Scientific Research Publishing

Journal of Software Engineering and Applications, 2021, 14, 442-453
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Towards a Framework for Evaluating Technologies for Implementing Microservices Architectures

Aristide Massaga*, **Georges Edouard Kouamou**

Department of Computer Engineering, National Advanced School of Engineering, Yaoundé, Cameroon
Email: *aristidemassaga@gmail.com

How to cite this paper: Massaga, A. and Kouamou, G.E. (2021) Towards a Framework for Evaluating Technologies for Implementing Microservices Architectures. *Journal of Software Engineering and Applications*, **14**, 442-453.
https://doi.org/10.4236/jsea.2021.148026

Received: July 13, 2021
Accepted: August 27, 2021
Published: August 30, 2021

Abstract

Microservice architecture is an architectural style, which allows structuring software as a suite of fine-grained services, each running in its process and deployed independently. Knowing the strengths and limitations of this architectural style, the development team is responsible to select the appropriate technologies which guarantee the consistency between the implementation and the design. This study proposes an evaluation framework which consists of a set of evaluation criteria that are architectural patterns recognized by the community and covering all the implementation aspects of software; and an evaluation function which combines these criteria for a given technology to determine its compatibility score with the microservice style, while taking into account the specific requirements of the software under development. Applying this approach to Spring Boot and JAVA EE technologies, we found that Spring Boot scores 96.3% while JAVA EE scores 44.4%. These scores reflect the effort required to conform software with the principles of this development style.

Keywords

Microservice Architecture, Evaluation Criteria, Architectural Patterns, Spring Boot, JAVA EE

1. Introduction

Like all architectural styles, the microservice style is a solution to a software structuring problem that the software industry has faced. A **microservice** is a lightweight, independent service that performs unique functions and collaborates with other similar services using a well-defined interface [1]. **Microservice architecture** is an architectural style that structures software as a collection of services: Highly maintainable and testable; loosely coupled; independently deployable; organized around enterprise capabilities; developed by a small team [2].

With the emergence of cloud computing and the increasing use of agility in software development processes, the microservice architecture offers many advantages, such that it becomes one of the most suitable styles for these new industry needs, as it offers developers: 1) Ease of integration and automatic deployment; 2) Freedom to develop and deploy independently; 3) Ease of understanding and modification for developers, allowing a new team member to be productive quickly. However, the decomposition of a monolithic software, into microservices also causes problems: 1) Due to distributed deployment, testing can become complicated and tedious; 2) Increasing the number of services can lead to information barriers; 3) Splitting the software into microservices is a highly complex operation.

Knowing the strengths and limitations of this architectural style, the responsibility is given to the development team to make the right technological choices so that the implementation is as consistent as possible with the design. This requires being able to verify that a technology retains the strengths of the style, that it provides optimal solutions to the problems underlying the style, and that it respects the development standards of the style. The reflection that we carry out in this work is part of this same problem, which is to know how to evaluate the contribution of technology for the implementation of a microservice oriented architecture.

In the literature, research is mainly oriented towards the verification of the architectural conformity of software [3] [4] [5]. Several approaches have been proposed [6] based on the recognition of code structuring (packages), design patterns or architectural patterns present in software.

Weinreich *et al.* [7] address the problem of verifying the conformity of software with the SOA architectural style. Their three-step approach is based on the identification of architectural patterns in software.

The main contributions of this paper are: 1) A catalog of microservice implementation architectural patterns; 2) A correlation between microservice (distributed systems) issues and architectural patterns; 3) An evaluation function taking as parameters a technology and the requirements of the developed software to assign a compatibility score.

In the reminder of this paper, **Section 2** describes the methodology applied in this study, which consists of the identification of the criteria and the construction of the evaluation function. **Section 3** presents an illustration based on the evaluation of two technologies Spring Boot and JAVA EE from the constructed framework. **Section 4** concludes the paper, then discusses the future works.

