

# Methodology for KPI-selection and incremental cost analysis for IoT-applications in development phase

Victor Desmet

Student number: 01205346

Supervisors: Prof. dr. ir. Sofie Verbrugge, Prof. dr. ir. Didier Colle  
Counsellors: Dr. ir. Frederic Vannieuwenborg, Timo Latruwe

Master's dissertation submitted in order to obtain the academic degree of  
Master of Science in Computer Science Engineering

Academic year 2019-2020



# Methodology for KPI-selection and incremental cost analysis for IoT-applications in development phase

Victor Desmet

Student number: 01205346

Supervisors: Prof. dr. ir. Sofie Verbrugge, Prof. dr. ir. Didier Colle  
Counsellors: Dr. ir. Frederic Vannieuwenborg, Timo Latruwe

Master's dissertation submitted in order to obtain the academic degree of  
Master of Science in Computer Science Engineering

Academic year 2019-2020



# Permission for use of content

“The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In the case of any other use, the limitations of the copyright have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation.”

Victor Desmet, January 2020



# Preface

With this preface I write, with some pride, the final words of my thesis. It has been a very educational year where my perseverance was tested very thoroughly.

My student days took me a little longer than I (and perhaps also my parents) wished for, but by submitting this work the end is near. Over the years at the University of Ghent, my sleepless nights in the Overpoort evolved into sleepless nights at my desk. As this is coming to an end, I think it is save to say I got the absolute maximum out of my student days. I'm looking forward to start the working life now.

Now my thesis is finished, I would like to take the time to thank some people.

First of all I want to thank all my friends and colleagues for the beautiful last years. The many philosophical conversations and discussions at night have certainly changed my view on life.

I also want to thank my roommates and Lucy for the the last year, accompanied with the necessary japes and larks, the discussions about this thesis have certainly provided me with some insights that contributed to this end result.

Next I want to thank Professor dr. ir. Sofie Verbrugge and Professor dr. ir Didier Colle for the opportunity they gave me to write this thesis. A special thanks to professor Didier Colle for the suggestions per mail after the last

feedback session of this year. This surely helped to stress some important parts of this thesis.

A huge thank you for Willem, Nina and my mom for the many hours of proofreading this work. Without your help the thesis would be full of “it’s”.

I want to thank both my supervisors twice. dr. ir. Frederic Vannieuwenborg and Timo Latruwe. A first thank you for the help during the last year. Your critical attitude when required, motivation when needed and help when asked was very helpful. A second thank you is appropriate for the last week of this thesis year. While coming out of vacation, as I can imagine with an overloaded mailbox and many work to do, you took the time to read my work and give some last minute feedback in order to make this master dissertation as good as possible.

A last thank you is for my parents, thanks for the opportunity to let me study and the support through all this years. You learned me that with hard work, one can achieve a lot!

Victor Desmet, January 2020



# Abstract

The last recent years, internet of things (IoT) applications found their way in many different domains and use cases. Key performance indicators (KPIs) for these applications help organizations to monitor and evaluate how well the applications achieve their main value objectives. To determine and define the correct KPIs in development phase can be a hard exercise for many organizations, this work proposes a methodology to predict KPIs for IoT applications in development phase. A cost model based on a 4-layer architecture for IoT applications, lists the different costs an organization should take into account when developing an IoT application. Based on this first cost model an incremental cost model is given that shows which incremental costs should be considered by organizations when they want to add new KPIs after development phase. Based on this incremental cost model, organizations can estimate the economic impact of missing KPIs in development phase. The second part of this work focuses on the methodology an development of a KPI suggestion web tool. This tool offers the opportunity to organizations to validate their KPIs in development phase and suggests other KPIs that could be useful for an organization based on the non-functional requirements of the application they want to develop. This web tool is used first to suggest KPIs to an organization, the incremental cost analysis can be used afterwards to take the decision whether or not to implement these suggested KPIs. The last section of this work validates this methodology against the Sundo use case of developing smart sunscreen dispensers. The methodology provides the Sundo organization with 8 new KPIs they did not think about, stating that both the web tool and incremental cost analysis create an added value for developing new IoT applications.

