

Evaluation of a Data Messaging System Solution

Case: Evaluation of Apache Kafka™ at
Accanto Systems

LAHTI UNIVERSITY OF APPLIED
SCIENCES
Degree programme in Business
Information Technology
Bachelor's Thesis
Spring 2017
Huong Nguyen

Lahti University of Applied Sciences
Degree Programme in Business Information Technology

NGUYEN, HUONG:

Evaluation of a Data Messaging
System Solution
Case: Evaluation of Apache Kafka™
at Accanto Systems

Bachelor's Thesis in Business
Information Technology

38 pages, 10 pages of appendices

Spring 2017

ABSTRACT

This study aims to explore the process of adapting a new Data Messaging System Solution, i.e Apache Kafka™ (Kafka), and to evaluate whether it is suitable for the needs at Accanto Systems.

The research follows the framework for Design Science research methodology. Evaluation of the artefact involves the use of a software quality model.

The results of the study confirm that Kafka is satisfactory as a Data Messaging System solution. The results may also serve as an implementation guideline for the company to use in future encounters with the topic of data messaging.

Keyword: Data Messaging System, software quality model, Apache Kafka™, cluster, artefact.

TABLE OF CONTENTS

1	INTRODUCTION	1
2	RESEARCH DESIGN	2
2.1	Research Questions and Objectives	2
2.2	Research Methodology	2
2.3	Research Process	3
3	PRACTICAL BACKGROUND	5
3.1	Data Messaging in Customer Experience Management	5
3.1.1	About Accanto Systems Products	5
3.1.2	Challenges in Adaptive Data Collection	7
3.2	Quality Requirements of a Data Messaging System Solution	9
4	THEORETICAL BACKGROUND	11
4.1	Enterprise Integration Patterns	11
4.2	Data Messaging Systems	12
4.2.1	Distributed Data Processing	12
4.2.2	Basic Concepts of Data Messaging Systems	13
4.2.3	Choosing a Messaging System	14
4.3	About Apache Kafka	15
4.3.1	Basic Features	15
4.3.2	Topics and Logs	16
4.3.3	Distribution Management	17
5	ARTEFACT DESIGN AND IMPLEMENTATION	19
5.1	Artefact Design – Apache Kafka Adaptation	19
5.2	Quality Requirements	21
5.2.1	Characteristic 1: Functional Suitability	22
5.2.2	Characteristic 2: Performance Efficiency	22
5.2.3	Characteristic 3: Compatibility	23
5.2.4	Characteristic 4: Usability	24
5.2.5	Characteristic 5: Reliability	24
5.2.6	Characteristic 6: Security	25
5.2.7	Characteristic 7: Maintainability	25
5.2.8	Characteristic 8: Portability	26
5.3	Implementation Priorities	26

5.4	Adaptation Process	26
6	ARTEFACT EVALUATION	28
6.1	Software Testing Methods	28
6.2	Test Objectives and Scope	28
6.3	Test Features	29
6.3.1	Testing Requirement Analysis	29
6.3.2	Test Coverage	32
6.4	Test Implementation	32
6.5	Test Results	33
6.5.1	Functional Suitability	34
6.5.2	Performance Efficiency	34
6.5.3	Compatibility	35
6.5.4	Usability	35
6.5.5	Reliability	35
6.5.6	Security	36
6.5.7	Mainainability	36
6.5.8	Portability	36
7	CONCLUSIONS	37
7.1	Summary of the Study	37
7.2	Reliability and Validity	37
7.3	Further Research Questions	38
	LIST OF REFERENCES	39
	APPENDIX	41

LIST OF ABBREVIATIONS

CEM	Customer Experience Management
DMS	Data Messaging Systems
FTP	File Transfer Protocol
JMS	Java Messaging Service
JSON	JavaScript Object Notation
Kafka	Apach Kafka™
Kettle	Pentaho Data Integration
SQM	StratOSS Quality Management
XML	eXtensible Markup Language

LIST OF FIGURES

Figure 1 Design Science Research Framework (Hevner et al. 2004, 80)...	3
Figure 2 Design Science Research Methodology Process Model (Peppers et al. 2007, 14).....	4
Figure 3 StratOSS use cases (Accanto Systems Oy 2016).....	6
Figure 4 Data Collection in SQM (Accanto Systems Oy 2016).....	7
Figure 5 Data Collection in Orchestrator (Accanto Systems Oy 2016)	8
Figure 6 The ISO/IEC FCD 25010 product quality standard (ISO/IEC 25010 2011)	10
Figure 7 Quality Model Building Process (Franch and Carvallo 2003, 36)	10
Figure 8 Kafka APIs (Apache Software Foundation 2017)	16
Figure 9 Producer-Consumer Log (Apache Software Foundation 2017)..	17
Figure 10 Anatomy of a Kafka topic (Apache Software Foundation 2017)	17
Figure 11 Kafka architecture with Zookeeper (Mouzakitits 2016)	18
Figure 12 Current SQM Architecture (Accanto Systems Oy 2016).....	19
Figure 13 Current SQM Architecture – Data Collection (Accanto Systems Oy 2016).....	20
Figure 14 Evaluated SQM data adaptation with Kafka (Accanto Systems Oy 2016).....	20
Figure 15 Requirement features for testing	31
Figure 16 Test hierarchy.....	33

LIST OF TABLES

Table 1 Requirements - Functional suitability	22
Table 2 Requirements - Performance efficiency.....	23
Table 3 Requirements – Compatibility	23
Table 4 Requirements - Usability.....	24
Table 5 Requirements - Reliability	24
Table 6 Requirements – Security	25
Table 7 Requirements - Maintainability	25
Table 8 Requirements - Portability	26
Table 9 Testing documentation terms	29
Table 10 Test feature importance levels.....	30
Table 11 Test results - Functional suitability.....	34
Table 12 Test results - Performance efficiency	34
Table 13 Test results - Compatibility	35
Table 14 Test results - Usability	35
Table 15 Test results - Reliability.....	35
Table 16 Test results - Security	36
Table 17 Test results - Maintainability	36
Table 18 Test results - Portability	36

1 INTRODUCTION

Accanto Systems is a company based in Lahti, Finland, specialized in developing Customer Experience Management solutions that enable Telecom Service Providers worldwide to prioritize actions that deliver the optimal business value.

The demand for extensive customer insight from businesses in this era of Big Data calls for Accanto Systems to be constantly striving to improve operations of real-time data processing. The company therefore puts a strong emphasis on the integration of a data messaging solution that can handle increasingly diverse data sources.

Hohpe and Woolf (2004, 57) define Data Messaging System (DMS) or Enterprise Messaging System as a set of agreements utilized by enterprises to facilitate the communication of information between different computer systems and applications. The implementation of a functional, well-designed, secure DMS that fits well with the rest of the system is essential to the success of the whole software product.

This paper inspects the need for adapting a DMS at Accanto Systems and attempts to find a solution for that need. The solution can be in the form of installing an existing DMS - Apache Kafka™ (Apache Kafka or Kafka) - into the company current system and evaluate its suitability. After evaluating the solution, as a result, the thesis provides the company with an artefact design and an implementation guideline.

The study is expected to contribute to a direct improvement of the DMS adaptation process, and pave way for more research into data messaging in the context of real-life software development in the future.

2 RESEARCH DESIGN

2.1 Research Questions and Objectives

This study sets out to solve a problem the company Accanto Systems is having: the need for a Data Messaging System solution in their Customer Experience Management products. The study considers Apache Kafka™ (Apache Kafka or Kafka) as a potential solution and proceeds to evaluate its adaptation.

The research questions of this study therefore will be specified as follows:

- Is Apache Kafka a suitable solution for Data Messaging System in existing Customer Experience Management products at Accanto Systems?
- How can the success of the Apache Kafka adaptation be evaluated?

2.2 Research Methodology

This study employs design science as the research framework. As opposed to description-driven research where explanations of the phenomenon are prioritized, design science focuses on delivering solutions to the research question via building artefacts to solve the problems, implementing the treatment of the artefacts, and evaluating the results to refine the artefact construction. The process of assessment and refinement is executed continuously until the design artefact can be implemented in the business environment and provide improvements to existing theories, as illustrated in figure 1.

