An Evaluation of Big Data Architectures

Valerie Garises and José G. Quenum
Namibia University of Science and Technology, 13 Jackson Kaujeua, Windhoek, Namibia

Keywords: Big Data, Software Architecture Analysis, Software Architecture Evaluation, Software Architectural Patterns.

Abstract: In this paper, we present a novel evaluation of architectural patterns and software architecture analysis using *Architecture-based Tradeoff Analysis Method* (ATAM). To facilitate the evaluation, we classify the Big Data intrinsic characteristics into quality attributes. We also categorised existing architectures following architectural patterns. Overall, our evaluation clearly shows that no single architectural pattern is enough to guarantee all the required quality attributes. As such, we recommend a combination of more than one pattern. The net effect of this would be to increase the benefits of each architectural pattern and then support the design of Big Data software architectures with several quality attributes.

1 INTRODUCTION

The ongoing Big Data challenge brings an increasing burden to software and system architects. Usually, they select a tool/infrastrcture (e.g., Apache Hadoop, Apache Spark) that satisfies most big data functions. (Garises and Quenum, 2018) pondered over the sustainability of this approach, whether it makes optimal use of resources and how it deals with maintenability. It argued the need for a reference architecture where the tools and infrastructure are selected and assembled suitably.

One critical step towards defining the reference architecture is to identify the central components that should make up a reference architecture as well as their roles and responsibilities. In this paper, we revisit existing big data architectures and analyse them using a software architecture evaluation method. The IEEE standard 1471 (W. Maier et al., 2000) defines a software architecture as the fundamental organisation of a system embodied in its components, their relationships to each other and the environment and the principles guiding its design and evolution. A software architecture is generally defined as a logical construct for defining and controlling the interfaces and the integration of all the components of the system. Similarly, (Bass et al., 2003) define it as the structure or structures of the system, which comprise components, their externally visible behaviour, and the relationships among them. In short, a software architecture is about a structure formed by components and the connections between them.

Furthermore, a software architecture is usually arrived at following an *architectural pattern*. According to (Garlan and Shaw, 1994), a pattern describes a problem which occurs over and over in the environment and then describes the core of the solution of that problem so that it can be applied a million times over. An architectural pattern thus determines the various components and connectors that can be associated with the instances of that style along with a set of constraints on their combination. Common architecture patterns include *Layers*, *Pipes* and *Filters*, *Model View Controller* (MVC), *Broker*, *Client-Server* and *Shared Repository*.

In this work, we selected various big data architectures and evaluated them using the *architecture-based trade-off analysis method* (ATAM); a scenario-based evaluation method. In order to conduct the evaluation, we mapped each one of them to an architectural pattern. Our evaluation reveals that no architectural pattern fulfils the requirements for Big data systems. It thus underscores the need for a combination of these architectural patterns in a big data system.

The remainder of the paper is as follows. Section 2 discusses architecture evaluation methods. Section 3 discusses the actual evaluation of selected big data architectures. Section 4 summarises our findings, while Section 5 draws some conclusions and sheds light on future work.

Garises, V. and Quenum, J.
An Evaluation of Big Data Architectures.
DOI: 10.5220/0007840801520159
In *Proceedings of the 8th International Conference on Data Science, Technology and Applications (DATA 2019)*, pages 152-159
ISBN: 978-989-758-377-3

2 SOFTWARE ARCHITECTURE ANALYSIS METHODS

The aim of analysing an architecture is to predict the quality of a software system before it is built. Evaluating an architecture, on the other hand, is to compare it with another architectural candidate or with a set of requirements in order to minimise the risks and provide evidence that the fundamental requirements have been addressed (Kazman et al., 2000). There are several software architecture evaluation methods such as *experience-based*, *scenario-based*, *mathematical modelling*, and *simulation-based* (Ali Babar et al., 2004). These evaluation methods are applicable in different phases of the software development cycle. Overall, a software architecture can be evaluated before, during and after implementation. In this paper, we focused on the scenario-based evaluation method. *Architecture-based Trade-off Analysis Method* (ATAM).