**Keywords:** Internet of Things, IoT, Key Performance Indicators, KPI, KPI suggestion, Incremental cost Analysis



# Methodology for KPI-selection and incremental cost analysis for IoT-applications in development phase

Victor Desmet

Supervisors: prof. dr. ir. Sofie Verbrugge, prof. dr. ir. Didier Colle, dr. ir. Frederic Vannieuwenborg, Timo Latruwe

**Abstract**—The last recent years, internet of things (IoT) applications found their way in many different domains and use cases. Key performance indicators (KPIs) for these applications help organizations to monitor and evaluate how well the applications achieve their main value objectives. To determine correct KPIs can be a hard exercise for many organizations. This work proposes a methodology to predict KPIs for IoT applications in development phase. A cost model based on a 4-layer architecture for IoT applications, lists the different costs an organization should take into account when developing an IoT application. Based on this generic cost model an incremental cost model is given that shows which costs should be taken into account when a KPI must be implemented after development phase. Organizations can estimate the economic impact of missing KPIs in development phase based on this model. The second part of this work focuses on the methodology an development of a KPI suggestion web tool. This tool suggests KPIs that could be useful for an organization based on the non-functional requirements of the application they want to develop. If an organization doubts whether or not to implement a suggested KPI, they can use the incremental cost model to determine possible cost drivers for the KPI after development phase. The last section of this work validates this methodology against the Sundo use case of developing smart sunscreen dispensers. The methodology provides the Sundo organization with 8 new KPIs they did not think about, stating that both the web tool and incremental cost analysis create an added value for developing new IoT applications.

**Index Terms**—Internet of Things, IoT, Key Performance Indicators, KPI, KPI suggestion, Incremental cost Analysis

## I. INTRODUCTION

The Internet of Things (IoT) opens up a whole range of new services and applications in many domains. Examples are: control and automation of lighting and heating (Smart Home), blood pressure and heart rate monitoring enabling remote healthcare (mHealth), vibrations monitoring of a machine part in order to perform preventive maintenance (smart industry), etc. The value of these IoT services often lays in the fact that the data produced by the sensors allows to take appropriate actions.

Nowadays, data analytics are often used to get insights and discover correlations between captured data points. In order to be able to use these techniques, one must have (access to) data. Since data is a product that results from most IoT applications, it is often not yet available in a development phase. In addition, data analytics can only generate insights based on what is measured by the IoT-device, eventually supplemented with external data. Thus, aspects not measured by the IoT-device cannot be included in the analysis.

Therefore, to maximize the expected impact of the IoT application (e.g. increased operational efficiency) whilst minimizing the costs for post-deployment data analysis and potential refactoring costs (e.g. adding a sensor in an existing and already deployed solution), one needs to have a clear view on what to measure and how to monitor certain aspects of the IoT-application at an early stage of the IoT-development.

In other words, being able to define the Key Performance Indicators (KPIs), and their metrics of the IoT-application and system before actual development and deployment, affects both the impact the IoT-system can have and the effort to get insights in these KPIs. The question is how and to what extent we can incorporate this step in the IoT-development phase?

This work starts with a literature study about what KPIs for IoT applications are and how they are determined and defined nowadays. Based on this literature study, both a cost model for the development of IoT applications as an incremental cost model for adding KPIs to an IoT application are given. A methodology and implementation of a KPI suggestion web tool allows organizations to validate their KPIs and suggests new, useful KPIs the organization may not have thought of at first. A validation of the methodology is given based on a use case concerning smart sunscreen dispensers. Finally, a conclusion about the chosen methodology is formulated together with some future improvements for the suggested methodology.

## II. LITERATURE STUDY

To create an incremental cost structure and describe a methodology for KPI suggestions a literature study was done. The 4-layer architecture from the work of Niyato et al. [1] is used as starting point to list the cost components of an IoT application. This architecture is adapted to the architecture given in figure 1. Using this architecture, the costs for every layer are determined and listed. The work of De Cock [2] is used to start the development of the incremental cost models. The work of Lucero [3] gives a lot of insights in the platform layer for the cost model. Whereas Leonard [4] describes how sensor prices are falling and what the impact of this trend means for the cost of IoT applications.