2. Methodology

Evaluating the compatibility of an architectural style with a certain technology (programming language or framework), consists in verifying that this technology preserves the assets of the style, that it brings solutions to the underlying problems but especially that it respects the standards on which the style is built. The evaluation framework that we propose is articulated in 2 parts:

1) **Choice of Evaluation Criteria:** A list of architectural criteria that technology must verify. This checklist is made up of architectural patterns universally recognized and accepted by the microservice community. They are solutions to implement the style at the service level and in their relationships. In fact, if they are respected, they cover all the aspects of the implementation of microservice software, lead to the conservation of the assets of the style, and are solutions to the problems presented.

2) **Evaluation Function:** This is a parametric function that: a) For a candidate technology t; b) A set of architectural patterns P from the identified architectural patterns deemed necessary for the software under development; c) And a vector of weighting coefficients E to express the levels of importance varying from one architectural pattern to another. This function returns a score, expressing the degree of compatibility of the studied technology according to the past parameters.

2.1. Choice of Evaluation Criteria

In terms of implementation, the architectural requirements of software vary greatly from one software to another. Therefore, the evaluation criteria to be established must cover as many implementation cases as possible. To achieve this goal, we proceeded in two steps:

1) Divide the implementation of software into design domains following the domain-driven design (DDD) methodology. At the end of this step, 11 main domains were identified, covering the main crosscutting concerns in the implementation of the microservice style.

2) Research the architectural patterns that serve as best practices for the implementation of each design domain. At the end of this step 27 main architectural patterns have been identified.

From this process we obtain the following Table 1 consisting of 3 columns:

- The identified domains;
- The problem covered by the domain: this is stated in the form of a question;
- Architectural patterns/evaluation criteria: these are the patterns that fall within this domain.

2.2. Evaluation Function

Let P be the set of evaluation criteria to study the compatibility of a technology. The elements of this set are taken from the 27 architectural patterns determined in Section 2.1, so we have: $P = \{p_1, p_2, \cdots, p_n\}$ with $n \le 27$.

Table 1. Table of evaluation criteria.

Domains	Issues covered	Architectural patterns/evaluation criteria
Data management	Which architecture to adopt for data management (reading, writing)?	Database per service API Composition SAGA Domain Event Event sourcing
Test management (Testing)	How to test processes involving several microservices?	Testing of service components Service Integration Contract Testing
Deployment	How to deploy services written in different languages while ensuring devops requirements?	Multiple service instances per host Service instance per container Serverless deployment
Cross-cutting concerns	How to allow a service to run in multiple environments without modification?	Externalized configuration
Communication style	How to make services communicate?	Remote Procedure Invocation (RPI) Message exchange (Messaging)
External API	How do clients access individual services?	API Gateway Backends for frontends
Service discovery	How does the client of a service, the API gateway or another service, discover the location of a service instance?	Service Registry Client-side service discovery Server-side service discovery Self-registration
Reliability	How do you prevent a network or service failure from affecting other services?	Circuit breaker
Security	How do you communicate the identity of the applicant to the departments processing the software?	Access token
Observability	How to understand the behavior of a software and solve problems?	Log aggregation Implementation measures Distributed tracing API Health Check
User interface templates	How do you implement a screen or UI page that displays data from multiple services?	Composition of the page fragment on the server side Composition of the client-side user interface

Let $E = \{e_1, e_2, \cdots, e_n\}$ with $e_i \in \{1,2,3,4,5\}$ where e_i represents the weighting coefficient associated with the architectural pattern p_i. It is used to express the level of importance of the pattern p_i with respect to the other patterns, for the software that is under development. Thus, the least important patterns will have the value 1 and the most important value 5.

Let h be the function whose role is to indicate if an architectural pattern is implemented or not in technology. It receives as input two parameters: the technology t and a pattern p_i. If this pattern is implemented in the given technology, then $h(t, p_i) = 1$, otherwise $h(t, p_i) = 0$.