In this paper, we set out to evaluate multiple architectures using multiple quality attributes. As such ATAM proves to be a good fit for our evaluation. ATAM is generally aimed at evaluating architectures after the design phase, but before they are implemented. However, in some cases, it has been used on deployed systems. In a survey, (Al-Jaroodi and Mohamed, 2016) identified common requirements for big data systems. These are:

R_1**: Scalability.** Big Data systems must be scalable, i.e., be able to increase and support different amounts of data, processing them uniformly, allocating resources without impacting costs or efficiency. To meet this requirement, it is required to distribute datasets and their processing across multiple computing and storage nodes;

R_2**: High Performance Computing.** Big Data systems must be able to process large streams of data in a short period of time, thus returning the results to users as efficiently as possible. In addition, the system should support computation intensive analytic, i.e., support diverse workloads, mixing request that requires rapid responses with long-running requests that perform complex analytic on significant portions of the data collection (Gorton and Klein, 2015);

R_3**: Modularity.** Big Data systems must offer distributed services, divided into modules, which can be used to access, process, visualise, and share data, and models from various domains, providing flexibility to change machine tasks;

R_4**: Consistency.** Big Data systems must support data consistency, heterogeneity, and exploitation. Different data formats must be also managed to represent useful information for the system (Cecchinel et al., 2014). Besides that, systems must be flexible to accommodate this multiple data exchange formats and must achieve the best accuracy as possible to perform these operations;

R_5**: Security.** Big Data systems must ensure security in the data and its manipulation in the architecture, supporting the integrity of information, exchanging data, multilevel policy-driven, access control, and prevent unauthorised access (Demchenko et al., 2016);

R_6**: Real-time Operations.** Big Data systems must be able to manage the continuous flow of data and its processing in real time, facilitating decision making;

R_7**: Inter-operability.** Big Data systems must be transparently intercommunicated to allow exchanging information between machines and processes, interfaces, and people (Santos et al., 2017); Thus, it must facilitate interoperability between disparate and heterogeneous systems, both existing and future;

R_8**: Availability.** Big Data systems must ensure high data availability, through data replication horizontal scaling, i.e., spread a data set over clusters of computers and storage nodes, avoiding bottlenecks (Gorton and Klein, 2015)

Since all these architectural requirements are classified as *non-functional* requirements, we mapped them onto a known quality model, specifically ISO/IEC 25010 (ISO/IEC, 2010). Table 1 provides a snapshot of this mapping.

Table 1: Mapping of Big Data Requirements to Quality Attributes of ISO/IEC 25010.

Requirements	Quality Attributes
Consistency	Not covered
Scalability	Portability: adaptability
Real-time Operation	Performance efficiency: time-behavior
High Performance Computing	Performance efficiency: time-behavior and resource utilisation
Security	Security: confidentiality, integrity and authenticity
Availability	Reliability: availability and recoverability
Modularity	Maintainability: modularity
Inter-operability	Compatibility: inter-operability

It should be noted that consistency is not covered by the quality model. Consistency represents the ability of all integrated information, whether from

one data source or many, to be *cohesive* and *usable*. The non-coverage of this attribute may be due to the fact that ISO/IEC 25010 represents a quality model for systems, while consistency represents information quality.

Due to the exponential growth of data over the last decade and, consequently, the increase in the number of Big Data systems, it is necessary to resort to architectural patterns for the development of these systems. Moreover, the Big Data requirements of software architectures in those systems can directly impact the outcome of the software quality challenges. Hence, the characterisation of these architectures against the requirements is fundamental.

3 ARCHITECTURE EVALUATION WITH ATAM

3.1 Business Drivers

As part of the ATAM process, business drivers were identified from the requirements. Here, we used the context of the Namibia healthcare use case and architectural requirements outlined in Section 2. The core business drivers are listed below and are mapped to the Big Data architectural requirements from which they were derived.

Modifiability is important as messaging standards may change over time and new transactions, orchestrations and health information systems (HIS) may be added over time (R_1 and R_4);

Scalability and Performance are important as such a system may be deployed at a national level and it should remain functional at scale (R_1 and R_2);

Availability is essential as the Big Data interoperability infrastructure needs always to be available so that vital health data can be captured and retrieved around the clock (R_6 and R_8);

Inter-operability is vital as the architecture should be able to communicate and exchange data with to a wide variety of environments (R_3 and R_7);

Security is important as a patient's health information is highly confidential and should not be tampered with or viewed by unauthorised parties (R_5).

