# Comparison Parameters

The parameters to measure the languages in this research are derived from the factors that are important for microservices and the general-purpose programming languages. According to Guidi et al. (2017) the important factors for microservices from a linguistic approach is the separation of behavior from deployment. This can be possible if the programming language treats Interfaces as the first-class entity. Besides Interfaces, Ports and Workflows are also important for microservices. The other factors important for the general purpose programming language is verbosity of the source code, size of the executables and the execution time of the code (Nanz & Furia, 2015). Apart from this Debug feature is also important for software development.

The research will talk in brief about each parameter that is used for the comparison of the language.

## Interfaces and Port

Microservices are deployed as the “black boxes” whose implementation details are hidden in order to support modular programming. The details of the functionality of the services is provided by the interfaces. Interfaces describe the set of operations that can be remotely invoked. Thus, interfaces become the first-class citizen of the microservices. Thus, it is important for a microservices languages to provide the set-theoretical operators like union and intersection that can work with the interfaces (Guidi et al., 2017).

Besides Interfaces, ports are important as communication port describes how the services are made available to the network. Each service may be equipped with many ports thus ports should be separate from the implementation of the service. Hence, the language for microservices should provide the capability to separate the port from the implementation of the service (Guidi et al., 2017).

This research will compare the three languages based on the support to have interfaces and parameter as the first-class citizen of the language. This can be demonstrated by the sample code built on the all the three languages.

## Graphical View Support

It is a common practise to draw the sequence diagram for the complex integration scenarios (Weerawarana et al., 2018). Also, there are different workflow languages like BPEL being used by the industry experts to simplify the complex integration scenarios (Guidi et al., 2017). Thus, it is helpful for the microservices language to provide the visual support or the workflow development options to support the development of microservices. This research will test the three languages for its capability to support visual designing.

## Verbosity:

Term Verbose means explaining in words more than needed. According to the developer community StackOverFlow, majority of the developers supported that a good code should be easy to comprehend at a glance (Why Is Verbosity Bad for a Programming Language?, n.d.). Also, according to the Gupta (2004), excess verbosity can cause the programmers at the beginner level to miss the conceptual learning in order to get the grasp of the huge code. When a programming language is less verbose it also mean that it is capable to do a task in less number of lines which also saves the development time with respect to writing code (Krishna et al., 2017). Considering all this factors, the line of code for each use case will be measured and the languages will be evaluated on the numbers obtained from the experiment.

## Size:

Size of the executable has a great impact on the deployment of the code. Reduced size of executable has several benefits like faster deployment, smaller disk size, small server space, reduced cost of hardware and easy management of code (Cepa, 2005). The factors that influence the size of the code is the use of third-party libraries or the external frameworks to support a development. However, the same can be avoided if the language development environment itself provides those libraries. (How Ballerina  Is Different From Other Programming Languages - DZone Integration, n.d.). In this study the executable built for every language will be evaluated against size of the executables for every integration scenario.

## Execution Time

It is a well-known fact that faster applications are preferred over the slow performing applications. Execution time forms an important parameter to measure the performance of any application. A new technology is always assessed on its performance and execution time is one of the parameter to measure the performance (Pongnumkul et al., 2017). In this research, for every use case the execution time will be recorded and compared for all the languages.

## Debug

Debugging is useful in various stages of software development like design where the error made in writing the code is identified, secondly it is also helpful in the later stages of testing. Debugging is also helpful in the error diagnostics in the production environment (Cheng et al., 2017).

All the three languages will be executed in the debug environment to check for the support of this feature.

# Integration Technology

It is a common practice in the IT industry to use the integration technology that is language or technology agnostic (Daya et al., n.d.). Being language-neutral makes it easier to use existing skills of the resources and does not limit the software development to one language. There are many such language agnostic technologies and the most widely adopted ones are REST, SOPA and RPC. Databases are the integral part of the microservices, hence in this research database is also include in the list of other technologies (Newman, 2015).

TODO:

Give overview of all the integration technologies

Integration Scenario

//Examples listed in the UBL spec

# References

Cepa, V. (2005). Using Reflection to Reduce the Size of. NET Executables. *J. Object Technol.*, *4*(7), 51–64.

Cheng, C.-T., Lee, Y.-H., Liu, C.-H., & Wu, M.-T. (2017). *Error diagnostic in a production environment*. Google Patents.

Daya, S., Duy, N. V., Eati, K., Ferreira, C. M., Glozic, D., Gucer, V., Gupta, M., Joshi, S., Lampkin, V., Martins, M., Narain, S., & Vennam, R. (n.d.). *Microservices: From Theory to Practice*. 170.

Guidi, C., Lanese, I., Mazzara, M., & Montesi, F. (2017). Microservices: A language-based approach. In *Present and Ulterior Software Engineering* (pp. 217–225). Springer.

Gupta, D. (2004). What is a good first programming language? *XRDS: Crossroads, The ACM Magazine for Students*, *10*(4), 7–7. https://doi.org/10.1145/1027313.1027320

*How Ballerina  Is Different From Other Programming Languages—DZone Integration*. (n.d.). Retrieved June 9, 2020, from https://dzone.com/articles/ballerinawhy-it-is-different-from-other-programmin

Krishna, R., Menzies, T., & Layman, L. (2017). Less is more: Minimizing code reorganization using XTREE. *Information and Software Technology*, *88*, 53–66.

Nanz, S., & Furia, C. A. (2015). A comparative study of programming languages in rosetta code. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, *1*, 778–788.

Newman, S. (2015). *Building microservices: Designing fine-grained systems* (First Edition). O’Reilly Media.

Pongnumkul, S., Siripanpornchana, C., & Thajchayapong, S. (2017). Performance analysis of private blockchain platforms in varying workloads. *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 1–6.

Weerawarana, S., Ekanayake, C., Perera, S., & Leymann, F. (2018). Bringing middleware to everyday programmers with ballerina. *International Conference on Business Process Management*, 12–27.

*Why is verbosity bad for a programming language?* (n.d.). Software Engineering Stack Exchange. Retrieved June 8, 2020, from https://softwareengineering.stackexchange.com/questions/141175/why-is-verbosity-bad-for-a-programming-language

More references –

<https://go-gale-com.whitireia.idm.oclc.org/ps/i.do?p=AONE&u=whitireia&id=GALE|A176617962&v=2.1&it=r&sid=summon>