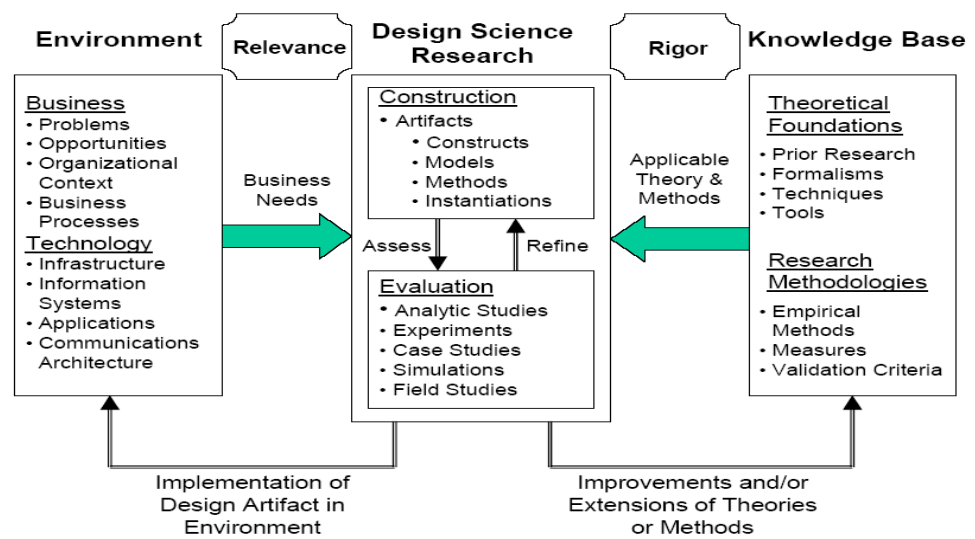


Figure 1 Design Science Research Framework (Hevner et al. 2004, 80)

This research methodology is especially popular in the field of information systems in which the pragmatic nature of the approach helps navigate the researchers towards working solutions (Hevner et al. 2004, 76) This is especially fitting given the nature of this study, in which the author aims to both help implement the artefact to improve the case company's operations and improve their personal knowledge base when it comes to applying theory to real-life problem solving.

2.3 Research Process

The study process iteration is performed through constant communication and supervision from Accanto Systems to ensure the artefact is of relevance to the company's requirements.

The author refers to a commonly accepted model provided by Peffers et al. (2007, 14) as illustrated in figure 2 below, for implementing research procedures and structuring this research paper.

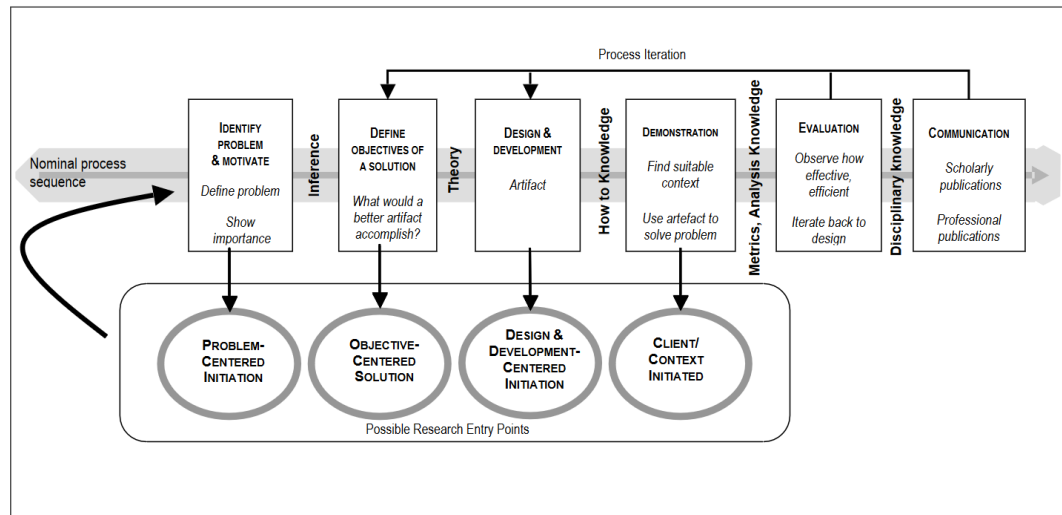


Figure 2 Design Science Research Methodology Process Model (Peffers et al. 2007, 14)

Chapter 3 of this paper discusses the motivations of the study, covering necessary practical information about the company's needs for a Data Messaging System, as well as the process of defining requirements for the finished artefact.

Chapter 4 provides theoretical background on Enterprise Integration in general and on Data Messaging Systems in particular. The chapter also introduces the characteristics of Apache Kafka and why it is proposed as the artefact to be tested.

Chapter 5 describes the process of design and implementation of Apache Kafka as a Data Messaging System solution into the current system of Accanto Systems products. Chapter 6 evaluates the artefact described in chapter 5 through methods of software testing. Finally, chapter 7 concludes the findings and discusses the reliability of the study and potential for further research questions.

3 PRACTICAL BACKGROUND

It is essential to have a basic understanding of the problem before designing a solution for the said problem. This chapter explains the real-life relevance of the research question and proposes methods to evaluate the study effectively.

3.1 Data Messaging in Customer Experience Management

This subchapter aims at providing a background of Customer Experience Management (CEM) solutions developed by Accanto Systems and exploring the need for a Data Messaging System of the company.

Gartner IT Glossary (2017) defines Customer Experience as a collective of the customer's perceptions of the brand based on interactions with that brand; and Customer Experience Management is a practice performed by the owner of the brand to understand and adapt to customer interactions and improve their customer experience. Effective CEM leads to sound business decisions that deliver personalized, satisfactory experiences to the business' customers and thus, to increasing in customer loyalty and good brand image. Gaining insight about customers allows businesses to have a great advantage over their competitors, but it requires collecting and analysing a large amount of customer feedback data from very diverse sources. (Hayes 2011.)

3.1.1 About Accanto Systems Products

StratOSS is an ecosystem developed by Accanto Systems that facilitates many aspects of customer experience management, especially network analytics, as illustrated in figure 3.

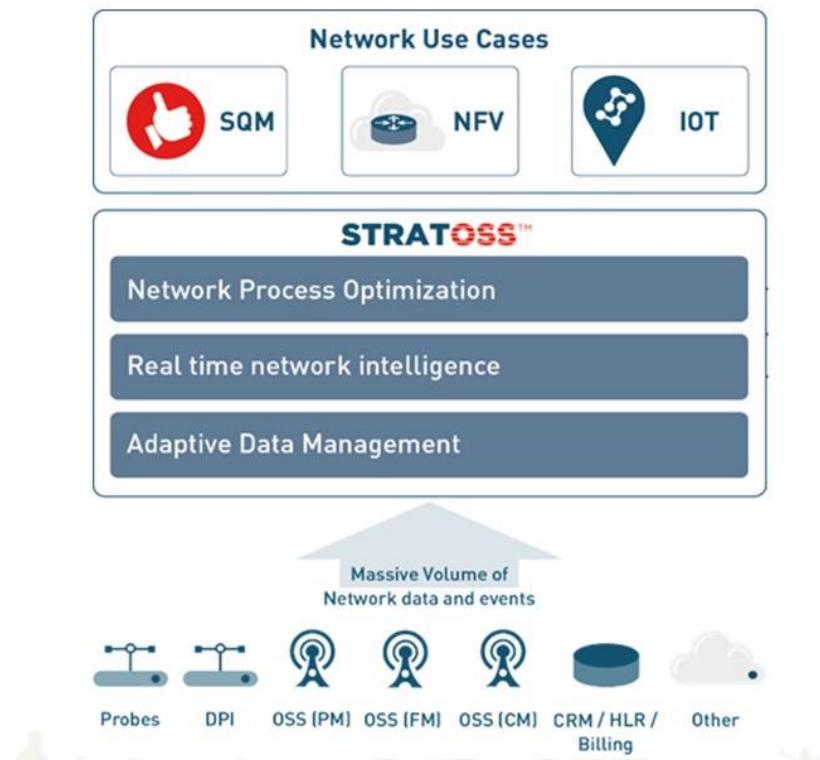


Figure 3 StratOSS use cases (Accanto Systems Oy 2016)

StratOSS serves as an intermediary between service-centric units such as mobile network customer care and network-focused units such as network operation engineers. Quality of service for end users is collected from devices and data centers, providing real-time insight on service impact so action to improve customer experience can be prompt and meaningful.

Notable spin-off products include StratOSS Quality Management (SQM) and StratOSS Network Functions Virtualization Orchestrator (Orchestrator), whose system architectures are also examined to clarify the research question. SQM examines service quality management use cases through the configuration of data sources, service models and reports. Orchestrator is an upcoming product focusing on intelligent orchestration services for network optimisation. (Accanto Systems Oy 2016.)

3.1.2 Challenges in Adaptive Data Collection

StratOSS Quality Management (SQM) is a product aimed at optimal processing of network events to give corporate customers insight into the distribution and performance of their network traffic. SQM customers are companies collecting their customer feedback from mobile network connections. Therefore, data sources for the system have been following widely-adapted protocol of the mobile industry, which means the format is quite static and can be manually configured before adapting into other components of SQM. Data transfer with manual configuration, as depicted in figure 4, can be employed to collect data packages and there has been no need of other extensive data messaging solutions. (Accanto Systems Oy 2016.)