3.2 Big Data Architectures

We selected several known Big Data reference architectures in the academia and industry. These include *Big Data Architecture Framework*, *Big Data Enterprise Model*, *Big Data Analytics Architecture in Healthcare*, *Microsoft Big Data Ecosystem Architecture*, *IBM Big Data Analytics Reference Architecture* and *Oracle Reference Architecture*. In the following subsections we describe some of these architectures. These architectures were analysed retrospective as some are already implemented.

3.2.1 Big Data Analytics Architecture in Healthcare

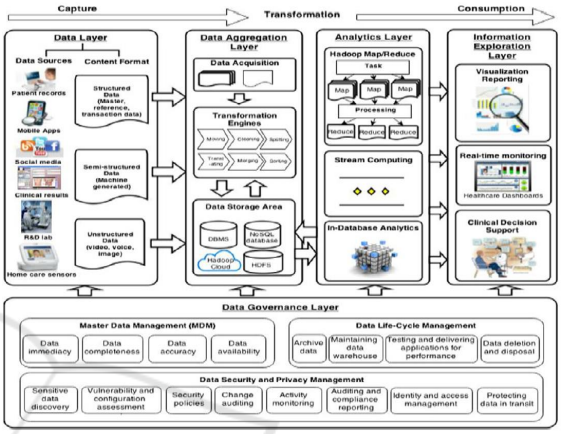

Figure 1: Big Data Analytics Architecture in Healthcare.

(Wang et al., 2018), define a Big Data analytics architecture embedded in the concept of data lifecycle framework that starts with data capture, proceeds via data transformation and concludes with data consumption. Figure 1 depicts the proposed best practice Big Data analytics architecture that loosely comprises five major architectural layers: (1) data, (2) data aggregation, (3) analytics, (4) information exploration, and (5) data governance. These logical layers make up the Big Data analytics components that perform specific functions, and will, therefore, enable healthcare managers to understand how to transform the healthcare data from various sources into meaningful clinical information through Big Data implementations.

3.2.2 IBM Big Data Reference Architecture

IBM introduces Big Data and Analytics Reference Architecture with eleven components as depicted in Figure 2. The reference architecture is intended to be used by sales professionals selling IBM software and designing end-to-end Big Data and analytics client solutions.

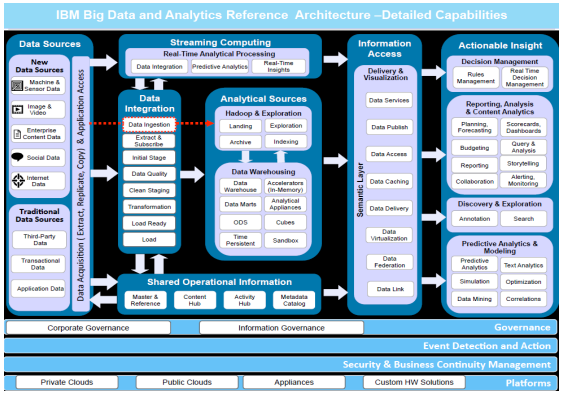

Figure 2: IBM Big Data Reference Architecture Components.

3.3 Underlying Architectural Patterns

In Table 2, we present a summary of the architectures above and their references and match the architectural patterns identified in each one.

Table 2: Big Data Reference Architectures and Architectural Patterns.

ID	Title and Reference	Architectural Pattern
A_1	*Big Data Architecture Framework* (BDAF)	**Pipes and Filters**
A_2	*Big Data Enterprise Model*	**Shared Repository**
A_3	*Big Data Analytics Architecture in Healthcare*	**Pipes and Filters**
A_4	*Microsoft Big Data Ecosystem Architecture*	**Pipes and Filters**
A_5	*IBM Big Data Analytics Reference Architecture*	**Pipes and Filters**
A_6	*Oracle Reference Architecture*	**Layered**

3.4 Quality Attributes and Utility Tree

Following step 5 of the ATAM process, a utility tree was drawn up as depicted in Figure 3. The utility tree maps scenarios derived from the architectural requirement of the architecture to the quality attributes that the scenarios fall under. To construct the utility tree many scenarios that the architecture needs to address must be elicited. The high-level requirements that were derived from the Namibia healthcare use case were used to formulate these scenarios along with known scenarios gathered. The utility tree contains a list of ten scenarios that have been prioritized and mapped to the particular quality attribute that it addresses.