The evaluation function is thus of the form $f(t, P, E)$, where t is a candidate technology for implementation of a microservice software.

The output of this function is a score, which indicates the level of compatibility of the technology with the microservice style according to the parameters received as input. Figure 1 shows a graphical representation of this function.

Our evaluation function is therefore as follows:

$$f(t,P,E) = e_1 \times h(t,p_1) + \cdots + e_n \times h(t,p_n) = \sum_{i=1}^{n} e_i \times h(t,p_i) \tag{1}$$

Knowing that:

$$0 \le h(t,p_i) \le 1 \quad \text{and} \quad 1 \le e_i \le 5$$

$$\Rightarrow 0 \le e_i \times h(t,p_i) \le 5$$

$$\Rightarrow \sum_{i=1}^{n} 0 \le \sum_{i=1}^{n} e_i \times h(t,p_i) \le \sum_{i=1}^{n} 5$$

$$\Rightarrow 0 \le \sum_{i=1}^{n} e_i \times h(t,p_i) \le 5n \quad \text{since} \quad n \le 27.$$

Therefore, the frame of the function f is:

$$f(t,P,E) \in [0, Max(E) \times n] \subset \mathbb{N} \tag{2}$$

From this, we see that the degree of accounting of technology according to the evaluation function varies between 0 and 135.

3. Illustration

In this section, we illustrate the evaluation of two technologies for the implementation of the microservice style: Spring Boot 2.2.2 and JAVA EE 7. The reasons for this choice are: 1) They are technologies based on the same language; 2) The widely used language [8]; 3) They are backend technologies; 4) The Spring

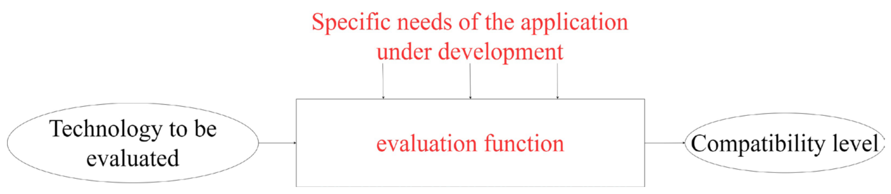
Specific needs of the application under development
Technology to be evaluated
evaluation function
Compatibility level

Figure 1. Evaluation function.

Boot framework and the JAVA EE platform are strongly used by the community [8] [9].

Before the evaluation begins, it is necessary to make some assumptions:

- **Assumption 1:** Since we are doing a broad study, the set *P* will correspond to the 27 patterns identified in Section 2.1.
- **Assumption 2:** All patterns $p_i \in P$ have the same importance level equal to 1, $\forall e_i \in E, e_i = 1$.
- **Assumption 3:** The value of the function *h*, is obtained by checking whether in the universe of official packages of the studied technology there is a package that implements the criterion passed as a parameter.

From hypotheses 1 and 2, it appears that the compatibility score resulting from the evaluation function will vary between 0 and 27.

Any evaluation will be done in two steps:

- Search for the value of the function h, for each criterion;
- Calculation of the value of the evaluation function $f(t,P,E)$.

3.1. Evaluation of Spring Boot 2.2.2

Spring Boot is a project or a micro framework that aims to facilitate the configuration of a Spring project and to reduce the time allocated to the start-up of a project. To achieve this goal, Spring Boot is based on several elements [10]:

- A web site (https://start.spring.io/) that allows to quickly generate the project structure;
- The use of "Starters" to manage the dependencies;
- Auto-configuration, which applies a default configuration at the start of the software for all dependencies present in it.

Spring cloud [11] is a project based on Spring Boot, designed to address the specific issue of microservices. It provides developers with tools to quickly create some common patterns in distributed systems.

From **Table 2** obtained by analysis of the Spring Boot technology, the value obtained for the function *f* is:

$$f(t,P,E) = 26.$$

Thus, Spring Boot is **96.3%** compatible with the microservice architecture.