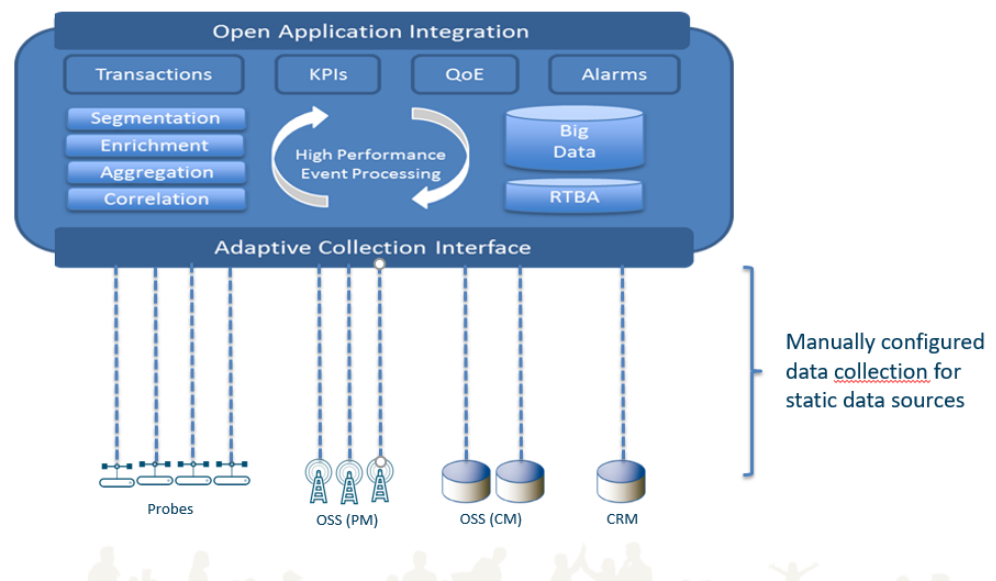


Figure 4 Data Collection in SQM (Accanto Systems Oy 2016)

However, during the design phase of StratOSS Network Functions Virtualization Orchestrator (Orchestrator), a new product of the company, it was discovered that this form of data collection may be unsuitable. Orchestrator's main function is to monitor the health of networks to scale traffic or heal components accordingly. This difference in product goal compared to SQM leads to a completely different approach of data

integration. Data sources collected for Orchestrator can come from anything from a video-on-demand server to a consumer television digibox, as can be seen in figure 5. As components should be initiated at any moment, without a concrete limit or format, manual configuration of data is impossible. (Accanto Systems Oy 2016.)

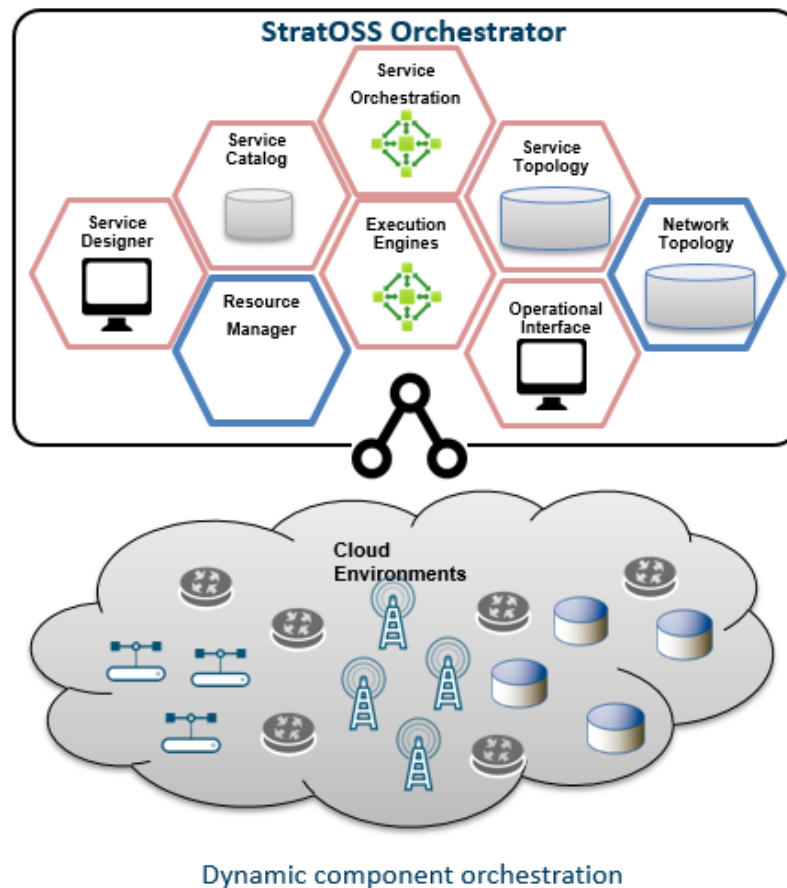


Figure 5 Data Collection in Orchestrator (Accanto Systems Oy 2016)

This calls for a dynamic, high-availability, high-capacity, scalable DMS to collect data to pre-configured topics before processing in the Orchestrator system. Apache Kafka fits into these requirements and therefore chosen to be studied as a potential messaging system.

3.2 Quality Requirements of a Data Messaging System Solution

This subchapter aims to achieve a better understanding of the desired solution and define evaluation criteria for this study. This leads to a better design and implementation of the artefact and a clearer approach when it comes to testing the artefact.

To know what should be expected of an unimplemented artefact may be disorienting, especially in the context of an unfamiliar software product ecosystem, considering every viewpoint has a different understanding of a “satisfactory solution.” In this case, the author has decided to use quality models to help define the desired objectives or characteristics of a satisfactory solution to the research problem. Quality models are a set of characteristics and sub-characteristics, as well as the relationships between them that offer researchers a clearer look at what constitutes the basis of requirements and for evaluating quality of the solution (Singh 2007, 438).

After consulting with Accanto Systems, the author concluded that following a common framework for software quality evaluation guarantees reliability of research and reusability for the company in the future. This study therefore will utilize a software quality standard to build a quality model, from which requirements of the adaptation are constructed.

The evaluation approach relies on the International Organization for Standardization and International Electrotechnical Commission 25010 Product Quality standard, which was selected for its generic nature, its adaptability, and its widespread use in creating quality models tailored to a wide variety of software domains.

The product quality model categorizes product quality properties into eight characteristics (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability). Each characteristic is composed of a set of related subcharacteristics, as depicted in figure 6. (ISO/IEC 25010 2011.)

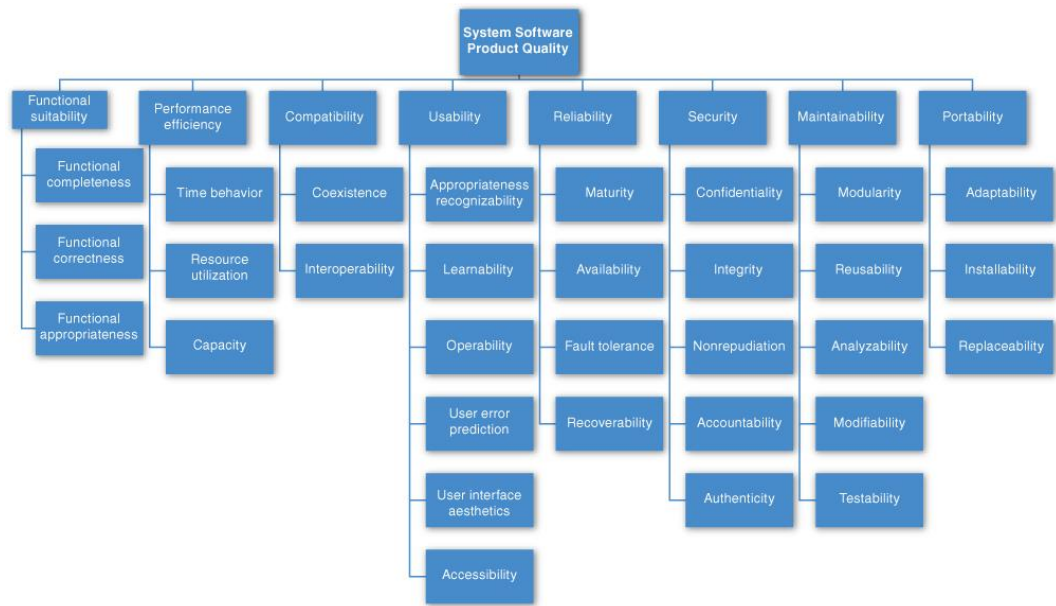


Figure 6 The ISO/IEC FCD 25010 product quality standard (ISO/IEC 25010 2011)

The research process follows the guideline proposed by Franch and Carvallo (2003, 36) for building a quality model for software package selection as shown in figure 7 below. The fine-tuning process of this quality model is closely assisted and supervised by Accanto Systems in every step to make sure the author chooses only the attributes most fitting for the scope of this study and the general company’s needs.

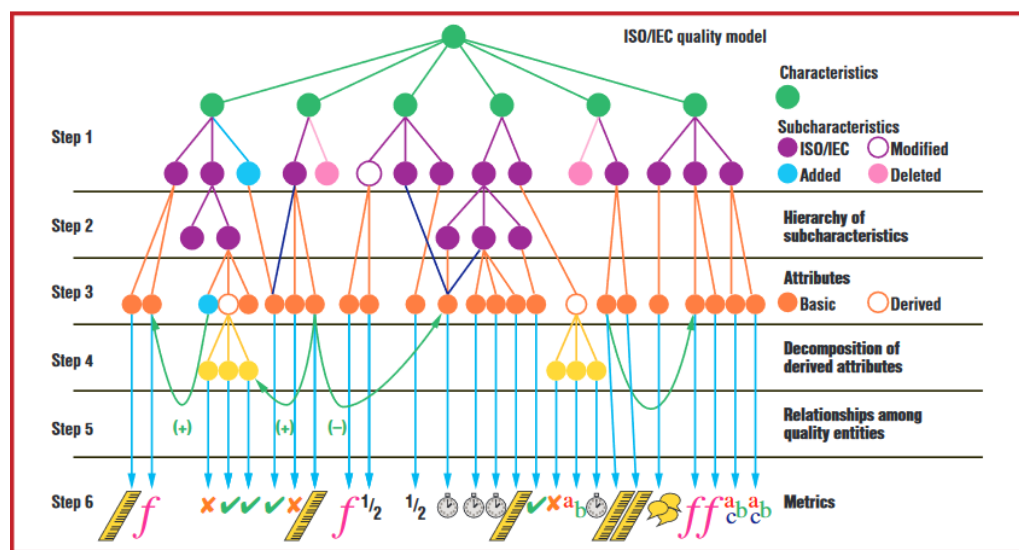


Figure 7 Quality Model Building Process (Franch and Carvallo 2003, 36)

4 THEORETICAL BACKGROUND

This chapter introduces the various concepts related to data integration and data messaging to provide better insight into the ultimate objectives of this study.