Furthermore, following Step 7 in the ATAM process, the evaluation scenarios that appeared in the utility tree were prioritized along two dimensions. The

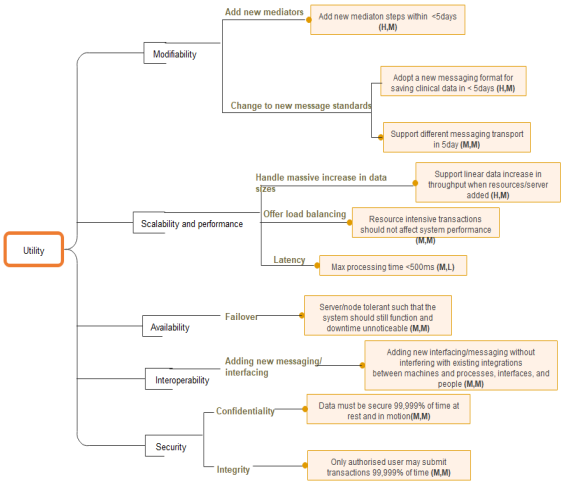

Figure 3: Utility Tree.

first dimension is the importance of the scenarios to the success of the architecture and the second dimension is the anticipated difficulty in achieving this scenario. These ratings can be seen in brackets on the utility tree scenarios, rated on a scale of high (H), medium (M) and low (L). Using these priorities, the highest rated quality attributes were identified. These attributes are the most important to the success of the architecture which also defines its quality. The attributes of high importance are chosen as the attributes that will be evaluated to determine the overall quality of the architecture.

Moreover, ATAM categorises the architectural attributes according to two risk categories namely runtime (availability, performance, availability) and development (modifiability, integration). Hence, the total architecture quality is a function that consists of modifiability, scalability, performance, interoperability, and security attributes. In other words, the overall HLM architecture quality can be defined as: $Q_{Arch} = f(Q_{Mod}, Q_{Sca}, Q_{per}, Q_{Sec}, Q_{inter})$. However, this analysis will only focus on the architecture development quality attributes which is a subset of function Q_{Arch}.

3.5 Analysing the Architectural Patterns

3.5.1 Pipes and Filters Pattern

In this architectural pattern data comes from an external source or producer and is managed through several steps, such as data collection, preparation, processing, and visualisation. Finally, the results of such processing are sent to consumers and or domain applications. In Table 2, four of the six architectures have

the Pipe and Filter (PF) architectural pattern. More details about the PF pattern can be found in (Bass et al., 2003).

- **Scenario 1 (Context).** This refers to the first scenario under the utility tree (Figure 3) under Scalability and Performance attribute. Big Data systems need to "handle a massive increase in data sizes" to extract useful value from this massive volume of data generated in real time. To meet this requirement data extraction must pass through several steps (e.g., data collection, data integration, pre-processing, processing, and data visualisation), which could support the overall data mining process. These steps have to be executed in a specific order to complete the accurate data processing and analysing.
- **Problem.** The Big Data architectures must support the execution of efficient data flows real-time as well as batch processing. The communication between the steps must be scalable, avoiding bottlenecks in performance that could result in failures or delays or inconsistencies in the data mining steps. Data must be processed independently in each step, and thus sent to the next to maintain real-time operation.
- **Solution.** This pattern structures the processing of a stream of data (Geerdink, 2015). Each processing step is implemented as a *filter*, with information flowing between the filters through a *pipe*. Filters can be combined in many ways to provide a family of solutions. Its importance is justified because the sequence of data analysis is essential. For example, it is not possible to extract value from data that have still not been preprocessed. This pattern increases complexity and could support data transformation, implying on high dependence between the filters (Buschmann et al., 2007).
- **Trade-offs.** This is one of the closest related patterns to the Big Data context because it was proposed to solve data processing issues. However, only such data flow is not sufficient to ensure that there will be quality in performance. Pipes and filters implementation usually achieve scalability and performance, but several Big Data specificities impact its execution and may minimise its benefits. Firstly, a high volume of data is generated in real-time and should be consumed by the filters and processed effectively. However, some of the pipe steps require that the entire dataset be loaded before processing and since there is no alternative way, processing queues are generated, thus impacting scalability, response time, and directly performance. The veracity of the data can also take risks in this pattern. There are no intermediary sources of input and output between the filters, that is, the data is the input of the first, and the extracted value of the data is the output of the last one. This increases the complexity (i.e., volume, variety) of the data causing some failures in the middle of the execution, and such failures can be propagated resulting in changes or even data loss. This indicates a central point of failure and at the same time a liability to safety.

