related to defects;
d) *repetitiveness* (R): *reconstruction of* components;
e) *usability* (U): output appropriation stage in accordance with the users' needs of the application.

Efficiency is measured based on elapsed time (T) and repetitiveness (R) factors, while effectiveness is measured based on application completeness (P), quality (Q), usability (U), and repetitiveness (R) factors.

3.2. Effectiveness & Efficiency Measurement Factors

a) Elapsed time (*T*): defined as the time to produce an application. In this study, the elapsed time was measured for each experimental stage to represent the relative time required to complete the application. Formula (01) is the formula to calculate elapsed time.

$$T = \frac{\Delta t}{P} \quad (01)$$

T = elapsed time
Δt = completion time (in minutes)
P = completeness of the apps (%)

T shows the amount of time (in minutes) that the developer takes to complete 1% of all of the given requirements. The lower the T value, the more efficient the time would have been used to complete the application.

b) Completeness of the app (*P*): the level of application completion according to its requirements. P was measured by looking into the completion of the required component using Yes/No answer options. This study used 27 requirements, to which each defined requirement was assigned a value of 1 if it is completed (Yes) and 0 if it is not completed (No). Formula (02) shows the formula that was used to illustrate the app completeness.

$$P = \frac{\sum_{i=1}^{n} r_i}{n} \times 100\% \quad (02)$$

P = completeness of the apps (%)
r_i = value of the i-*th* requirement
n = number of requirements

P shows the percentage of requirements that are completed by the developer. The higher the P-value, the more effective the completion of the application.

c) Quality of the app (*Q*): the software quality related to defects. (*Q*) is measured by testing the application of business rules that have been defined and contain provisions related to technical and data validation and application functionality. Business rules must be applied for the developed software to functionally operate and to fulfill the owner's quality expectations. The application of business rules can also reduce the number of defects in the software application by using different operational scenarios that have been prepared to handle the flaws of the application. This study examined 16 (sixteen) business rules, each of which was defined and assigned a value of 1 if successfully applied to the software (Yes) and 0 if not successfully applied (No). Formula (03) shows the formula that had been used to define the application's quality.

$$Q = \frac{\sum_{i=1}^{n} b_i}{n} \times 100\% \quad (03)$$

Q = quality of the apps (%)
b_i = value of the i-*th* business rule
n = number of business rules

Q is shown in percentage to indicate business rules that can be successfully applied by a developer. The higher the Q value, the more effective the application development by the developer.

d) Repetitiveness (*R*): repetition intensity of the creation of the same component during application development. The (*R*) value is obtained by directly examining the source code generated by the developer based on 11 (eleven) measurement indicators. Formula (04) shows the process of calculating the efficiency value of (*R*) for each developer. Each defined indicator (*k*) is assigned a value of 1 (Yes) if a repeated component is found and a value of 0 (No) if it is not found.

$$R = \frac{\sum_{i=1}^{n} k_i}{n} \times 100\% \quad (04)$$

R = repetitiveness (%)
k_i = value of the i-*th* indicator
n = number of indicators

The lower the (*R*) value, the more efficient the software development process because it shows less repetition of the same component created by the developer.

Besides being able to produce efficiency values, *R* can also show the effectiveness of using the framework by comparing the efficiency values between stage 1 and stage 2. The effectiveness value is obtained by reducing the efficiency value in stage 1 by stage 2. The higher the value obtained, the more effective the role of a framework in minimizing the creation of the same component.

e) Usability (*U*): appropriateness of output against users' needs. The (*U*) value is measured by testing the acquired software usability specifications which are related to user needs. This study examined 18 usability specifications. Each defined usability specification was assigned a value of 1 if successfully applied to the software (Yes) and 0 if not successfully applied (No). Formula (05) shows the formula that had been used to express usability.

$$U = \frac{\sum_{i=1}^{n} u_i}{n} \times 100\% \quad (05)$$

U = usability (%)
u_i = value of the i-*th* usability specifications
n = number of usability specifications

(*U*) indicates the percentage of usability specifications that the developer successfully implements. The higher the (*U*) value, the more effective the application that had been done by the developer.

Formula 02, 03, and 05 are developed to formulate the effectiveness which shows the completion of the product based on the fulfillment of requirements, quality, and usability. Whereas Formula 01 and 04 are developed to formulate the efficiency should aim to minimize the use of resources based on time and repetitiveness in lines of code.