4.1 Enterprise Integration Patterns

An ongoing challenge for many software companies is a need for better integration of applications, as a functional software product is usually made by multiple different systems and components. Hohpe and Woolf (2004, 1) define this challenge as follows:

Enterprise integration is the task of making separate applications work together to produce a unified set of functionalities.

According to Hohpe and Woolf (2004, 37), a successful integration in this case should fulfill several criteria:

- Minimal inter-dependencies between integrated applications
- Simple changes for existing systems
- Understanding of technologies needed for integration
- Unified, flexible, and extensible agreement on data format
- Reduced latency of data exchange
- Ability for an application to invoke functionalities in another application
- Asynchronous procedures

With these criteria in mind, four different integration patterns are considered, and each style addresses some criteria better than others:

- File transfer: Shared data is transferred in a file, which is produced by some applications and consumed by others.
- Shared database: A common database is used to store data among applications.

- Remote procedure invocation: Applications remotely invoke permitted procedures to run certain functions and exchange data.
- Messaging: Applications are connected to a common messaging system, and data is exchanged via messages.

Usually enterprises would implement all four of the above-mentioned integration patterns to varying degrees depending on the demands of the company. However, this study focuses only on the Messaging pattern, and the following chapter explores the possibilities of different messaging systems.

4.2 Data Messaging Systems

This subchapter introduces the background of Data Messaging Systems in general and explains the reasons for choosing Apache Kafka as the focus of this study.

4.2.1 Distributed Data Processing

To mention data messaging systems in the context of this study means also to introduce the concept of distributed systems. Distributed system is a model where components on a network relay information and coordinate their actions through the passage of messages (Coulouris et al. 2011, 2). Applications of distributed computing include major network applications like the World Wide Web, or cluster computing projects, or telephone and cellular networks.

At the onset of the Big Data movement, distributed data processing tools were mostly designed to look at data in batches rather than as continuous streams. This is reflected at many large businesses' decision to follow the Extract – Transform – Load approach for years, which is the combination of two integration patterns mentioned in the previous subchapter - file transfer and shared database. Using the Extract – Transform – Load approach means jobs would have to be run every night to extract data

from some database, then transform said data, then store the data in another database. (Patil 2016.)

However, more recently enterprises have realized the full potential of processing data as they happen - this realization shifts the demand of software companies to a distributed messaging framework that can handle big data in real time. Hence Apache Kafka was introduced by LinkedIn engineers in 2011, aiming to provide a durable, scalable, low-latency messaging system that can handle big data in real time. (Patil 2016.)

4.2.2 Basic Concepts of Data Messaging Systems

A Data Messaging System (DMS) is a set of enterprise-wide standards allows for the asynchronous communication between different interfaces, where data sent by one system can be stored in the queue of another system until processed (Hohpe & Woolf 2004, 57). Some basic keywords are listed below for a better understanding of DMS:

- A message channel serves as a virtual pipe that connects a sender/publisher and a receiver/subscriber. Facilitation of message channels are required upon setup of a Data Messaging System.
- A message is a packet of data to be transmitted on a channel. Usually data transfer will require formatting of messages on both the sending and the receiving ends.
- Transformation of a message reconciles the difference in data format of a message from a sender to a receiver.
- Messaging endpoint is a layer of code that serves as an interface between existing applications and the messaging system, enabling the bridge of data between them.

DMS stands out as the preferred data communication method because its architecture allows changes in the formats of messages to have minimum impact on subscribers to the messages. DMS is run with the usage of structured messages (using formats such as XML or JSON), and

appropriate protocols, i.e Data Distribution Service, Message Queuing, or SOAP with web services. (Hohpe & Woolf 2004, 53.)

4.2.3 Choosing a Messaging System

Traditionally, messaging systems usually employ two models: point-to-point queuing and publish-subscribe. In a queue, only one receiver can successfully consume any given message, which means multiple subscribers may read from a source and each record goes to one of them. In publish-subscribe channel, however, the record is broadcast to all subscribers, and the subscriber only gets the message once and the copies disappear upon being consumed. (Hohpe & Woolf 2004, 103-106; Patil 2016.)

Each of these two models has different strengths and weaknesses. Patil (2016) points out that, a point-to-point channel allows scaling of data processing, which is divided up over multiple instances where subscribers do not have to coordinate with each other. Unfortunately, such point-to-point queues are not very flexible—the message can only be consumed by one subscriber. On the other hand, a publish-subscribe channel allows data broadcast to multiple processes, but has no way of scaling processing since every message goes to every subscriber.

The innovation of a product like Apache Kafka is that its model has both of these properties—a topic can scale processing and is also multi-subscriber. This makes Kafka stand out as the state-of-the-art data messaging solution.

Several solutions for a Data Messaging System have been considered before Kafka is chosen as the focus of this evaluation study. There are contemporary products such as Apache ActiveMQ, RabbitMQ and Flume that also enable data messaging between different platforms. Although there is function overlap between these systems, Kafka is still considered the go-to data streaming solution, superior to the above-mentioned messaging systems for numerous reasons:

- Adapatability. Unlike Flume, Kafka is not specifically designed for Hadoop integration, and can be used to process data across a wide variety of applications and platforms.
- High availability. A Kafka cluster can scale horizontally, and its replication mechanism allows data to be preserved even if a leader fails.
- Multi-purposeness. Kafka is designed to support both batch and real-time use cases.

(Shapira & Holoman 2014.)

4.3 About Apache Kafka

This subchapter provides a technical background of Apache Kafka as a distributed data messaging system.

4.3.1 Basic Features

Apache Kafka is a publish-subscribe messaging system that runs as a cluster on one or more server. Kafka maintains streams of messages in “topics”. Each record of message consists of a key, a value, and a timestamp; these messages can be used to store any object and get passed around in byte arrays. (Apache Software Foundation 2017.)

Kafka has four core APIs as can be seen in figure 8:

- Producer API lets an application publish a stream of messages to Kafka topics.
- Consumer API lets an application subscribe to topics and read messages produced to such topics.
- Streams API lets an application act as a stream processor, transforming input streams from some topics into output streams to other topics.
- Connector API connects Kafka topics to existing data systems and applications using connectors.

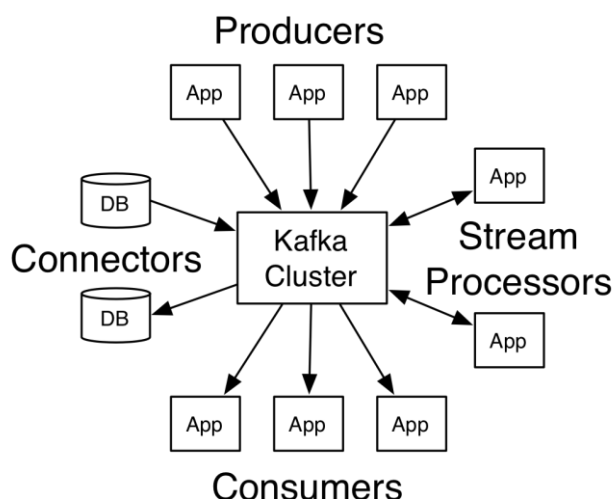


Figure 8 Kafka APIs (Apache Software Foundation 2017)

According to Apache (2017), due to the flexibility in structure, Kafka is especially useful in building real-time streaming data applications that collect data between systems or transform the streams of data. In Kafka, a stream processor takes continual streams of data from input topics, performs processing on the data, and produces continual streams of data to output topics.

4.3.2 Topics and Logs

A topic is where messages are published to. Many consumers may subscribe to one topic to process data written to it. Consumers control offset – position of records - and can consume messages in any order, whether by a reset to an older offset, or skipping ahead to the most recent one, as can be seen in figure 9. This allows a consumer flexibility to reprocess data from the past or skip ahead to the most recent record and immediately get the current data. (Apache Software Foundation 2017.)