3.2. Evaluation of JAVA EE 7

JEE (Java Enterprise Edition) is a specification for Oracle's Java platform for enterprise software. The platform extends Java Platform, Standard Edition (Java SE) by providing an object-relational mapping API, distributed and multi-tier architectures, and web services. The platform is primarily based on modular components running on a software server as in **Figure 2**.

The JAVA EE platform proposes an organization of the code, according to the MVC model (**Figure 3**). In the JAVA EE universe, each element has a specific designation:

- The Controller is called Servlet;

Table 2. Table of values of the function h, for the Spring Boot 2.2.2 technology.

	Evaluation criteria	h	Justification
p1	Database per service	1	Since Spring offers packages for connecting to and manipulating most existing DBMS, it is fully compatible with this criterion.
p2	API Composition	1	Thanks to Spring's data-flow starter, it is possible to compose APIs to obtain data.
p3	SAGA	1	Thanks to the JMS and ActiveMQ starter, Spring software can manage the events (transmission and reception) necessary for this criterion.
p4	Domain Event	1	Thanks to the JMS and ActiveMQ starter, Spring software can manage the events (transmission and reception) necessary for this criterion.
p5	Event sourcing	1	Thanks to the JMS and ActiveMQ starter, Spring software can manage the events (transmission, reception, subscription) necessary for this criterion.
p6	Testing of service components	1	Thanks to the Spring starter, especially the MOCK tool.
p7	Service Integration Contract Testing	1	Thanks to the cloud-contract starter, we test the integration of services.
p8	Multiple service instances per host	1	As soon as a JVM is installed on a host, Spring software can be launched and the execution port is dynamically assigned.
p9	Service instance per container	1	Thanks to the web-starter, Spring embeds its own web server, making deployment in a container extremely easy.
p10	Serverless deployment	1	Thanks to the dependency managers that exist in JAVA, it is possible to send just its source code for deployment.
p11	Externalized configuration	1	By a simple modification of the Spring configuration file, it is possible to tell it where to go to get its configuration, depending on the execution that is done.
p12	Remote Procedure Invocation (RPI)	1	As Spring uses JAVA, it embeds all the remote procedure calling techniques.
p13	Message exchange (Messaging)	1	Thanks to the JMS and ActiveMQ starter, services can exchange messages and subscribe.
p14	API Gateway	1	Spring boot offers starters to return data in almost any format including JSON, XML.
p15	Backends for frontends	1	Thanks to different starters allowing to create controllers (MVC) according to the call method, we can have several APIs per client.
p16	Service Registry	1	Several implementations are available including the most used eureka-zuul of netflix.
p17	Client-side service discovery	1	Several implementations are available including the most used eureka-zuul of netflix.
p18	Server-side service discovery	1	Several implementations are available including the most used netflix service-registry.

Continued

p19	Self-registration	1	Several implementations are available including the most used eureka-client of netflix.
p20	Circuit breaker	1	Several implementations are available including the most used Hystrix from netflix.
p21	Access token	1	By combining the security starter and the jjwt dependency of maven, we obtain a secure system by token.
p22	Log aggregation	1	Thanks to the sleuth starter and RabbitMQ, Spring allows a centralized management of the Log.
p23	Implementation measures	1	Thanks to the Actuator starter, it is possible to have the health status of the software at any time; thanks to available URLs, REST.
p24	Distributed tracing	1	Thanks to the sleuth starter and RabbitMQ, Spring allows for distributed log management.
p25	API Health Check	1	Thanks to the Actuator starter, it is possible to have the health status of the software at any time; thanks to available REST URLs.
p26	Composition of the page fragment on the server side	1	Using the thymeleaf starter, we can do server-side fragment composition and make a view functional.
p27	Composition of the client-side user interface	0	Spring is server only.
	Total		**h = 1, 26 times & h = 0, 1 time**