All of the factors above were calculated and analyzed using Harmonic Mean [8] (Formula 06), and the Pearson Correlation [9].

$$H = \frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\frac{1}{x_3}+\ldots+\frac{1}{x_n}} \quad (06)$$

Harmonic Mean is used to determine the average value of all participants on each factor in successful application development. Pearson's product-moment correlation method was used to see the correlation between the investigated parameters or factors. In this study, the results of the correlation analysis became the basis in interpreting which factors influence one another and further analysis on other facts that may be found during the software development process. Data sources for evaluation and analysis of the application were obtained through interviews, logbooks, and repositories, and a combination of these sources.

4. RESULT AND DISCUSSION

The case used for this study was an application development project on Final Project Management (TA) in the Department of Computer Engineering and Informatics, Politeknik Negeri Bandung. There are two types of users of this application: TA Coordinator and Student (TA Group Leader). This project incorporated 11 Business Rules, 4 Assumptions, 7 Requirements, and 16 Sub-requirements. This project asked the developer to create an application in a Progressive Web Application (PWA) that consisted of a front-end, back-end and connected to the Application Programming Interface (API).

4.1. Measurement Results of Effectiveness & Efficiency Factors

Table 1 shows the average success rate of application developed based on 5 (five) observed factors: elapsed time (*T*), completeness (*P*), quality of apps (*Q*), repetitiveness (*R*), and product usability (*U*) in the two stages.

Table 1. Application Success Rate

Factor	Stage 1	Stage 2
Elapsed time (*T*)	29.86 min/1%	14.11 min/1%
Completeness (*P*)	64.13%	77.2%
Quality of apps (*Q*)	64.46%	69.75%
Repetitiveness (*R*)	74.04%	12.12%
Usability (*U*)	59.13%	65.73%

Elapsed time (*T*): Based on Table 1, the completion of the 1% requirement in stage 1 took approximately twice as much in stage 2. However, because this study used the same case, the minimum time required for the second stage of the experiment was also influenced by the starting point. In stage 1, the developer needed time to understand the case requirements, while in stage 2 the developer was able to work more quickly because they have already had an overview of the application from the previous stage.

Although the (*T*) value differed significantly, the difference of completeness value (*P*) was not significant. This shows that although the framework work time was shorter, the participants' ability to manage time and work remained not optimum. The same occurred with the (*Q*) value which was not significantly different in the two stages. This shows that although the use of framework was able to speed up tasks, it did not necessarily improve product quality. In stage 2, the second iteration in which the resulting quality did not increase significantly, results showed that quality improvement does not depend on the use of the framework, rather on the developer's ability to evaluate the system needed to produce quality applications.

Completeness (*P*): Table 1 shows a 13.07% P factor escalation from stage 1 to stage 2. With the framework and considering stage 2 condition, which was the second time that participants developed the same application, this escalation is considered insignificant. Several requirements were not met in this experiment, among others due to the system processing time and understanding of requirements. Understanding the requirements requires the ability to analyze, solve problems, and comprehend the system. Both in working with or without the framework, developers need to understand the system or the application domain on which they are working and are capable to elaborate the problem. With this capability, developers will be able to understand the process as a whole, identify needs and technology, and systematically complete the work.

Quality of apps (*Q*): Table 1 shows a (*Q*) factor increase of 5.29% from stage 1 to stage 2. This increase is considered insignificant, based on the condition that stage 2 of the work was supported by framework and participants have already had experience from the previous work. To obtain a good (*Q*) factor, developers

need to have critical thinking skills, systems analysis, and a thorough understanding of its processes and functional requirements. Equipped with these capacities, developers can capture and consider quality from various standpoints or scenarios to create applications that will not only meet the list of functional requirements but also have good quality standards. In this case, quality is measured by the accomplishment of business rules that have to be analyzed for their applicability to various possible scenarios. Whether or not tools were used, the results show that the numbers of applied business rules did not significantly differ. The condition in which developers merely follow the rules rather than understand how they should have been applied according to requirements was a causal factor of these results.