Interviews with experts in the IoT-domain reinforced the need for a KPI suggestion tool. These experts were IoT responsables in big companies such as Deloitte, SAP and Daikin. Other interviews with people who deal with KPIs in their job gave some insights into how KPIs are defined. These people include staff of Imec, Rombit, Televic rail, Philips and the

retail factory. In total more than 15 interviews with people using KPIs were taken. In his book about Key performance indicators Parmenter [5] talks about how winning KPIs should be defined and proposes a six-stage implementation plan to discover and implement useful KPIs. The work of Fukuda et al. [6] describes the use of a KPI net to link KPIs together. KPILibrary [7] provides a lot of KPIs to fill this KPI net. Willaert et al. [8] defines a methodology for performance measurement that supports a process-oriented vision on an organization. This work uses the 4-layer architecture discussed

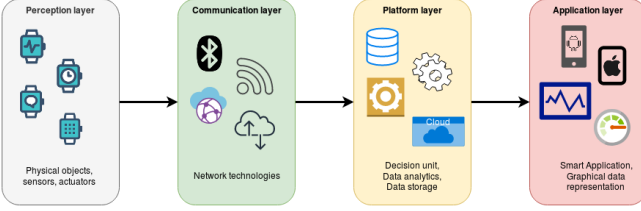


Figure 1. The 4 layer architecture used as basis for the incremental cost model

above and creates an incremental cost model that allows to identify the costs of adding KPIs after the development phase of an application. Based on the literature study done, KPIs are nowadays mostly based on gut feeling. For this, the work also defines a methodology to suggest useful KPIs for an IoT applications based on the input of non-functional requirements of this application.

### III. INCREMENTAL COST ANALYSIS

Starting from the 4-layer architecture as described in section II and given in figure 1, this section provides an incremental cost model for every layer. In the remainder of this section we differentiate two different types of costs. The first one is the added cost. This is the cost that is added when a new KPI needs to be implemented but that could not have been avoided by adding the KPI in development phase. An example is the cost of a single sensor. It doesn't matter whether the sensor is bought in development phase or after deployment, the price is the same (or very comparable) in both cases. The second type of cost is the incremental cost, this is the cost that is introduced by adding a KPI to an already deployed application but that would have been avoided by implementing the KPI in development phase. In the cost models given below, the added cost is displayed as green components while the incremental cost is displayed as red components. White components are components where the cost is not affected by adding a new KPI

#### A. The Perception Layer

The perception layer is the layer responsible for sensing the real world. It includes sensors, actuators, processors, firmware. ... All hardware, software and firmware components of the perception layer are aggregated under the name of smart objects. These smart objects need to be adapted when KPIs are added after development phase. To add new KPIs,

sensors are required for the measurement of these KPIs. The casing of the application may also need to be adopted when an extra sensor needs to fit in. Next to the hardware costs, the certification costs and redeployment costs bring an incremental cost when KPIs need to be implemented after development phase. Figure 2 gives the incremental cost model for the perception layer. The model differentiates the cost to buy or rent smart objects and the cost to make these smart objects. Buying or renting mostly gives a lower startup cost for organizations. Making smart objects, on the other hand, provides much more customised smart objects. Next to that, an import cost may need to be paid when sensors are bought abroad. The production and maintenance cost are not affected by implementing a new KPI after development phase. The need for redeployment after the implementation of a new KPI is called the redeployment cost in figure 2 and adds an incremental cost to the application.