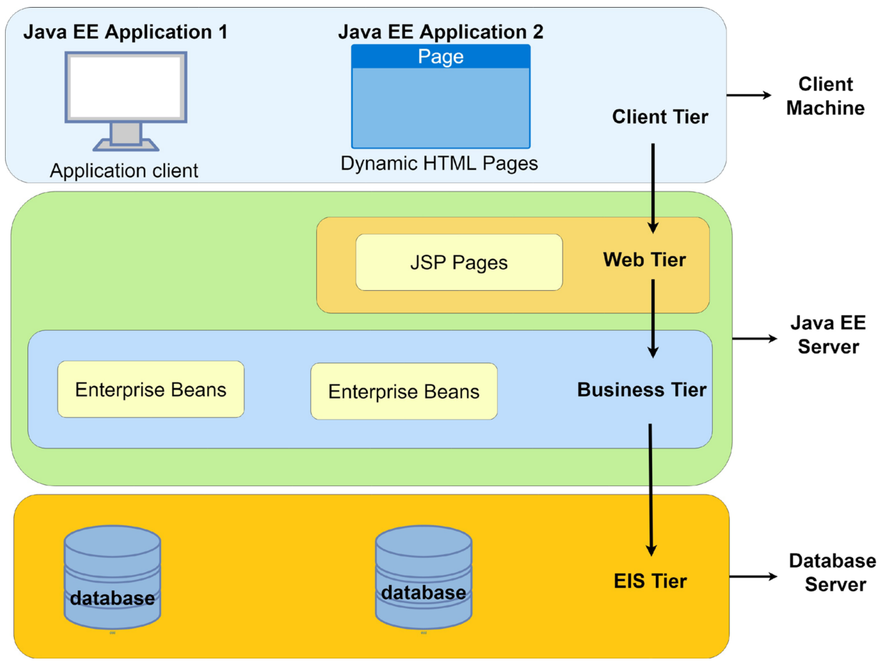

Figure 2. JAVA EE software architecture [12].

- The Model is generally managed by Java objects or JavaBeans;
- The View is managed by JSP pages.

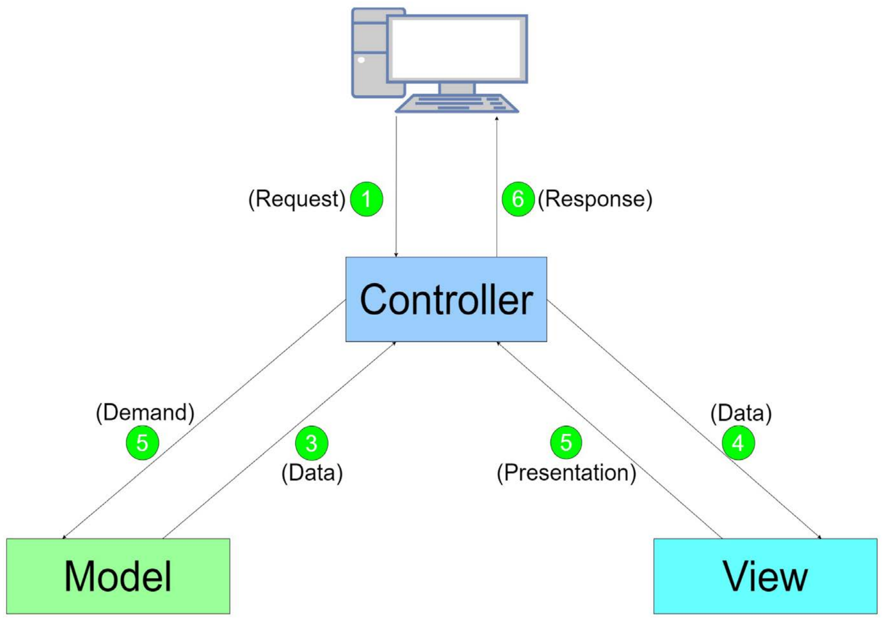

Figure 3. MVC architectural model.