Repetitiveness (*R*): Referring to Table 1, the efficiency value of (*R*) factor in stages 1 (74.04%) and 2 (12.12%) differ at 61.92 percent. This significant difference indicates that the framework is able to effectively minimize the repetition of source codes that have the same functionality. During the first stage, developers tend to repeat parts of the source code that have the same function rather than building components for reuse. Meanwhile, in stage 2, developers less frequently repeated the source codes that had the same functionality because the framework had already provided the common components for software development. In stage 2, the majority of developers already had a good understanding of how to use the framework. By using framework components correctly, the resulting source code was more efficient and easier to maintain.

Usability (*U*): In this study, usability is referred to as the way the system meets users' requirements in terms of easy operationalization, learning, and understanding, based on the output viewed by the user. Table 1 shows a 6.6% (**U**) factor increase from stage 1 to stage 2, which is insignificant, given that stage 2 was supported by framework and participants have had prior work experiences. Similar to the (*P*) and (*Q*) factors, the (*U*) factor can help developers complete all of the requirements quickly and easily trace the references. However, the usability value is mainly determined by the developers' ability to identify users' needs and processes that will be experienced within the application. Therefore, a developer should have analytical and problem-solving skills to interpret user needs to be applied to and appropriately use the application. This study did not show a significant increase in usability value because the developer used the same logical design, process design, and needs identification for both stages based on the result of stage 1. The designs in stage 2 did not significantly change, even though the technological aspect of its implementation had been different.

4.2. Correlation Analysis

Correlation analysis was performed using Pearson Correlation. The coefficient number in Table 2 and Table 3 shows how strong the correlation between variables is. The closer the number to 0, the weaker the correlation. While the sign indicates the nature of the correlation, whether positively correlated (+) or negatively correlated (-).

Table 2 shows correlation analysis results for the five factors in stage 1 explained above.

Table 2. Stage 1 Correlation Analysis

	T	P	Q		R	U
T	--					
P	**-0.91**	--				
Q	**-0.83**	**0.83**	--			
R	0.54	-0.71	-0.30		--	
U	**-0.92**	**0.94**	**0.91**		-0.44	--

The analysis showed a strong negative correlation between (*T*) and (*P*) factors (-0.91 correlation coefficient), (*Q*) (-0.83 correlation coefficient), and (*U*) (-0.92 correlation coefficient). This demonstrates that the more efficient the working time (lower elapsed time or faster processing time per 1% requirement), the higher the effectiveness or number of completed products (*P*), better application quality (*Q*), and increased usability (*U*).

Besides T, usability also has a strong positive correlation with (*P*) and (*Q*). A high correlation between (*P*) and (*U*) illustrates that if a completed product has good usability, then it has sufficiently or completely fulfilled the requirements. Table 1 shows a usability level of less than 60%, indicated by a product achievement level of only 64%. The high correlation between (*Q*) and (*U*) factors can also be articulated as having a good quality product, which was focused on preventing or minimizing defects in this study, and good usability. This correlation is illustrated in Table 1, which shows that the quality of the application is more or less proportional to its usability.

Table 3 shows correlation analysis for the above-mentioned five factors in stage 2.

Table 3. Stage 2 Correlation Analysis

	T	P	Q	R	U
T	--				
P	-0.38	--			
Q	-0.12	0.31	--		
R	-0.59	-0.48	-0.10	--	
U	**-0.74**	**0.75**	0.65	0.09	--

Based on Pearson Correlation, results of the correlation analysis showed a fairly high negative correlation between (*T*) and (*U*) factors (-0.74 correlation coefficient), and a fairly high positive correlation between (*U*) and (*P*) (0.75 correlation coefficient). As in stage 1, this shows that the more efficient the work time (lower elapsed time or faster completion time per 1% requirement), the more usability (*U*) specifications had been completed. In addition, the high correlation between (*P*) and (*U*) can indicate that when the product has good usability, then the product sufficiently or completely meets the requirements in general.

There are differences in the correlation analysis results in Stage 1 and Stage 2, partly due to the situation in stage 2 in which the developer had a better understanding of the application's problem domain, thus the relatively shortened total work time but was able to achieve the same results. Furthermore, repetitiveness tends to hide correlation with other aspects because it is detached from the result and work completion. In Stage 1, without a framework, repetitiveness almost always occurred (74.04% in Table 1), while in Stage 2 it rarely occurred due to several repetitions that had been already tackled by the framework (12.12% in Table 2).