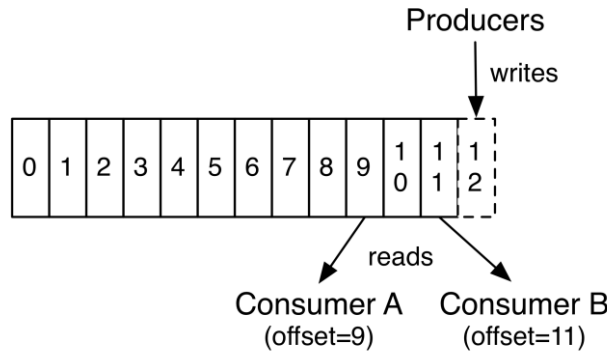


Figure 9 Producer-Consumer Log (Apache Software Foundation 2017)

According to Apache (2017), the Kafka cluster maintains a structured commit log for each topic. As can be seen in figure 10, each partition of said topic is maintained in a sequence of continually appended records. The partitions here are distributed over the servers in the Kafka cluster while being replicated across servers for fault tolerance. Kafka brokers are stateless—they do not track consumption, leaving message deletion to a configurable retention policy.

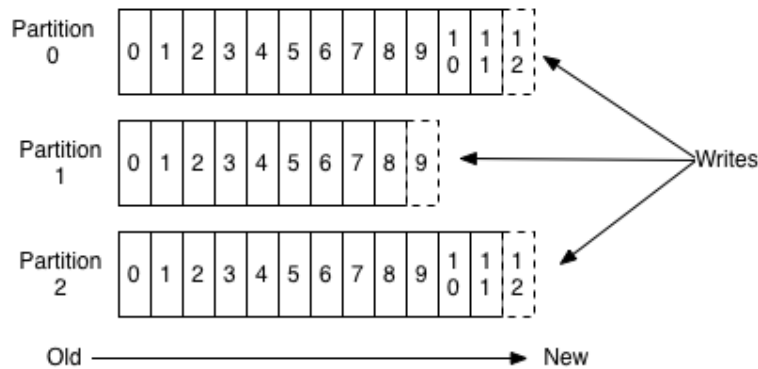


Figure 10 Anatomy of a Kafka topic (Apache Software Foundation 2017)

4.3.3 Distribution Management

The partitions in Kafka are distributed over the servers in a cluster with each server handling requests for several partitions. Each partition is also replicated, or ‘copied’, across different servers to ensure the cluster would still work as intended even if some components shut down.

In a Kafka cluster, each partition has one 'leader' server and several 'follower' servers. The 'leader' handles read and write requests for the partitions, and if it fails, a random 'follower' will become the new leader. The key in managing this system of fault tolerance is a tool called Zookeeper, which must be installed before installing Kafka. (Apache Software Foundation 2017.)

Zookeeper is a product also developed by Apache, specialized in managing configuration for distributed synchronization. It serves as the glue that holds it all together, as illustrated in figure 11, and it is responsible for the following:

- electing a controller (Kafka broker that manages partition leaders)
- recording cluster membership
- topic configuration
- ACLs (maintaining authentication between brokers)

(Mouzakitits 2016.)

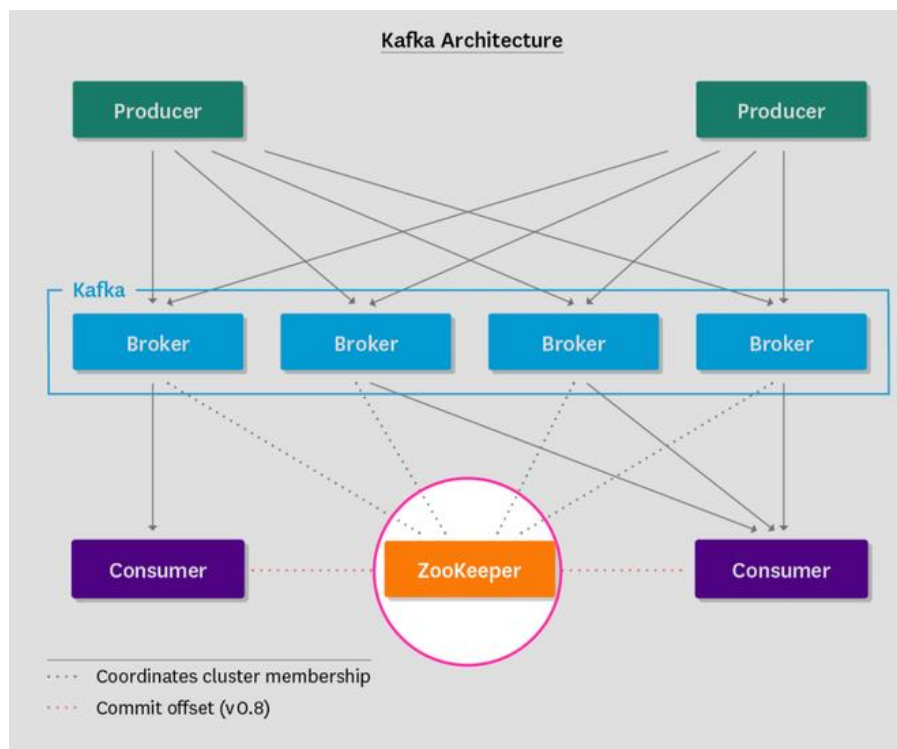


Figure 11 Kafka architecture with Zookeeper (Mouzakitits 2016)

5 ARTEFACT DESIGN AND IMPLEMENTATION

5.1 Artefact Design – Apache Kafka Adaptation

For the scope of this thesis, the author would confine the Apache Kafka artefact implementation to the case of data collection in StratOSS Quality Management (SQM).

The goal of this artefact is to adapt Apache Kafka as a data messaging system in SQM current architecture, particularly in the data collection process.

Current StratOSS Architecture

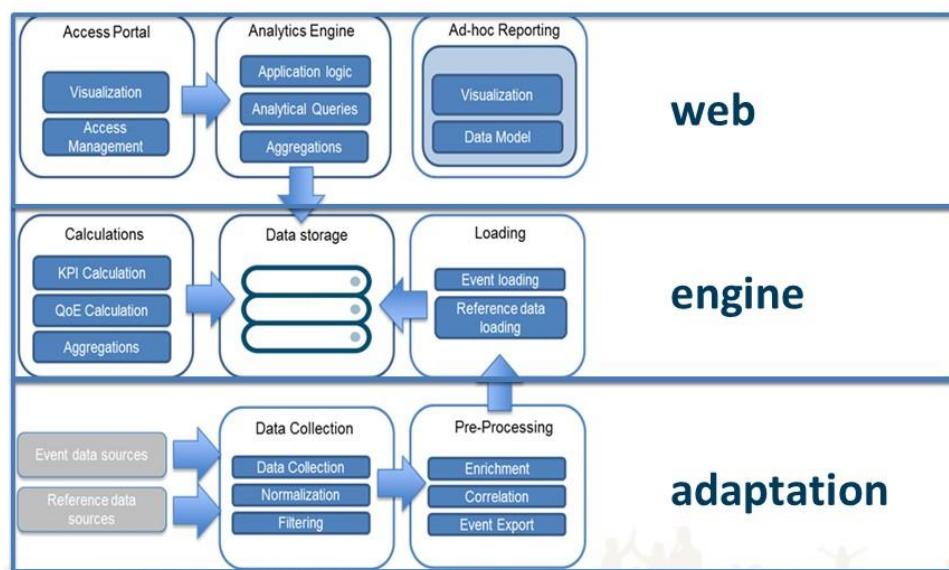


Figure 12 Current SQM Architecture (Accanto Systems Oy 2016)

Figure 12 above describes the data flows in current SQM architecture. Data are structured as 3 layers: adaptation from different data sources, transformation in engine, and retrievable for customer insight from the web portal. For this study, we take a closer look at the adaptation process of data sources, especially how the “Data Collection” step has been handled.

As can be seen from figure 13 below, data sources such as network events are exported via FTP as raw files in a repository, while developers

use JMS to keep track and send those files to Kettle adapter. Kettle then performs initial transformation of the raw files into those of compatible format with the rest of the system. Files are then renamed and then loaded to the engine of the application.

Here which set of data uses which configuration settings before going into the adapter is the process that is manually handled by going through a collection of pre-defined data size and format. This works fine for limited data sources, but may be time-consuming with more dynamic sources.

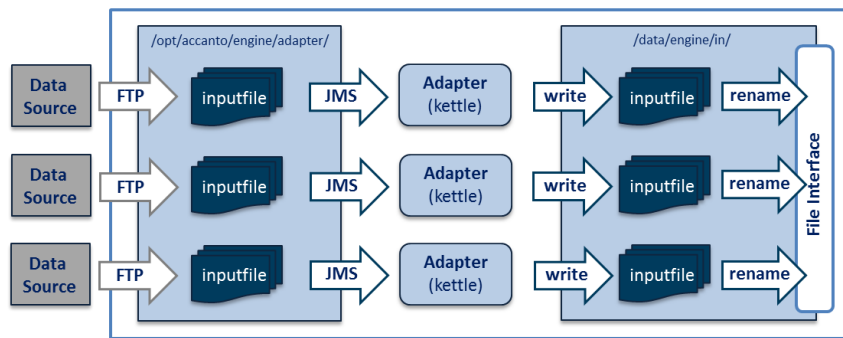


Figure 13 Current SQM Architecture – Data Collection (Accanto Systems Oy 2016)

The study proposes to use Kafka as the messaging channel for the “Data Collection” step, as seen in figure 14 below.