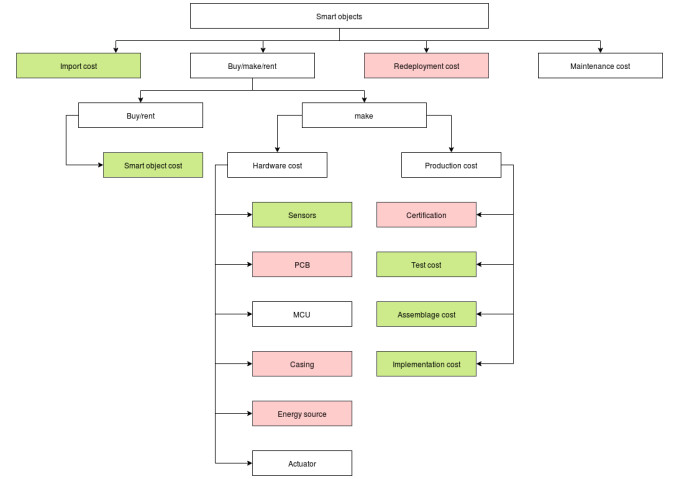


Figure 2. The incremental cost model for the perception layer

#### B. The Communication Layer

The communication layer provides communication between the perception layer and the platform layer. This means it takes care of the data provided by the smart objects and sends this data to a server where it can be processed and/or stored. An organization can choose to either use a private or public network to provide communication for its application. Both options require subscription to a (mostly paid) network. When adding KPIs to the application, more frequently or larger (in terms of bytes) communication may be needed which brings an added cost. Figure 3 gives an overview of the different cost components of the communication layer.

#### C. The Platform Layer

The platform layer contains the data storage and data processing for the data received from the perception layer through the communication layer. The platform layer stores data from and metadata about the application. Device management of the different gateways enables the application to send uplink

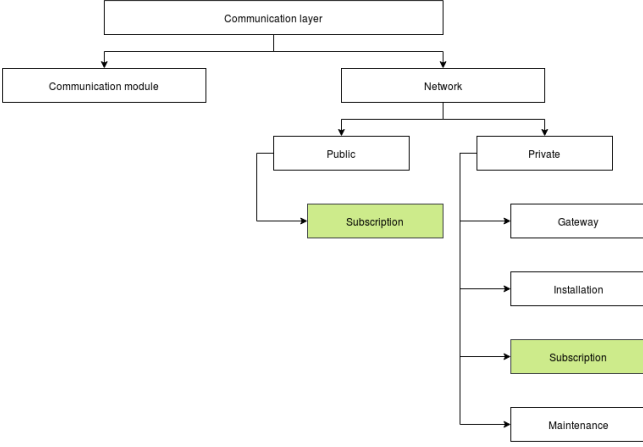


Figure 3. The incremental cost model for the communication layer

packets to the different nodes it wants to communicate with. Adding new KPIs to the application will produce an added cost in the data storage components of the platform layer. Figure 4 shows the cost components of the platform layer.

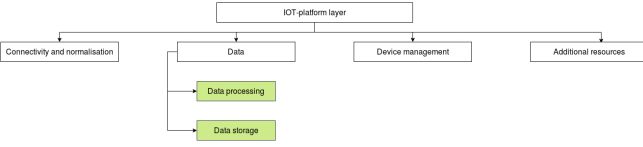


Figure 4. The incremental cost model for the platform layer

#### D. The Application Layer

The application layer is, as the name mentions, very specific for each IoT application. The layer is a connection of IoT technologies and sector professional technologies. Its main goal is to share the information produced by the smart objects and secure the information safety. It provides specific services to the end users through analyzed and processed data. As this layer contains mostly software, its main cost components are development, deployment, maintenance and expansion costs. Adding new KPIs to the application clearly belongs to this expansion cost. The nature of software learns that expanding an application almost always requires a larger effort than implementing the required feature in development phase. For this, the expansion component is displayed as an incremental cost in figure 5.

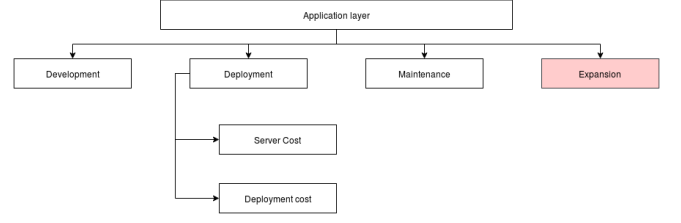


Figure 5. The incremental cost model for the application layer

KPIs for an IoT application. The tool should be used by organizations who want to discover new KPIs they did not think about at first. This in order to avoid the incremental costs described in section III. The architectural design of the tool is given in figure 6. The methodology consists of 6 main parts, each explained in detail in the sections below.