3.5.2 Shared Repository Pattern

In this architecture, there is a shared repository to communicate to all the architecture components, i.e., components for Big Data processing, technological modules, the broker that process data generated by several sources, and the server connected to the domain applications. This repository is responsible for receiving requests from all these components and maintain data availability and consistency (Chen et al., 2015).

- **Scenario 2 (Context).** This refers to the first and second scenarios under the utility tree (Figure 3) under Modifiability attribute. Also, to the massive volumes of data which managed and processed it must be able to handle a variety of data and messaging types to transfer data to different formats, domains and systems. These systems must be flexible to "add or remove mediators and messaging standards" while processing and storing the vast variety of data types. The processed data must be shared in different formats to different consumers, devices, application, systems, people and domains.
- **Problem.** In this scenario, preparing a Big Data repository often involves more mediators and messaging standards and challenges when such data must be shared among different consumers, producers and domains in real-time. Shared processing, storage and management also increase a control issue related to how and when data will be available in order to be read or written in the shared memory and storage.
- **Solution.** The shared repository pattern consists of using a data repository as communication among different software components crossing domain boundaries. Information produced by a component is stored in the shared repository, and other components will retrieve it if they request such data. The repository manages the common data handling application's operation and

their permissions to access and modify the repository. Thus, this pattern reduces the communication flows by allowing only requested data, which entails a gain in performance in Big Data systems. Reuse of data benefits the third party, who gets information from a common shared repository.

- **Trade-offs.** The main benefits of this pattern include the combination with different storage patterns for improving the way large volumes of data are stored using cloud and distributed systems. However, the shared repository pattern has some drawbacks in the performance and scalability that can be severely affected in distributed environments by concurrent access to the data. Shared repository transmission can fail, and the whole system is immediately affected by losing the synchronisation with other services as well as real-time response. In addition, modifying the structure, mediators, messaging standards of the shared repository has a negative impact on Big Data systems resulting in modifications in the interfaces and software communication systems.

3.5.3 Layered Pattern

In this architectural pattern the architecture is divided in the functionalities into layers according to the different steps of data analytics (e.g., storage, publisher, sources, and applications). Moreover, (Santos et al., 2017) proposed a common layer in Big Data architectures, focusing on security, management and monitoring, which includes components that provide base functionalities needed in the other layers, ensuring the proper operation of the whole infrastructure. Communication and data flow between the layers can be implemented through interfaces and APIs.