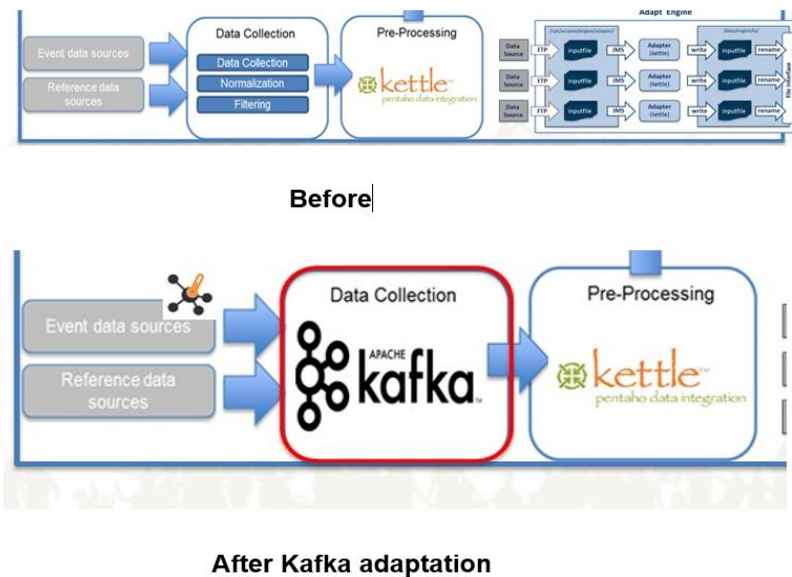


Figure 14 Evaluated SQM data adaptation with Kafka (Accanto Systems Oy 2016)

Kafka can hold a continuous stream of data directly from the event sources to the adapter interface. In contrast to the current data collection process, with Kafka data will be collected to different 'topics' according to specified configuration, and each topic will go to the desired path for the adapter to pick up data. No manual configuration, and no storage and maintenance of files will be necessary for this stage.

5.2 Quality Requirements

This subchapter explains the requirements for a successful adaptation planned in the previous subchapter. As outlined in subchapter 3.2, the study bases its requirements of the artefact on software quality requirements following standard ISO/IEC 25010:2011.

The company first states the general desired areas of interest, for example "The system should be able to run on a cluster" or "The data messaging system should not interfere with other components of the system." The thesis author studies both the nature of available data messaging systems and the company's current product architecture; and divides and sorts the company requests into suitable characteristics and subcharacteristics. The author also proposes several qualities that she discovers to be contributing towards solving the problem. The company and the author then go through the quality model again, and proceed to break down the attributes and rate them due to their importance to the project.

The following subchapters describe requirement attributes for the DMS solution, categorized under characteristics. Each subchapter contains a brief definition of the main characteristics and subcharacteristics as defined by the International Organization for Standardization and their subsequent attributes specified by the thesis author and the company.

5.2.1 Characteristic 1: Functional Suitability

Functional suitability refers to the degree to which the artefact provides functions that meet stated and implied needs when used under specified circumstances.

Attributes belonging to subcharacteristic functional completeness rates the degree to which the functions of the artefact cover all the objectives of the solution.

Functional correctness considers whether the artefact produces the correct results with the needed degree of precision.

Functional appropriateness considers the remaining tasks to be done to ensure the accomplishment of the objectives.

Sub-characteristics	Attributes	Priority
Functional completeness	Installed on a single machine: Windows and Linux	Very high
	Installed Kafka on 3-server Zookeeper cluster	Very high
Functional correctness	nProbe Cento flows successfully publishes to Kafka topic	Very high
	Spoon connector consumer successfully reads probe data from Kafka topic	Very high
Functional appropriateness	Data is in a suitable format for Spoon transformations	High
	Probe data modified by Kettle is successfully published to a different Kafka topic for loading	Medium

Table 1 Requirements - Functional suitability

5.2.2 Characteristic 2: Performance Efficiency

Performance efficiency concerns the performance level relative to the available software and hardware resources.

Resource utilization refers to the amounts and types of resources the artefact uses when performing its functions that meet requirements.

Sub-characteristics	Attributes	Priority
Resource utilization	Appropriate disc usage	High
	Appropriate CPU usage	High

Table 2 Requirements - Performance efficiency

5.2.3 Characteristic 3: Compatibility

Compatibility is the degree to which the artefact can perform with other products while sharing the same hardware or software environments.

Co-existence refers to whether or not the artefact can share resources with other components of the system without any damage to the internal workings of the artefact.

Interoperability is the ability for the artefact to exchange information with other systems or products, and use the exchanged information.

Sub-characteristics	Attributes	Priority
Co-existence	Supported: Self-managed load balancing	Low
Interoperability	Supported: DB Connector, File Connector, Data Aggregators, probes and metrics input	Low

Table 3 Requirements – Compatibility

5.2.4 Characteristic 4: Usability

Usability is the measurement of whether or not the artefact can be used by the end users in a specified context with efficiency and satisfaction.

Attributes	Priority	Note
Installation/configuration guide and recommended setup	High	Will be provided by the thesis author

Table 4 Requirements - Usability

5.2.5 Characteristic 5: Reliability

Reliability is the degree to which the artefact can function under unconventional conditions, for a specified period of time.

Maturity refers to how the artefact maintain the state of reliability by managing the frequency of failure.

Availability is the degree to which the artefact is operational when required for use, such as when the product is online.

Fault tolerance considers how the artefact operates in the presence of hardware or software faults.

Sub-characteristics	Attributes	Priority
Maturity	Alerts for hardware/software faults (disc almost full, abnormal message size, Zookeeper state, no active Kafka controller etc.)	Low
Availability & Fault Tolerance	Performing when Kafka broker(s) breaks down	Medium

Table 5 Requirements - Reliability

5.2.6 Characteristic 6: Security

Security is the degree to which the artefact protects data so that users have the appropriate access to their authorization level.

Confidentiality is how data can be accessible only to those with authorization.

Integrity refers to the prevention of unauthorized access.

Accountability refers to how the actions of users can be accurately traced back to them.

Sub-characteristics	Attributes	Priority
Confidentiality	Authorization to read/write from client	Low
Integrity	Encrypted data transfer between brokers using SSL	Medium
	Support for LDAP protocol integration	Medium
	External authorization services are supported	Medium
Accountability	Available log of actions	Medium

Table 6 Requirements – Security

5.2.7 Characteristic 7: Maintainability

Maintainability describes the efficiency with which the artefact can be monitored by maintainers.

Attributes	Priority
Available metrics tracking with notifications	Low

Table 7 Requirements - Maintainability

5.2.8 Characteristic 8: Portability

Portability is how easy the artefact can be applied to a different environment.

Adaptability is how efficiently the components can be adapted for different usage environments.

Sub-characteristics	Attributes	Priority
Adaptability	Able to expand the cluster capacity when too loaded (e.g. add new broker)	Low

Table 8 Requirements - Portability

5.3 Implementation Priorities

Since the study revolves around finding a working solution, the focus is, first and foremost, on ensuring the functionality of the artefact. In this case, it means Apache Kafka is successfully installed in the existing system of SQM and works with other components of the system. As seen in table 1 in the previous subchapter, attributes related to the characteristic 'Functional suitability' are given the highest priority.

5.4 Adaptation Process

The adaptation is performed by installing Kafka in the development environment. The production environment is deemed by the company to be unnecessary for the scope of this study. The author will replicate components of the data adaptation process in SQM system and run it with Apache Kafka as the new messaging channel.

Installation has two phases:

- Installing Kafka on single machines of both Windows and Linux – based environment. After successful installation, the author will test basic functionalities of Kafka.
- Cluster installation of Kafka along with other components of the process will be performed on a cluster of 3 Linux-based servers.
 - In this phase, the evaluation of the artefact takes place based on previously defined requirements.
 - Every action and technologies used in this phase will be documented for future configuration guide.

After installation is done, the author will examine the functionality, performance, compability, usability, reliability, security, maintainability, portability of the artefact.

6 ARTEFACT EVALUATION

The artefact Apache Kafka adaptation is measured and evaluated with software testing methods based on requirements in chapter 5.

6.1 Software Testing Methods

Software testing is the process of evaluating a software component against pre-defined requirements by detecting differences between given input and expected output. There are two major methods of software testing: blackbox testing and whitebox testing. Blackbox testing, mostly employed in functional tests, focuses on the output of the tests against the system input and execution, ignoring internal mechanism of the system. Whitebox testing, on the other hand, takes into consideration the system internal mechanism, aiming to make sure that the product behaves the way it is supposed to. Blackbox testing is usually used at the ending phase of development and whitebox testing is used at the start. Elements of both methods are usually combined in software testing. (Myers 2004, 9-14.)

In this case, the object of the tests is not a finished software product, but the adaptation of a component into an existing product. As mentioned in chapter 3, the study also follows a software quality model in evaluating this adaptation, which helps prioritize the testing process. Therefore, both whitebox and blackbox testing will be implemented – blackbox testing is used to validate the artefact functional installation, while whitebox testing is used to verify the artefact follows the system's non-functional requirements.

6.2 Test Objectives and Scope

The purpose of this integration test is to verify that all requirements of a successful Kafka adaptation into StratOSS architecture are met.