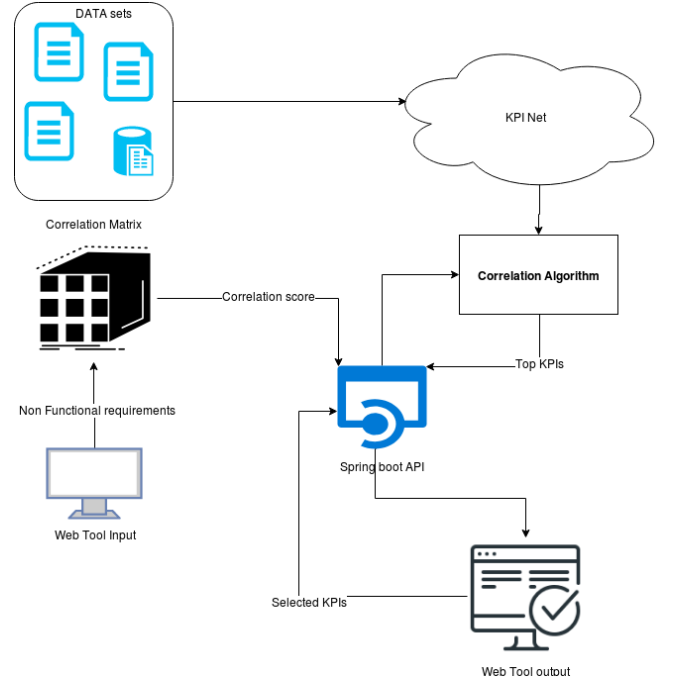


Figure 6. The main components of the KPI suggestion tool

#### A. User input front-end

The first part of the tool is the user input, situated in the lower left corner of figure 6. Here, the user estimates the non-functional requirements of his application on a likert scale (from 1 to 7). The choice was made to focus on non-functional requirements as these are more general compared to functional requirements. The non-functional requirements are sent to the correlation matrix where a score for each KPI is determined (see section IV-C).

#### B. KPI net

The second part of the application is the KPI net. This KPI net is a relational database where all data is stored. It contains

### IV. KPI SUGGESTION TOOL

This section handles the methodology for the KPI suggestion tool. This tool takes user input in the form of non-functional requirements. A non-functional requirements is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Examples of such requirements are: security, availability, maintainability. Based on this non-functional requirements the tool suggests useful

each KPI, the data sources for every KPI, the non-functional requirements described above and the correlations between these requirements and the KPIs. In total 13 requirements and 46 KPIs are stored in the KPI net. The back-end application (see section IV-D) provides endpoints to add KPIs and requirements as future improvements.

### C. Correlation matrix

The correlation matrix provides a mapping between the non-functional requirements of the application and the KPIs stored in the KPI net. The KPI net contains correlation coefficients (CC) that determine how important each non-functional requirements is for each KPI. The correlation uses this score to transform the ratings of the non-functional requirements into a score for each KPI in the KPI net using equation 1.

$$S_k = \sum_{r \in \Omega} C(r, k) * I(r) \quad (1)$$

With

- $S_k$  : Score for KPI k
- $\Omega$  : Collection of all requirements
- $C(r, k)$  : CC between requirement r and KPI k
- $I(r)$  : Score for requirement r as rated by the user

The KPIs with the best score will be marked as the list of top KPIs for the application. The scores for every KPI, computed as in equation 1 are sent to the back-end application.

### D. Back-end application

The data from the KPI net and the output of the correlation matrix are both sent to the back-end api which combines both parts to find a set of suggested KPIs. First, the back-end selects a list of top KPIs based on the scores received from the correlation matrix. The back-end now searches for other suggested KPIs which have a lot of common data sources with this list of top KPIs. The correlation algorithm (see section IV-E) is responsible to find the KPIs with the lowest cost. These suggested KPIs are considered the so called "low hanging fruit" KPIs for the organization. This means the suggested KPIs are the ones that can be implemented with minimal effort for the organization. The back-end returns these top KPIs together with the suggested KPIs to the front-end user output. The back-end also provides API endpoints to add KPIs, requirements and correlations to the KPI net.

