Software Components Selection in Microservices-based Systems

Valentina Lenarduzzi
Tampere University of Technology
Tampere, Finland
valentina.lenarduzzi@tut.fi

Outi Sievi-Korte
Tampere University of Technology
Tampere, Finland
outi.sievi-korte@tut.fi

ABSTRACT

Software is commonly developed integrating custom-developed code with existing components. Companies commonly develop only a small percentage of the whole system under development.

The research on component selection is in great need of future studies. Selecting components is one of the most important activities in a system development process and choosing the right ones is one of the key-factors

Our goal is to support developers in selecting alternative components in case the component is not working anymore or future versions cannot be used in the future.

In this paper, we present our idea reporting the approach we want to follow and the road map.

KEYWORDS

Component Selection, Microservices

ACM Reference Format:
Valentina Lenarduzzi and Outi Sievi-Korte. 2018. Software Components Selection in Microservices-based Systems. In *XP '18 Companion: XP '18 Companion, May 21–25, 2018, Porto, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3234152.3234154

1 INTRODUCTION

Software is commonly developed by integrating custom-developed code with existing components. Companies often develop only a small percentage of the whole system under development. Examples of integrated components can be software libraries, such as XML parsers, architectural frameworks such as Spring, or databases. The selection of the most suitable components is commonly carried out by means of several processes [1].

The development of microservices-based systems [2] also includes the usage of several components. Several companies are currently re-architecting their systems with a microservice pattern [28][25]. However, the microservices patterns are still not clear from practitioners point of view [26] and several practitioners still have issues in identify the best patterns and to avoid anti-patterns [27].

Microservices (MS) are relatively small, autonomous and independent services that work together, are modeled around a business capability, and have a single and clearly defined purpose [2]. Independent business processes are commonly extracted from monolithic systems and deployed in separated container as independent microservices. This, creating a distributed system with the benefits of the independence between teams but the drawback of the communication between teams that is commonly more complex [29] [30] [23].

In monolithic systems, the whole application is commonly deployed in a single web server, as example, a monolithic application can be packaged as a war file and deployed in a Tomcat server. Differently than monolithic systems, MS are independently deployed in separated containers, such as Docker containers, communicating to each other. DockerHub [1] is one of the most common cloud-based container-repository in which companies store and distribute container images. For example, developers can download the official container image of an existing product from DockerHub and deploy it in their own servers.

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Containers isolate software from its surroundings, allowing to use them in different environment with the same settings.

Containers are abstractions at the app-layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

Compared to monolithic systems, containerization opens new scenarios for the identification of components, as depicted in Figure 1:

- S1 Integration of existing components in the single MS. This case resembles monolithic systems. The single MS can be considered as an independent small-sized monolithic project. Developers can thus adopt the same selection strategies they commonly adopt in other monolithic projects [1].
- S2 Microservice mainly based on the composition of different components.
- S3 Microservice based on an existing docker (or any containerization system) image customized by the company.
- S4 Microservice composed composed solely by an existing docker (or any containerization system) image

Our goal is to support the developers in selecting alternative components in case the component is not working anymore or future updates bring mismatches in versioning, which forces to select an alternative. Therefore, we formulate two research goals:

XP '18 Companion, May 21–25, 2018, Porto, Portugal

ACM ISBN 978-1-4503-6422-5/18/05…$15.00
https://doi.org/10.1145/3234152.3234154

[1] DockerHub https://hub.docker.com/explore/

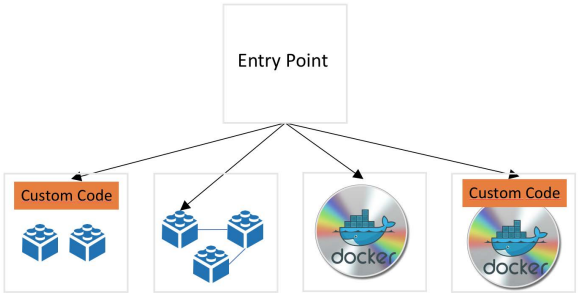

Figure 1: Different scenarios for component adoption in microservices-based systems

G1 Understand how components are actually selected in practical software development, in the context of MS-based systems. To achieve this goal we aim to characterize the motivations behind and the factors considered during the adoption of components.

G2 Create a component selection model to support developers in identifying the most suitable components and possible alternatives. Our aim is to propose a process to select the most suitable set of components together with a set of back-up alternatives.

2 BACKGROUND

The research on component selection is in great need of future studies, as supported by [3]. Selecting components is one of the most important activities in a system development process and choosing the right ones is one of the key-factors [1] for success, since from the beginning of the project when the company defined the Minimum Viable Product (MVP) [4]. Off-the-Shelf components include Components off the shelf (COTS) and Open Source Systems (OSS) [5]. The challenge is how to select among the different sourcing options. The selection of components is generally supported by component repositories and specialized source engines [5].