With a view to maintaining integrity of the development research, the author of this study decides to follow Accanto Systems' testing guidelines,

and subsequently models this test plan after the company's test plan template.

The testing documentations will include the terms listed in the following table.

Term	Description
Test set	Test set consists of test cases. Test set can include test cases for one feature (e.g. Installation on Windows) or bunch of similar requirements to be verified (e.g. authorization reports).
Test case	Test case consist of test steps to verify single feature or sales item (e.g. start Zookeeper instance). After a test case is executed, its status is shown as passed or failed.
Test step	Test steps describes needed actions to verify deliverables.
(Requirement) Package	Requirement package consists of requirement features. A package describes a characteristic or subcharacteristic of the desired outcome (e.g Functional completeness).
(Requirement) Feature	Requirement feature describes an attribute of the desired outcome (e.g Successful installation).

Table 9 Testing documentation terms

6.3 Test Features

6.3.1 Testing Requirement Analysis

Testing features are developed according to requirements previously listed in subchapter 5.2. The importance levels of the features are categorized according to the importance of the requirements and how critically it will affect the adaptation process.

Importance	Description
1 – Critical	Feature must be 100% covered by test cases. The adaptation is considered successful only when all test cases linked to 'Critical' feature pass. Feature directly and significantly affects meeting the goals of the adaptation or the existing system's main functionalities.
2 – High	Feature must be 100% covered by test cases. Feature indicates that the adaptation process is not functioning but the overall system remains operational.
3 – Medium	Feature does not need 100% coverage. Feature is not critical to the adaptation process, but should be inspected for documentation purposes. Feature may not be tested.
4 - Low	Feature is not important to the integration process. Feature may not be tested.

Table 10 Test feature importance levels

Figure 15 below illustrates the hierarchy, along with the importance level, of the requirements:



































Name	Importance
☐  Kafka Integration	
☐  Functional suitability	1 - Critical
☐  Functional completeness	1 - Critical
 <u>Installation on Windows</u>	1 - Critical
 <u>Installation on Linux</u>	1 - Critical
 <u>Cluster installation</u>	1 - Critical
☐  Functional correctness	1 - Critical
 <u>nProbe successfully publish flows to Kafka topic</u>	1 - Critical
 <u>Spoon connector consumer successfully reads probe ...</u>	1 - Critical
☐  Functional appropriateness	3 - Medium
 <u>Data is in a suitable format for Spoon transformat...</u>	2 - High
 <u>Probe data modified by Kettle is succesfully publi...</u>	3 - Medium
☐  Performance efficiency	2 - High
☐  Resource utilization	2 - High
 <u>Appropriate CPU usage</u>	2 - High
 <u>Appropriate disc usage</u>	2 - High
☐  Compatibility	4 - Low
 <u>Co-existence</u>	4 - Low
 <u>Interoperability</u>	4 - Low
 <u>Usability</u>	2 - High
☐  Reliability	3 - Medium
 <u>Maturity: alerts for hardware and software faults</u>	4 - Low
☐  Availability and Fault Tolerance	3 - Medium
 <u>Still running when 1 broker broke down</u>	3 - Medium
☐  Security	3 - Medium
 <u>Confidentiality: authorization to read/write from ...</u>	4 - Low
☐  Integrity	3 - Medium
 <u>Support for external authorization services</u>	3 - Medium
 <u>Support for LDAP</u>	3 - Medium
 <u>SSL encrypted data transfer</u>	3 - Medium
 <u>Accountability</u>	3 - Medium
 <u>Maintainability</u>	4 - Low
☐  Portability	4 - Low
 <u>Adaptability</u>	4 - Low

Figure 15 Requirement features for testing

6.3.2 Test Coverage

As mentioned in table 3 above, all the features marked with importance level greater than or equal to “High” are required to be covered. Within the scope of this study, the thesis author along with the company agree that the following features will be excluded from the current test run:

- Functional suitability
 - o Functional appropriateness: Probe data modified by Kettle is successfully published to a different Kafka topic for loading (Priority: Medium)
- Security
 - o Accountability: Log of actions (Priority: Medium)

6.4 Test Implementation

Testing is implemented with the use of SpiraTest testing management system. Testing is planned and performed by the author of this thesis with revision by supervisors at Accanto Systems.

Several test cases may be written to cover one requirement feature, or one test case may be linked to several features. However, the scope of this test requires that each requirement feature must be linked with at least one test case.

Each test case should have at least one test step. Test step is marked as passed if the actual result in the test step is the same as the expected result. In such cases, there is no need to fill the actual result field.

Folder – e.g. Functional suitability, Functional completeness

Test Set – e.g. Installation tests

Test Case – e.g. Kafka Installation on Windows environment

Test Step - e.g. verify that Zookeeper is installed.

Figure 16 Test hierarchy

Test step is marked as failed if the actual result is different from the expected result. In such cases, actual result field must be filled to give as much supporting information as practical. When needed, new issue must be created to test management system for issue tracking purposes.

6.5 Test Results

As can be seen from the tables below, all tests included in the testing coverage scope of this study have passed, except for one belonging to feature “Support for LDAP protocol integration” linked to characteristic ‘Security’. This is a test of priority ‘Medium’, which means the passing of the test is not critical to the adaptation process, but should be inspected for documentation purposes.

All tests of priority ‘Very high’, which are required for the basic functionalities of the adaptation, have passed. A more detailed report of the testing results organized by test cases with steps can be found in the appendix.

6.5.1 Functional Suitability

Sub-characteristic package	Feature	Priority	Test result
Functional completeness	Installed on a single machine: Windows and Linux	1-Critical	Passed
	Installed Kafka on 3-server Zookeeper cluster	1-Critical	Passed
Functional correctness	nProbe Cento flows successfully publishes to Kafka topic	1-Critical	Passed
	Spoon connector consumer successfully reads probe data from Kafka topic	1-Critical	Passed
Functional appropriateness	Data is in a suitable format for Spoon transformations	2-High	Passed
	Probe data modified by Kettle is successfully published to a different Kafka topic for loading	3-Medium	Not tested

Table 11 Test results - Functional suitability

6.5.2 Performance Efficiency

Sub-characteristic package	Feature	Priority	Test result
Resource utilization	Appropriate disc usage	2-High	Passed
	Appropriate CPU usage	2-High	Passed

Table 12 Test results - Performance efficiency

6.5.3 Compatibility

Sub-characteristic package	Feature	Priority	Test result
Co-existence	Supported: Self-managed load balancing	4-Low	Passed
Interoperability	Supported: DB Connector, File Connector, Data Aggregators, probes and metrics input	4-Low	Passed

Table 13 Test results - Compatibility

6.5.4 Usability

Feature	Priority	Test result	Note
Installation/configuration guide and recommended setup	2-High	Passed	Separately provided to the company by the thesis author

Table 14 Test results - Usability

6.5.5 Reliability

Sub-characteristic package	Feature	Priority	Test result
Maturity	Alerts for hardware/software faults (disc almost full, abnormal message size, Zookeeper state, no active Kafka controller etc.)	4-Low	Passed
Availability & Fault Tolerance	Performing when n Kafka broker(s) breaks down	3-Medium	Passed

Table 15 Test results - Reliability

6.5.6 Security

Sub-characteristic package	Feature	Priority	Test result	Note
Confidentiality	Authorization to read/write from client	4-Low	Passed	
Integrity	Encrypted data transfer between brokers using SSL	3-Medium	Passed	
	Support for LDAP protocol integration	3-Medium	Failed	Not currently supported by Apache Kafka
	External authorization services are supported	3-Medium	Passed	
Accountability	Available log of actions	3-Medium	Not tested	

Table 16 Test results - Security

6.5.7 Maintainability

Feature	Priority	Test result
Available metrics tracking with notifications	4-Low	Passed

Table 17 Test results - Maintainability

6.5.8 Portability

Sub-characteristic package	Feature	Priority	Test result
Adaptability	Able to expand the cluster capacity when too loaded (e.g. add new broker)	4-Low	Passed

Table 18 Test results - Portability

7 CONCLUSIONS

7.1 Summary of the Study

This thesis aimed to develop a Data Messaging System solution for Customer Experience Management products at Accanto Systems. The objectives were evaluated using software quality model built on the framework of Quality Standard ISO/IEC 25010:2011. An initial adaptation was installed and tested against the current system of data collection in StratOSS Quality Management.

Based on the testing plan results, it can be concluded that Apache Kafka fulfilled the need for a Data Messaging System in Accanto Systems. However, it did not yet support an authentication protocol commonly used by customers of the company.

The study also confirmed that Apache Kafka as a Data Messaging System is easy to install and maintain, scalable, suitable for projects with high demands for high availability and performance.

7.2 Reliability and Validity

This study was conducted in cooperation between the author and Accanto Systems, where the company provides considerate supervision to ensure the desired efficiency of a working life development research. The research criteria had been continuously reviewed and updated in the process of Kafka adaptation so that any weaknesses revealed by the artefact would result in a thorough modification of the installation guideline. The test cases for this feasibility research had been designed with the configurations that are specifically fitting for the targetted production environment. The testing procedures also followed recognised industry standards with commonly used and reliable tools.