4.3. Relation Analysis between Programming Tools and Skills

These skills are decided by identifying those that are related to digitalization and developers' needs (developer skills). 95 developer skills have been identified thus far [10, 11]. The next step was selecting the developer skills relevant to the project as had been prepared by this study, which thus identified 55 relevant developer skills. The developer skills identification process continued by analyzing the skills that were unobtainable using a framework and adjusting the skills with the ongoing research. This process identified 26 skills that were grouped into 5 skill categories related to SDLC: (1) problem solving, (2) analytical, (3) functional, (4) decision-making, and (5) full-stack development, which is a general summary of the first four SDLC-related skill categories.

Based on interviews, product presentations, product compliance analysis, five-factor measurement, as well as references to the correlation analysis results and effectiveness and efficiency measurement factors mentioned above, it was found that there is a relation between effectiveness and efficiency with the identified skill categories. The analysis was able to identify that effectiveness measured based on (*P*), (*Q*), and (*U*) factors affected developer skills in problem-solving, analysis, and functional skills. Meanwhile, efficiency that is measured based on (*T*) and (*R*) factors affected developer skills in decision making. As mentioned above, problem-solving, analytical, functional, and decision-making skills are interrelated and affect full-stack development which is focused on SDLC and its application. A model of the effectiveness-efficiency relation with developer skills is illustrated in Figure 1.

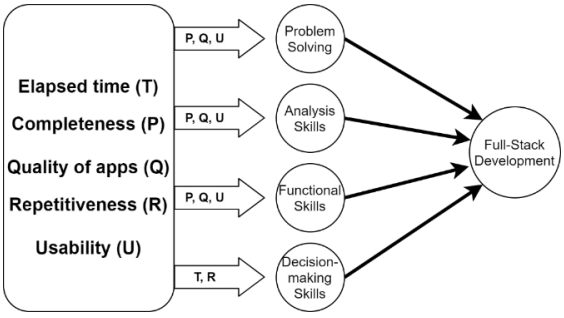

Figure 1 Relationship Model between Effectiveness and Efficiency Factors against Developer Skills

Table 1 is also related to the capability of developers in application development (SDLC), especially in solving problems (problem-solving), analyzing (analytical skills), understanding system functionality in a design (functional skills), and making decisions (decision-making skills).

Problem-solving: The less than optimum (*P*), (*Q*), and (*U*) percentages show that the application also did not provide the optimum solution to the problem. This indicates that the developer could not identify the problem domain. By not being able to comprehensively identify the problem domain, it indicates that the developer does not have enough capability to capture the linkage between the problems and the given requirements.

Analytical skills: problem-solving skill is inseparable from analytical skill. To be able to identify problem domains, developers need to have the capability to analyze all the data/information that has been provided to them and observe the relationships among the data in a conceptual abstract picture of the requirements of the existing system or the system that will be developed (which includes a description of the input, process, and output of the system). By analyzing the requirements, the developer is expected to be able to identify user needs and system requirements. Less than optimum (*P*), (*Q*), and (*U*) percentage values explain that the developer's analytical skills that are related to understanding system requirements need to be improved.

Functional skills: when requirements cannot be thoroughly understood, it means that developers were not able to thoroughly see the relationships between requirements as complete system functionality. As a result, the overall application design requirements could not be accommodated. Evidence of this condition is shown by the less than maximum percentage completeness that indicated that the requirements could not be fulfilled. The system functionality is thus incomplete, which affected usability and quality imperfection.

Decision-making skills: time and module/library management are highly affected by the developer's capability in making decisions, and the condition or situation becomes the decided method to implement and complete the task. In terms of more efficiently managing work time, a developer needs to have the capacity to indicate aspects that can affect completion time. This includes deciding whether or not a new module/library should be made or to use an existing one. Based on the (*R*) results, the majority of developers in stage 2 tend to take advantage of components that were already available from the framework. Some developers did not take full advantage of framework components in stage 2 because they were able to reuse the components that they had created in stage 1. The variety of these decisions by developers in stage 2 is consistent with the increasingly significant (*T*) in stage 2. It is thus essential that developers are capable to make decisions based on conditional and situational considerations in developing and using components.