Ayala et al. [5] proposed a component selection process with identification, evaluation and choosing as the main common steps. Hauge et. al [6] found that the most important criteria for decision making for OSS are compliance to standards and matching the functionality provided by the component to the customer needs Another important criterion for OSS component selection is the vitality of the OSS community [14]. Architectural considerations are also considered an important selection factor [7], as well as risk factors and integration and maintenance [8]. Torchiano and Morisio [7] conducted a survey on COTS among companies, asking also about the selection criteria during COTS selection process. The participants usually listed a variety of reasons behind selection of COTS. The main reason behind selecting a component is how well-known it is to the company. The most recent work is a survey [1], conducted by Petersen et al., among 22 industrial cases in order to understand how decision making progress when a COTS is chosen. The main results highlighted that companies applied optimization approaches.

Selecting open source components has a variety of issues to be considered compared to choosing among COTS. Differently than in closed source software, with open source software developers can access the source code of the components adopted and, if needed, modify it. The processes for adopting and selecting open source software has been investigated for years, and several open source adoption models have been proposed [9], [10], [11], [12], [13].

However, as for selecting general-purpose open source software, the selection of a MS also includes trustworthiness issues. OSS is not always trustworthy, and developers do not always have all the information required to evaluate the OSS component (or the docker image containing a certain OSS product) [14], [15].

3 THE APPROACH

In this section, we explain each step of the approach, ad depicted in Figure 2

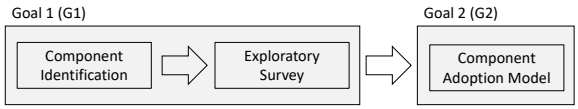

Figure 2: The Approach

Step 1: Component Identification: In the first step, we will define a selection process to support developers in identifying the most suitable components and possible alternatives. This step will be carried out by first applying the literature review proposed by Petersen et al. [1] in the context of MS.

Step 2: Exploratory Survey: In order to understand how developers select components for their MS-based systems, we will conduct an exploratory survey among different companies. The goal is to understand the different motivations behind and factors considered for the components adoption, also considering the scenarios highlighted in Section 1. This step will be carried out replicating the approach adopted in [12].

Step 3: Component adoption model: Based on the results obtained in the literature review, and based on the factors highlighted in the survey carried out in the second step , we will propose an initial component adoption model. The model will support developers in understanding in which context using container images is more effective than (a) customizing components, (b) deploying them in containers and (c) identify strategies for the replacement of existing components in case the current component break the system. The component adoption model will be defined extending [15] and [17]. We designed this step as an exploratory case survey [19]. Exploratory case surveys bridge the gap between surveys and case studies. Since we cannot plan in advance the number of available case studies, this is by far one of the most suitable approach because every new case study analyzed will add value to previous individual cases, and the individual results can be incorporated to draw conclusions in the quantitative synthesis.

The process will be based on four sub-steps:

(1) Cases study selection. In this step, we will identify a set of companies working with MS, including both companies developing in-house and outsourcing of open-source intensive.

The target participants are in roles with decision making power in selecting components.

(2) Data extraction design the questionnaire for eliciting the main process, motivations, and characteristics adopted in the selected cases.
(3) Conduct the coding. In this step we will apply open and selective coding techniques. Results will be analyzed in a focus group composed by experts in the domain of MS and driven by experts in domain of empirical software engineering.
(4) Analysis of the coding with statistical approaches.
(5) Exploit machine learning techniques to discover hidden rules in the data about the selection of the components for microservices.

4 ROAD MAP

We already analyzed the literature on component adoption both in the context of monolithic systems and MS-based systems. Currently, there are no studies considering MS or containerizations. We are identifying a set of Open Source projects, so as to understand common patterns in component selection and identification. Projects will be of different domains and different size, both considering lines of codes and functional size [16]. The selected projects will be shared among different works and teams [20] [24] [22] [23], so as to validate their fitness for different purposes and to increase the quality of the collected data, also considering different information. We are currently designing the data extraction form, combining the questionnaires adopted in [1], [15], [17] and [18]. The first questionnaire will be adopted in a trial case so as to test it before starting interviewing companies. We are planning to conduct the first set of interviews during XP2018.