### E. Correlation Algorithm

The correlation algorithm determines a set of suggested KPIs based on the list of top KPIs. It uses a cost function for this task. At the moment this cost function is the amount of data sources. Using this cost function together with the score as calculated in equation 1, the correlation algorithm determines a relative cost score for each KPI using equation 2.

$$R_c(k) = \frac{a(1 + c(k))}{b * S_k} \quad (2)$$

With

- $R_c(k)$  : The relative cost of KPI k
- $c(k)$  : The cost of the KPI
- $S_k$  : The score of KPI k based on the requirements
- $a$  : The weight coefficient for the cost
- $b$  : The weight coefficient for the score

The weight coefficients used in the above equation enable the possibility to give more importance to the cost of the KPI. Notice the formula uses  $1 + c(k)$  in the numerator in order to always retrieve a cost larger than zero. If this would not be used, total irrelevant KPIs that do not need extra data sources will always be suggested. The KPIs with the lowest relative cost are selected as suggested KPIs.

### F. User output front-end

The result of the back-end, this is the top KPIs and the suggested "low hanging fruit" KPIs are sent to the user as output of the tool. The user can now select which KPIs he wants to implement. Based on this selection the user can choose to recalculate the suggested KPIs. The selected KPIs will now serve as the list of top KPIs and the back-end will select new "low hanging fruit" KPIs based on this selection. This is an iterative process that continues until the user is satisfied with the current suggested KPIs.

## V. VALIDATION

This section validates the above described methodology. The application used for validation is the Sundo use case. Sundo is a startup company that provides smart sunscreen dispensers to cities. Accommodated with sun information like the actual UV-index and temperature the dispensers offers inhabitants the possibility to protect against the sun. With a user friendly on-line dashboard, the city can monitor the consumption of each dispenser in a simple way. In addition, automatic notifications about dispensers that are almost empty offer the possibility to plan optimised routes to refill the empty dispensers.

As Sundo develops a new application it is an ideal use case to test the KPI suggestion tool. To test the tool, the importance of every non-functional requirement was estimated. Also, all KPIs have already taken into account are listed. The KPI suggestion tool processes this input and provides the suggested KPIs it has found as output. Afterwards a comparison between the initial KPIs and the results from the KPI suggestion tool was done. This validation was done both with and without the option to iterate on the returned KPIs by the tool (described in section IV-F). Sundo needed 3 iterations for the suggestion tool to reach a stable solution. In this solution, 8 new KPIs were found that were considered worth investigating. This states the above described methodology is indeed capable of suggesting useful KPIs for an IoT application.

## VI. CONCLUSION

This work describes a new methodology to discover potentially interesting KPIs for IoT use cases. It starts by listing the cost components for an IoT-application. Based on these cost models the required effort to add a new KPI to the IoT-application is estimated. The cost models differentiate between added cost

and the incremental cost. Adding a KPI to the application will in some cost components bring an extra cost. When this cost is the same as when the KPI is implemented in development phase, we call this the added cost. This cost is inevitable and is specific to the facts that KPIs are dynamic. It is however possible that costs, due to adding KPIs to the application, could have been avoided by implementing these KPIs in development phase. This is called the incremental cost and allows a company to get an impression which effort it needs to deliver to add a new KPI to the application.

The incremental cost models given in this work are based on a 4 layer architecture for IoT applications. The first layer is the perception layer which includes the data sources needed for the application. This perception layer communicates with the platform layer through a communication layer. The platform layer is, among other tasks, responsible for preprocessing and storing the data provided from the smart objects layer. At last the application layer presents the stored data to the end user in such a way it creates a useful application. The cost models should be used together with the companies knowledge about the application. The models give a first impression on which cost components will create an incremental cost when KPIs are added. The models don't say much about what the magnitude of this cost is, as this dependent on multiple variables. The number of devices, cost of the needed data sources, and type of application are only some variable factors that determine the size of the cost for adding new KPIs. The first part of this thesis has as main purpose to offer the end user an overview on which type of costs he will face when missing a KPI in development phase.

As the incremental cost analysis gives an overview of the different cost components for adding KPIs to an IoT-application, the second part offers the end user a KPI suggestion tool to predict which KPIs are useful for his application. The tool starts with the non-functional requirements of the application and processes this input together with a prestored KPI net to suggest the user with useful KPIs.