Development skills: these comprehensive skills in software development and, in terms of SDLC, encompass the balance between **conceptual** understanding and **technical** understanding that a developer must have. To be able to achieve conceptual understanding, it is necessary for a developer to have the capability to analyze the problem domain, as well as the capability to determine the right solution for the problem through an abstraction or solution design model that can be seen from the data design model, the process and the manifestation of the solution. Technical understanding requires the capability that is relevant to accommodate and implement the system design model. The capability does not only apply to programming but also in understanding the surrounding environment and support system in which the application will operate. If the two understandings do not balance, or if the developer only has the conceptual or technical capability, a comprehensive and high-quality application design will not be developed.

5. CONCLUSION

The results of this study show a developer's technical capability is stronger than their conceptual capability. This is shown by the effectiveness and efficiency measurement of the framework in the application development by the participants. However, this result does not significantly show better results without tools. This was due to insufficient problem-solving capability, analytical skills, and functional design skills. This indicates that conceptual understanding capability becomes significant to consider in software development learning.

This research provides insight into the framework as a tool that is limited to only provide support for speed and easiness in application development, due to having tools that are ready to use. Therefore, the framework does not directly support the development of required, efficient, and high-quality applications. The analysis showed that the success of the application development strongly depends on the five developer's skills: problem-solving, analytical skills, functional skills, decision-making skills, and development skills.

ACKNOWLEDGMENTS

This study was supported by a competitive grant for the capacity development of program studies of Politeknik Negeri Bandung, grant number B/75.5/PL1.R7/PG.00.03/2021.

REFERENCES

[1] Harry L. Levinson & Richard M. Librizzi. Using Software Development Tools and Practices in Acquisition. Software Engineering Institute, Carnegie Mellon University; 2013.

[2] Ruchika Aggarwal. Top Software Development Tools, Frameworks, and Libraries in 2021 [Internet]. India, Gurugram: ValueCoders; 2020 [updated 2020 October 12; cited 2021 May 25]. Available from: https://www.valuecoders.com/blog/technology-and-apps/top-software-development-frameworks-tools-and-libraries/

[3] Varvana Myllärniemi, Sari Kujala, Mikko Raatikainen, and Piia Sevón. Development as a journey: factors supporting the adoption and use of software frameworks. Journal of Software Engineering Research and Development; 2018, 6:6. DOI: https://doi.org/10.1186/s40411-018-0050-8

[4] Matheus C Viana, Rosângela AD Penteado, Antônio F do Prado, and Rafael S Durelli. 2015. F3T: a tool to support the F3 approach on the development and reuse of frameworks. Journal of Software Engineering Research and Development; 2015, 3:4. DOI: https://doi.org/10.1186/s40411-015-0017-y

[5] Philip Mayer, Michael Kirsch, and Minh Anh Le. On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers. Journal of Software Engineering Research and Development; 2017, 5:1. DOI: https://doi.org/10.1186/s40411-017-0035-z

[6] George A. Sielis, Aimilia Tzanavari, and George A. Papadopoulos. 2017. ArchReco: a software tool to assist software design based on context aware recommendations of design patterns. Journal of Software Engineering Research and Development;

2017, 5:2. DOI: http://doi.org/10.1186/s40411-017-0036-y

[7] Cheng Zhang and David Budgen. What Do We Know about the Effectiveness of Software Design Patterns? IEEE Transactions on Software Engineering, Vol. 38, No. 5; 2012, 9. DOI: https://doi.org/10.1109/TSE.2011.79

[8] Jasmin Komić. Harmonic Mean. Berlin, Heidelberg: Springer, International Encyclopedia of Statistical Science; 2011. DOI: https://doi.org/10.1007/978-3-642-04898-2_645

[9] Ygnve Lindsjørn, Dag I. K. Sjøberg, Torgeir Dingsøyr, Gunnar R. Bergersen, and Tore Dybå. Teamwork quality and project success in software development: A survey of agile development teams. Journal of Systems & Software, 122, 274-286; 2016. DOI: https://doi.org/10.1016/j.jss.2016.09.028

[10] Anna Wiedemann and Manuel Wiesche. Are You Ready for DevOps? Required Skill Set for DevOps Teams. Twenty-Sixth European Conference on Information Systems (EICS2018), 123, 1-16; 2018. https://aisel.aisnet.org/ecis2018_rp/123

[11] Hasan Bakhshi, Jonathan M. Downing, Michael A. Osborne and Philippe Schneider. 2017. The Future of Skills: Employment in 2030. PEARSON. ISBN: 978-0-992-42595-1