From Table 3 obtained by analysis of the JAVA EE 7 technology, the value obtained for the function f is:

$$f(t,P,E)=12\,.$$

Thus, JAVA EE is **44.4%** compatible with the microservice architecture.

3.3. Discussion

For the Spring Boot platform, the score obtained is 96.3% of compatibility. This can be explained on the one hand by the fact that it is a framework based on a very popular, very rich language, with a great deal of maturity and a large community, and on the other hand by the very design of this framework, which makes it capable of evolving rapidly and integrating new packages (starters) that are configured automatically but that can also be configured as desired. These packages make development very simple and cover a wide range of needs.

For the JAVA EE platform, the score obtained is 44.4% of compatibility. However, this figure may vary depending on the software server used, which may offer additional services. This score indicates an incompatibility of the specification with the microservice style. This incompatibility can be explained by the very design of the platform. Indeed, the platform is designed in a purely SOA style and therefore does not address any of the issues introduced by the microservice style, in particular the major issues such as distributed data management, service discovery, API composition, etc.

At the end of this evaluation, one thing is clear: We have evaluated two technologies, both based on the JAVA language, but it is important to note the significant difference in scores obtained by the two. While one is very compatible, the other one is almost not. This difference can be explained in several ways.

Table 3. Table of values for the function h, for JAVA EE 7 technology.

	Evaluation Criteria	h	Justification
p1	Database per service	1	DBMS developers provide the necessary drivers to connect to their servers. JAVA being very popular, these drivers are available.
p2	API Composition	0	No implementation available
p3	SAGA	0	No implementation available
p4	Domain Event	1	The events are managed thanks to the ActiveMQ service whose driver is available.
p5	Event sourcing	1	Events are handled by the ActiveMQ service, whose driver is available.
p6	Testing of service components	0	No implementation available
p7	Service Integration Contract Testing	0	No implementation available
p8	Multiple service instances per host	0	The deployment is done in a software server and only one instance of the server can be launched. Moreover, the software runs on a port.
p9	Service instance per container	1	Software can be launched in a container.
p10	Serverless deployment	1	A JAVA EE software can be deployed without a server. Because all dependencies can be loaded on a repository.
p11	Externalized configuration	0	No implementation available
p12	Remote Procedure Invocation (RPI)	1	JAVA native remote procedure calling techniques are available.
p13	Message exchange (Messaging)	1	JAVA EE, has an API, JMS for message communication.
p14	API Gateway	1	JAVA EE, allows the implementation of REST API.
p15	Backends for frontends	0	No implementation available
p16	Service Registry	0	No implementation available
p17	Client-side service discovery	0	No implementation available
p18	Server-side service discovery	0	No implementation available
p19	Self-registration	0	No implementation available
p20	Circuit breaker	0	No implementation available
p21	Access token	1	Token-based API security is available.
p22	Log aggregation	0	No implementation available
p23	Implementation measures	1	Software control is done through the software server.
p24	Distributed tracing	0	No implementation available
p25	API Health Check	1	The software is controlled through the software server.
p26	Composition of the page fragment on the server side	1	Thanks to JSPs, we can compose fragments on the server side and make a view functional.
p27	Composition of the client-side user interface	0	JAVA EE, is server only.
Total			**h = 1, 12 times & h = 0, 15 times**

- The design of the two technologies is very different: Indeed, JAVA EE is designed in a purely SOA logic, fixed, requiring many configurations. Spring Boot, on the other hand, allows to create the desired software (SOA, microservice, REST API, command line) just by integrating the corresponding starter; it adds all the necessary dependencies and configuration to start immediately;
- The Spring Boot community is larger than the JAVA EE community: While the JAVA EE specifications come from Oracle, the starters developed by the Spring Boot community can be integrated into the official project, which makes it possible to have starters addressing almost all the issues. This is the case of Netflix, which is one of the pioneers in the field of microservices architectures and has produced many starters dedicated to the style;
- Ease of use: Indeed, thanks to its auto-configuration system, the development and deployment of Spring Boot software requires almost no configuration, nor any server, all the elements are in the jar file resulting from the compilation; whereas for JAVA EE, the configuration is manual, tedious and the deployment requires the presence of a software server previously installed.