When combined, the incremental cost analysis and KPI suggestion tool have a great value for companies. The suggestion tool offers an additional resource to help companies to determine KPIs they may not have thought of before. This way the chance of missing KPIs in development phase decreases for a company. When the suggestion tool suggests KPIs but the company is not sure whether or not this KPI will be useful in the future, it can use the incremental cost analysis to compare the cost of adding the KPI in the development phase of the application to the cost of adding the KPI in a later stage.

## VII. FUTURE WORK

The above described incremental cost models list the main incremental and added cost components for adding KPIs to an already deployed IoT-application. The models indicate the different cost components of an IoT application and illustrate which costs to expect when adding KPIs to these applications. The models however, do not give any clues about the magnitude of these costs or which the largest cost drivers are. With the current model, an organization can identify these costs but

needs to estimate the size of this cost for itself. As a first improvement the different costs and their magnitude should be listed and described. This way, organizations can, instead of only identifying the cost based on these models, also predict very precise how much missing a KPI for an IoT application will cost and which are the main cost drivers.

This work also provides a first methodology and suggestion tool to predict useful KPIs. In future work the KPI net should be expanded and more literature and investigation should be done into how KPIs correlate with the non-functional requirements for an application. When more data sets of KPIs can be found, the distances between the KPIs in the KPI net could be expanded. At the moment, the only distance between KPIs is their common data sources. This metric could be combined with the correlation found by investigating data sets from two or more KPIs.

A feedback loop, that allows the suggestion tool to learn and improve from previous selected KPIs based on non-functional requirements input could be a first step to predict KPIs using a machine learning approach. The correlation coefficients between KPIs and requirements are now based on literature. In the future it would be beneficial to provide a learning system that changes these correlations based on selected KPIs by the organizations. The correlation score between a KPI and a requirement should, whenever the KPI is selected as a KPI to be implemented by the organization, be increased everytime the input score for this requirement is rather high or decreased when the input score is rather low. Such an implementation needs profound research to make sure the scores don't change too drastically based on a single user's choice of KPIs and can be reset when needed.

## REFERENCES

- [1] D. Niyato, X. Lu, P. Wang, D. I. Kim, and Z. Han, "Economics of Internet of Things: An information market approach," *IEEE Wireless Communications*, vol. 23, no. 4, pp. 136–145, 2016.
- [2] G. De Cock, "Techno-economische kostmodellering voor IoT-technologieën via implementatie van een online selectie- en modelleringstool," 2019.
- [3] S. Lucero, "IoT platforms: enabling the Internet of Things," *IHS Technology*, vol. Whitepaper, no. March, pp. 1–19, 2016.
- [4] M. Leonard, "Declining price of IoT sensors means greater use in manufacturing — Supply Chain Dive." [Online]. Available: <https://www.supplychaindive.com/news/declining-price-iot-sensors-manufacturing/564980/>
- [5] P. David, *Key Performance Indicators: developing, implementing, and using winning KPIs*, 2013, vol. 53, no. 9. [Online]. Available: <https://books.google.be/books?hl=nl&lr={&}id=bKkxBwAAQBAJ&oi=fnd&pg=PA101&dq=Key+performance+indicators+:+developing,+implementing,+and+using+winning+KPIs+&ots=cY{ }Y10h-bx{&}sig=A7YcIJzkOisUWmdgvuz9N1tPO0{#}v=onepage{&}q=Keyperformanceindicators{ }3Adeveloping{ }2Cimplementing{ }2CandusingwinningKPIs{&}f=false>
- [6] M. Fukuda, J.-J. Jeng, and Y. Li, "United States US 20090281 845A1 (12)," Tech. Rep. 10. [Online]. Available: <https://patentimages.storage.googleapis.com/cd/b6/e3/22a91ecfddb8ba/US20090281845A1.pdf>
- [7] KPIlibrary, "Key Performance Indicators (KPI) Examples, Dashboard & Reporting." [Online]. Available: <http://kpilibrary.com/>