REFERENCES

[1] K. Petersen et al. Choosing Component Origins for Software Intensive Systems: In-house, COTS, OSS or Outsourcing? - A Case Survey. Transactions on Software Engineering, vol. PP, no. 99, pp. 1-1. (2017)
[2] J. Lewis and M. Fowler. MicroServices. (2014)
[3] D. Badampudi, C. Wohlin, and K. Petersen. 2016. Software component decision-making: In house, OSS, COTS or outsourcing - A systematic literature review. Journal of Systems and Software. Vol 121, pp. 105-124. (2016)
[4] D. Taibi and V. Lenarduzzi. MVP explained: A Systematic Mapping on the Definition of Minimum Viable Product. Euromicro Conference on Software Engineering and Advanced Applications (SEAA2016). pp. 112-119. (2016)
[5] C. Ayala, O. Hauge, R. Conradi, X. Franch, J. Li. Selection of third party software in Off-The-Shelf-based software development-An interview study with industrial practitioners.
[6] O. Hauge, T. Osterlie, C.F. Sorensen, and M. Gerea. An empirical study on selection of Open Source Software - Preliminary results. Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. pp. 42-47. (2009)
[7] M. Torchiano, M. Morisio. Overlooked Aspects of COTS-Based Development. IEEE Software. Vol. 21(2). pp. 88-93. (2004)
[8] J. Li, R. Conradi, O. P. Slyngstad, M. Torchiano, M. Morisio and C. Bunse, A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software Components. Transactions on Software Engineering, vol. 34, no. 2, pp. 271-286. (2008)
[9] D. Taibi, L. Lavazza and S. Motrasca. OpenBQR: a framework for the assessment of OSS.International Conference on Open Source Software. pp. 173-186. (2007)
[10] A. Wasserman, P. Murugan and C. Chan. The business readiness rating model: an evaluation framework for open source. EFOSS Workshop. (2006)
[11] Method for Qualification and Selection of Open Source software (QSOS) version 1.6. Atos Origin.(2006)
[12] V. Del Bianco, L. Lavazza, S. Morasca and D. Taibi. A Survey on Open Source Software Trustworthiness. IEEE Software, vol. 28, no. 5, pp. 67-75. (2011)
[13] A. Wasserman, et al. OSSpal: Finding and Evaluating Open Source Software. Open Source Systems: Towards Robust Practices. Springer International Publishing. pp. 193-203. (2017)
[14] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi. An empirical investigation of perceived reliability of open source Java programs. In Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12). pp.1109-1114. (2012)
[15] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi. Predicting OSS trustworthiness on the basis of elementary code assessment. International Symposium on Empirical Software Engineering and Measurement. Art. 36 , 4 pages. (2010)
[16] V. Lenarduzzi, I. Lunesu, M. Matta and D. Taibi. Functional Size Measures and Effort Estimation in Agile Development: A Replicated Study. Agile Processes in Software Engineering and Extreme Programming. pp. 105-116. (2015)
[17] V. Del Bianco, L. Lavazza, S. Morasca, D. Taibi, and D. Tosi. The QualiSPo approach to OSS product quality evaluation. International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. pp. 23-28. (2010)
[18] S. Morasca, D. Taibi, and D. Tosi. Towards certifying the testing process of Open-Source Software: New challenges or old methodologies?. Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. pp. 25-30. (2009)
[19] R. Larsson. Case Survey Methodology: Quantitative Analysis of Patterns Across Case Studies. Academy of Management Journal, vol. 36 n.6. (1993)
[20] L.Pellegrini, V. Lenarduzzi. Are Code smells the root cause of faults? A continuous experimentation approach. 19th International Conference on Agile Processes in Software Engineering and Extreme Programming. XP 2018. (2018)
[21] V.Lenarduzzi, I.Lunesu. The Lean Strategies: A tale of user-driven innovation processes. 19th International Conference on Agile Processes in Software Engineering and Extreme Programming. XP 2018. (2018)
[22] V.Lenarduzzi, I.Lunesu, M.Marchesi, R.Tonelli. Blockchain applications for Agile methodologies. 19th International Conference on Agile Processes in Software Engineering and Extreme Programming. XP 2018. (2018)
[23] V.Lenarduzzi, O.Sievi-Korte. On the negative impact of team independence in microservices software development. XP 2018 Scientific Workshops. 2th International Workshop on Microservices: Agile and DevOps Experience (MADE18) (2018)
[24] F. Auer, M. Felderer and V. Lenarduzzi. Towards Defining a Microservice Migration Framework. XP 2018 Scientific Workshops. 2th International Workshop on Microservices: Agile and DevOps Experience (MADE18) (2018)
[25] D.Taibi, V. Lenarduzzi and C. Pahl. Processes, Motivations and Issues for Migrating to Microservices Architectures: An Empirical Investigation. IEEE Cloud Computing. Vol. 4(5), Art. 8125558. pp. 22-32. (2017)
[26] D. Taibi, V. Lenarduzzi and C. Pahl. Architectural patterns for microservices: a systematic mapping study. International Conference on Cloud Computing and Services Science (CLOSER2018), Funchal (Madeira) 19 - 21 March 2018, In Press.
[27] D. Taibi and V. Lenarduzzi. On the Definition of Microservices Bad Smells. IEEE Software. Vol 35, Issue 3, May/June 2018
[28] D.Taibi, V. Lenarduzzi, C. Pahl and A. Janes. Microservices in Agile Software Development: First results on Issues, Advantages, and Disadvantages. First International Workshop on Microservices for Agile software development (WMSA17), art. 23. (2017)
[29] D. Taibi, V. Lenarduzzi, A. Janes, M.O. Ahmad and K. Liukkunen. Comparing Requirements Decomposition Within the Scrum, Scrum with Kanban, XP, and Banana Development Processes. Agile Processes in Software Engineering and Extreme Programming (XP2017). pp. 68-83. (2017)
[30] D. Taibi, V. Lenarduzzi, M. O. Ahmad, and K. Liukkunen. Comparing Communication Effort within the Scrum, Scrum with Kanban, XP, and Banana Development Processes. 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17). pp. 258-263. (2017)