4. Conclusions and Further Works

In this paper, we propose an evaluation framework to guide the developers in their tasks of selecting the technologies for the implementation of software-oriented microservices architectures. The proposed framework is based mainly on a set of evaluation criteria consisting of 27 architectural patterns from the domain literature and an evaluation function. This function takes into account the specific requirements of the software under development in order to assign a score to a technology that expresses the level of its compatibility with the microservice style.

This evaluation framework is applied to the Spring Boot 2.2.2 framework and the JAVA EE 7 platform under the assumptions that, all criteria have the same level of importance so each is graded to 1 and the value of the function is obtained by checking whether each criterium is implemented or not. Although both are based on the Java language, they obtained very different scores respectively 96.3% for Spring Boot and 44.4% for Java EE.

The future directions of this work are threefold. Firstly, the evaluation criteria will be extended to improve the accuracy of the evaluation. Secondly, a benchmark making a classification of existing technologies and the implementation of a support tool is necessary to automate the process of evaluating the conformity of existing software with the microservice style.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Dmitry, N. and Sneps-Sneppe, M. (2014) On Micro-Services Architecture. *International Journal of Open Information Technologies*, **2**, 4 p.

[2] Richardson, C. (2019) Microservices Pattern: Microservice Architecture Pattern. http://microservices.io/patterns/microservices.html

[3] Herold, S. (2011) Architectural Compliance in Component-Based Systems: Foundations, Specification, and Checking of Architectural Rules. Ph.D. Thesis, Clausthal University of Technology, Clausthal-Zellerfeld, Germany.

[4] Weinreich, R., Miesbauer, C., Buchgeher, G. and Kriechbaum, T. (2012) Extracting and Facilitating Architecture in Service-Oriented Software Systems. 2012 *Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, Helsinki, Finland, 20-24 August 2012, 81-90. https://doi.org/10.1109/WICSA-ECSA.212.16

[5] Gampa, S., Yazhini, Senthilkumaran, U. and Narayanan, M. (2016) Methods for Evaluating Software Architecture-A Survey. *International Journal of Pharmacy & Technology*, **8**, 25720-25733. https://www.researchgate.net/publication/316887447.

[6] Knodel, J. and Popescu, D. (2007) A Comparison of Static Architecture Compliance Checking Approaches. 2007 *Working IEEE/IFIP Conference on Software Architecture* (*WICSA*'07), Mumbai, India, 6-9 January 2007, 12 p. https://doi.org/10.1109/WICSA.2007.1

[7] Weinreich, R. and Buchgeher, G. (2014) Automatic Reference Architecture Conformance Checking for SOA-Based Software Systems. 2014 *IEEE/IFIP Conference on Software Architecture*, Sydney, NSW, Australia, 7-11 April 2014, 95-104. https://doi.org/10.1109/WICSA.2014.22

[8] Stack Overflow Developer Survey 2020 (s. d.). Stack Overflow. https://insights.stackoverflow.com/survey/2020/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2020

[9] (2021) Java EE Usage Statistics. https://trends.builtwith.com/framework/Java-EE

[10] Craig, W. (2019) Spring in Action. 5 Edition, Manning Publications, Shelter Island, New York.

[11] (2020) Spring Cloud. https://spring.io/projects/spring-cloud

[12] (2010) Distributed Multitiered Softwares—The Java EE 5 Tutorial. https://docs.oracle.com/javaee/5/tutorial/doc/bnaay.html