- **Scenario 3 (Context).** This refers to the scenario under the utility tree (Figure 3) under Interoperability attribute. Big Data systems require the separation of roles among their components. Such division must be established at both the module level (i.e., data processing, data storage, data visualisation, data integration) and at the abstraction level (i.e., communication, security, management). However, these different components must be tightly integrated and interoperable to exchange data and messages among the different layers. Besides that, these systems must orchestrate the flow of data to execute business processes, and hence to ensure that the objectives of the system are fulfilled and that data is not lost.
- **Problem.** From the above scenario, Big Data architectures must allow the division of responsibilities, by grouping them based on goals and functionalities. Such architectures must provide means to process data in real-time, because sometimes a specific workflow depends on the results of the previous one. Moreover, the communication protocols must guarantee high performance and data scalability, avoiding bottlenecks.
- **Solution.** The layered pattern provides a horizontal division of software responsibilities by grouping a set of functionalities in encapsulated and independent layers. The specific partitioning criteria can be defined along various dimensions, such as abstraction, granularity, hardware distance, and rate of change. Layers of an architecture generally communicate with adjacent layers through implemented interfaces. Therefore, one layer can make requests to the next layer, and wait for the response while responding to requests from the previous layer. Such interfaces must be scalable and implement communication protocols to ensure efficiency.
- **Trade-offs.** The waiting time and the execution of communication protocols may delay processing and cause the response to not be as fast as expected. Besides, these communication protocols impact scalability, since the high volume of data must be processed at the same time and can cause a bottleneck between the layers. Finally, because data have to go through different layers, the processing performance is directly impacted. Regarding Big Data requirements, this pattern supports the modularity by providing a division of concerns in the application. The independence of each layer enables the implementation of different protocols for keeping the data safe to be sent only when necessary, allowing a positive impact on the security requirement of Big Data. Considering weaknesses, three of the most important Big Data requirements are minimised by this pattern (i.e., Real-Time Operation (R_6), Scalability (R_1), and High-Performance computing (R_2)). Big Data requires the processing of data at run-time because a specific layer depends on the result of the previous one. Hence, the wait time and the execution of communication protocols may delay processing and cause the response not to be as fast as expected. These communication protocols impact scalability, since the high volume of data, must be processed at the same time and can cause a bottleneck between the layers. Finally, because data have to go through different layers, the processing performance is directly impacted.

4 EVALUATION RESULTS

From the evaluation discussed in Section 3 it is clear that Big Data systems must be designed considering their 5V's intrinsic characteristics of Volume, Velocity, Variety, Veracity, and Value. From a perspective of software architecture, it is important to define the requirements of quality attributes that must be fulfilled by those systems. In Table ref, we summarise the results, mapping the Big Data characteristics into quality attributes and therefore, indicating which architectural patterns meet individually the Big Data requirements.

Table 3: Quality Requirements vs Architectural Patterns.

Characteristics of Big Data Systems	Quality Attributes	Layered	Shared Repository	Pipes and Filters
Volume, *Velocity*	Scalability	-	-	-
Volume, *Velocity*	Performance	-	-	-
Volume	Modularity	+	-	+
Veracity	Consistency		+	-
Veracity	Security	+	+	-
Velocity	Real-time	-	-	-
Variety, *Value*	Interoperability			
Value	Availability		-	

Scalability, performance and real-time operation are related to the velocity at which data is generated. The system needs to be scalable to handle this velocity, as well as processing this data in real time to avoid bottlenecks and not impact performance. In terms of the characteristic of volume, the related attributes are scalability, performance, and modularity. Big Data systems need to adapt to the high and low volumes of data that are generated at run-time. In addition, these data need to be executed with high performance, and if possible, in different modules, to increase the modularity and division of responsibilities. Data veracity has aspects of security and consistency. The data processed must remain consistent throughout the processing flow, and backups must be guaranteed so that data can be recovered if the system fails. In addition, communication between the modules must be secure to avoid data losses and data alteration during this process. The variety of data is addressed by interoperability requirements, which must ensure that data provided by different systems and different formats are processed for value acquisition. Finally, availability is related to value acquisition for ensuring that the generated knowledge and data is available to stakeholders and to execute desired business processes.

As expected, none of the patterns directly fulfil all Big Data requirements. This is mostly due to the fact that each of them was proposed to address a specific problem. Moreover, Performance, scalability, and availability are one of the most critical requirements for Big Data systems. Performance and real-time operation are not supported by any of the patterns detailed in this work. This is due to the lack of evidence in the body of work presented in Table 3 about how software architectures proposed for Big Data systems address performance, availability, and real-time requirements. In this perspective, it is of utmost importance to investigate which architectural solutions can be integrated with patterns discussed in this work to successfully address those requirements.

The main disadvantage related to patterns offering partial solutions, is that architects must prioritize some requirements more than others (e.g., scalability, performance, security), causing adverse effects between the 5V's characteristics of Big Data system. Therefore, it is identified that the union of more than one of the patterns could add the benefits of each and then support the design of Big Data software architectures with more quality. However, such a union must be investigated carefully, in order to address the new requirements and guarantee that the crucial requirements are still considered. Moreover, the implementation of each pattern could be adapted in order to support different characteristics.