It is worth noting, however, that to effectively study Apache Kafka means to study an entire open-source ecosystem supporting it, which continues to

adapt to constant needs of the market. There is also the possibilities that the structure of Accanto Systems' products would also change in the future, requiring a different solution for Data Messaging System. Therefore, some aspects discussed in this study may require revisits when current solutions have better alternatives.

7.3 Further Research Questions

The limited scope of this research allows for only the initial stage of artefact implementation, which is adaptation of Apache Kafka into StratOSS Quality Management. The author would like to examine the features that have not been tested in the scope of this thesis. The author would also like to continue to explore the full capacity of Apache Kafka in production with StratOSS Network Functions Virtualization Orchestrator in the future, where Apache Kafka would be put to its highest potential with more dynamic data sources.

LIST OF REFERENCES

Printed sources

Coulouris, G., Dollimore, J., Kindberg, T. & Blair, G. 2011. Distributed Systems: Concepts and Design. 5th Edition. Boston: Addison-Wesley.

Franch, X. & Carvallo, J. P. 2003. Using Quality Models in Software Package Selection. IEEE Software, 34-41.

Hevner, A. R., March, S. T., Park, S. & Ram, S. 2004. Design Science in Information Systems Research. MIS Quarterly 28 (1), 75-106.

Hohpe, G. & Woolf, B. 2004. Enterprise Integration Patterns. Boston: Addison-Wesley.

ISO/IEC 25010, 2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva: International Organization for Standardization.

Myers, G. J., Badgett, T., Thomas, T. M. & Sandler, C. 2004. The art of software testing. 18th edition. Hoboken: John Wiley & Sons.

Peppers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. 2007. A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems 24 (3), 45-78.

Singh, I. 2007. Different Software Quality Model. International Journal on Recent and innovation trends in computing and communication 1 (5), 438-442.

Digital sources

Accanto Systems Oy. 2016. StratOSS product introduction. Unpublished confidential document.

Apache Software Foundation. 2017. Introduction. Apache Kafka [accessed 19 March 2017]. Available at:

<https://kafka.apache.org/>

Gartner, Inc. 2017. IT Glossary: Customer Experience Management. Gartner IT Glossary [accessed 10 April 2017]. Available at:

<http://www.gartner.com/it-glossary/customer-experience-management-cem>

Hayes, B. 2013. Big Data has Big Implications for Customer Experience Management. IBM Big Data Hub [accessed 10 April 2017]. Available at:

<http://www.ibmbigdatahub.com/blog/big-data-has-big-implications-customer-experience-management>

Mouzakitits, E. 2016. Monitoring Kafka performance metrics. Datadog [accessed 10 April 2017]. Available at:

<https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/>

Patil, S. 2016. Build a continuous big data messaging system with Kafka. JavaWorld [accessed 10 April 2017]. Available at:

<http://www.javaworld.com/article/3060078/big-data/big-data-messaging-with-kafka-part-1.html>

Shapira, G. & Holoman, J. 2014. Apache Kafka for Beginners. Cloudera [accessed 10 April 2017]. Available at:

<http://blog.cloudera.com/blog/2014/09/apache-kafka-for-beginners/>

APPENDIX

TEST CASE REPORT- PROJECT: KAFKA ADAPTATION

This report is generated by SpiraTest Reporting.

Test TC:2831-Installation on Linux

This test is performed on a Linux-based platform

Step	Description	Expected Result	Last Status
1	Download and un-tar the binary release	Folder created with compiled version of Kafka	Passed
2	Start Zookeeper instance	Zookeeper running on port 2181	Passed
3	Start Kafka server	Kafka listening on port 9092	Passed
4	Create a sample topic	Topic created with a single partition and only one replica	Passed
5	Send messages to topic with command line producer and consumer	Consumer outputs messages sent by producer in real time	Passed

Test TC:2832-Cluster Installation

This test is performed on a Hadoop cluster of 3 Linux servers, one run on Centos 7 and the other two are on Ubuntu 16.04

Step	Description	Expected Result	Last Status
1	Install Kafka locally on each server of the cluster	Kafka successfully performs functions on local level	Passed
2	Modify Zookeeper configuration file on each server and start Zookeeper	Details on Zookeeper configuration files match. Zookeeper establishes connection to all 3 nodes/servers	Passed
3	Modify Kafka server configuration file on each server and start Kafka	Details on server configuration files match. Kafka cluster is established in all 3 nodes	Passed
4	Install Kafka Manager	Kafka cluster and its nodes is managable with the web GUI on port 9000	Passed
5	Create a sample topic (preferably via Kafka Manager) and assign partitions and elect preferred replica.	Topic created with 4 partitions and 3 replicas	Passed
6	Send messages to topic with command-line producer and command-line consumer	Consumer from one server outputs messages sent by producer from another server in real time	Passed

Test TC:2828-Installation on Windows

This test is performed on a Windows 10 platform

Step	Description	Expected Result	Last Status
1	Download and un-tar the binary release	Folder created with compiled version of Kafka	Passed
2	Start Zookeeper instance	Zookeeper running on port 2181	Passed
3	Start Kafka server	Kafka listening on port 9092	Passed
4	Create a sample topic	Topic created with a single partition and only one replica	Passed
5	Send messages to topic with command line producer and consumer	Consumer outputs messages sent by producer in real time	Passed

Test TC:2833-nTop-nProbe installation

Step	Description	Expected Result	Last Status
1	Install nTop package	The following packages are installed: ntopng, nprobe, cento, n2disk, pfring	Passed
2	Setup a dummy interface for probe data	Success	Passed
3	Configure hugepages and start pf_ring service	Success-checked by nBox web GUI	Passed

Test TC:2834-nProbe Cento - Kafka connection

Step	Description	Expected Result	Last Status
1	Generate flows by nProbe Cento and export to Kafka topics	'Flow exporter queue' status shown in the terminal	Passed
2	Kafka successfully consumes messages from a different node	Command-line consumer displays probe data	Passed

Test TC:2835-Spoon consumer gets data from Kafka

Step	Description	Expected Result	Last Status
1	Install Spoon	Spoon installed and run successfully	Passed
2	Install Kafka Consumer plugin	Kafka Consumer step added	Passed
3	Modify the configuration for Kafka Consumer step	Port and topic are correct	Passed
4	Run the Spoon transformation	Data shown in the consumer (refer to the previous test case) is written to log	Passed

Test TC:2836-Spoon modifies data and publishes to Kafka

Step	Description	Expected Result	Sample Data	Last Status
1	Call 'Spoon consumer gets data from Kafka'			N/A
2	Spoon JSON data transformation: filter values of a field (e.g <8)	Filtered JSON objects		Passed
3	Data modified to data stream format	JSON objects become binary feed		Passed

Test TC:2837-Spoon data input format

Step	Description	Expected Result	Sample Data	Last Status
1	Call'Spoon consumer gets data from Kafka'			N/A
2	Add and configure Select Values step: change filed metadata from binary to normal	Step success		Passed
3	Add and configure JSON Data Input step	Data is written to log is recpgnized as JSON format		Passed

Test TC:2840-Disc usage

Step	Description	Expected Result	Last Status
1	Proper disc usage on node handling nprobe export	500MiB	Passed
2	Proper disc usage on node handling Spoon	2GiB	Passed

Test TC:2838-CPU usage

Step	Description	Expected Result	Last Status
1	Proper CPU usage on node handling nprobe export	<30%	Passed
2	Proper CPU usage on node handling Spoon	<8%	Passed

Test TC:2841-Load-balancing support

Step	Description	Expected Result	Last Status
1	Self-managed load balancing by Zookeeper in the form of reassigning partitions and electing replica leader	Available	Passed

Test TC:2842-Connector support

- DB Connector
- File Connector
- Data Aggregators
- Probes input
- Metrics input

Step	Description	Expected Result	Last Status
1	Connectors in the list are supported	According to Kafka documentation	Passed

Test TC:2843-User/configuration guide

Step	Description	Expected Result	Last Status
1	Installation guide	Provided by thesis author	Passed
2	Test report	Provided by thesis author	Passed

Test TC:2844-Alerts for hardware/software faults

- disc almost full
- abnormal message size
- changed Zookeeper state
- no active Kafka controller

Step	Description	Expected Result	Last Status
1	Configure alerts when the system encounter faults	Success	Passed

Test TC:2845-Still running when 1 broker broke down

Step	Description	Expected Result	Last Status
1	Force stop one broker	Kafka processes are still running. Replicas are automatically handled	Passed

Test TC:2846-Read/write authorization support

Step	Description	Expected Result	Last Status
1	Support for read/write authorization	Available	Passed

Test TC:2847-External authorization services

Step	Description	Expected Result	Last Status
1	Support for authorization services e.g TLS	Available	Passed

Test TC:2848-LDAP support

Step	Description	Expected Result	Last Status
1	Support for LDAP authentication system	Available	Failed

Test TC:2849-Data transfer encryption

Step	Description	Expected Result	Last Status
1	Data transfer encryption by SSL	Available	Passed

Test TC:2851-Metrics tracking support

	Description	Expected Result	Last Status
1	Metrics monitoring	Available	Passed

Test TC:2852-Ability to add new broker

Step	Description	Expected Result	Last Status
1	Manually add a new broker to a running Kafka cluster	Success	Passed