5 CONCLUSION

The Big Data context is promising due to the fact that data complexity is increasing, and therefore tends to bring more challenges to software architectures. In this paper, we investigated the applicability of architectural patterns, namely, layered, pipes and filters, and shared repository to the Big Data context using the ATAM evaluation method. We observed that none of these patterns can successfully meet the expected requirements for Big Data since each pattern was conceived to solve a specific problem. Using a specific

pattern to achieve some requirements, cause trade-offs between others desired requirements for Big Data systems. In this context, the integration of patterns to consolidate a software architecture for those systems must be carefully investigated. It is required to resort to the preliminary identification of trade-offs to define alternatives to mitigate negative impacts imposed by architectural patterns selected to conform to the software architecture of Big Data systems. This work can be used as a basis for such a trade-off analysis.

As future work, we intend to design Big Data reference architecture using the combination of these established patterns, and if necessary, propose a new one, aiming at supporting the design and documentation of software architectures for Big Data systems in healthcare domain.

REFERENCES

Al-Jaroodi, J. and Mohamed, N. (2016). Characteristics and requirements of big data analytics applications. In *2nd IEEE International Conference on Collaboration and Internet Computing, CIC 2016, Pittsburgh, PA, USA, November 1-3, 2016*, pages 426–432.

Ali Babar, M., Zhu, L., and Jeffery, D. R. (2004). A framework for classifying and comparing software architecture evaluation methods. *2004 Australian Software Engineering Conference. Proceedings.*, pages 309–318.

Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition.

Buschmann, F., Kevlin, H., and Schmidt, D. C. (2007). *Pattern-oriented software architecture, 4th Edition*. Wiley series in software design patterns. Wiley.

Cecchinel, C., Jimenez, M., Mosser, S., and Riveill, M. (2014). An architecture to support the collection of big data in the internet of things. In *2014 IEEE World Congress on Services, SERVICES 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 442–449.

Chen, H., Kazman, R., Haziyev, S., and Hrytsay, O. (2015). Big data system development: An embedded case study with a global outsourcing firm. In *1st IEEE/ACM International Workshop on Big Data Software Engineering, BIGDSE 2015, Florence, Italy, May 23, 2015*, pages 44–50.

Demchenko, Y., Turkmen, F., de Laat, C., Blanchet, C., and Loomis, C. (2016). Cloud based big data infrastructure: Architectural components and automated provisioning. In *International Conference on High Performance Computing & Simulation, HPCS 2016, Innsbruck, Austria, July 18-22, 2016*, pages 628–636.

Garises, V. and Quenum, J. G. (2018). The road towards big data infrastructure in the health care sector: The case of namibia. In *Proceedings of the* 19^{th} *IEEE Mediterranean Electronical Conference, IEEE MELECON'18*, pages 98–103, Marrakech, Morocco. IEEE.

Garlan, D. and Shaw, M. (1994). An introduction to software architecture. Technical report, Software Engineering Institute, Pittsburgh, PA, USA.

Geerdink, B. (2015). A reference architecture for big data solutions - introducing a model to perform predictive analytics using big data technology. *IJBDI*, 2(4):236–249.

Gorton, I. and Klein, J. (2015). Distribution, data, deployment: Software architecture convergence in big data systems. *IEEE Software*, 32:78–85.

ISO/IEC (2010). Iso/iec 25010 system and software quality models. Technical report, ISO.

Kazman, R., Klein, M. H., and Clements, P. C. (2000). Atam: Method for architecture evaluation. Technical report, Software Engineering Institute, Pittsburgh, PA, USA.

Santos, M. Y., e Sá, J. O., Costa, C., Galvão, J., Andrade, C., Martinho, B., Vale Lima, F., and Eduarda, C. (2017). A big data analytics architecture for industry 4.0. In *Recent Advances in Information Systems and Technologies - Volume 2 [WorldCIST'17, Porto Santo Island, Madeira, Portugal, April 11-13, 2017].*, pages 175–184.

W. Maier, M., Emery, D., and Hilliard, R. (2000). Recommended Practice for Architectural Description of Software-Intensive Systems. Technical report, IEEE.

Wang, Y., Kung, L., Wang, W. Y. C., and Cegielski, C. G. (2018). An integrated big data analytics-enabled transformation model: Application to health care. *Information & Management*, 55(1):64–79.