- [8] P. Willaert, J. Willems, and S. Viaene, "740 2006 IRMA International Conference Process Performance Measurement: Identifying KPI's that Link Process Performance to Company Strategy," Tech. Rep., 2006. [Online]. Available: <http://www.idea-group.com>



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Situation of this master dissertation . . . . .	1
1.2	Problem Description . . . . .	2
1.3	Goal . . . . .	4
1.4	Structure of this master dissertation . . . . .	5
<b>2</b>	<b>Literature study</b>	<b>7</b>
2.1	What is IoT? . . . . .	7
2.2	What are KPIs? . . . . .	9
2.3	Cost analysis literature . . . . .	10
2.3.1	Architecture of IOT-applications . . . . .	10
2.3.2	Cost structure of IOT-applications . . . . .	12
2.4	KPI suggestion tool literature . . . . .	14
2.4.1	State of the art analysis . . . . .	15

2.4.2	KPI suggestion tool . . . . .	18
<b>3</b>	<b>Methodology</b>	<b>21</b>
3.1	Incremental Cost Analysis . . . . .	22
3.1.1	Introduction . . . . .	22
3.1.2	Cost Structure . . . . .	22
3.1.2.1	Perception Layer . . . . .	23
3.1.2.2	Communication layer . . . . .	26
3.1.2.3	Platform layer . . . . .	27
3.1.2.4	Application layer . . . . .	27
3.1.3	Incremental Cost Structure . . . . .	28
3.1.3.1	Perception layer . . . . .	29
3.1.3.2	Communication layer . . . . .	32
3.1.3.3	Platform layer . . . . .	33
3.1.3.4	Application layer . . . . .	34
3.1.3.5	Final model . . . . .	34
3.1.4	Future Improvements . . . . .	35
3.2	KPI Suggestion Tool . . . . .	37
3.2.1	Introduction . . . . .	37
3.2.2	Components . . . . .	40
3.2.2.1	User input front-end . . . . .	40

3.2.2.2	KPI net . . . . .	41
3.2.2.3	Correlation matrix . . . . .	42
3.2.2.4	Back-end of the application . . . . .	46
3.2.2.5	KPI calculation algorithm . . . . .	49
3.2.2.6	User output frontend . . . . .	51
3.2.3	Future improvements . . . . .	53
3.2.3.1	KPI clustering . . . . .	54
3.2.3.2	KPI expansion . . . . .	57
3.2.3.3	Machine Learning approach to improve correlations . . . . .	57
3.3	Combining both methodologies . . . . .	58
<b>4</b>	<b>Validation</b>	<b>61</b>
4.1	The Sundo use case . . . . .	61
4.2	Validation Strategy . . . . .	62
4.3	Validation Outcome . . . . .	64
<b>5</b>	<b>Conclusion</b>	<b>69</b>
<b>6</b>	<b>Future improvements</b>	<b>73</b>
6.1	Incremental cost analysis . . . . .	73
6.2	KPI suggestion tool . . . . .	74

## Appendices

<b>Appendix A Questionnaire for IoT Companies</b>	<b>85</b>
<b>Appendix B Content of the KPI net</b>	<b>89</b>
B.1 Non-functional requirements . . . . .	89
B.2 KPIs . . . . .	91
<b>Appendix C Deployment of the application</b>	<b>97</b>
C.1 Download the application . . . . .	97
C.2 Install docker . . . . .	98
C.3 Deploy the application . . . . .	99
<b>Appendix D Expanding the KPI net</b>	<b>101</b>

# List of Figures

1.1	Ideal schematic overview of the creation of an (IOT) application	3
1.2	Lifecycle of how KPIs are used [1] . . . . .	4
2.1	The four phases of the industrial revolution [2] . . . . .	9
2.2	The 4-layer architecture used in the remainder of this work . .	12
2.3	The 5-layer architecture of an IOT-application . . . . .	13
2.4	The average price of an IOT-sensor is falling [3] . . . . .	14
2.5	The correlated KPIs as used in the work of Stricker [4], this idea is used as basis for the correlations in the KPI net . . . .	19
3.1	A schematic overview of the 4-layer architecture . . . . .	23
3.2	Overview of the components of the smart objects layer . . . .	25
3.3	Overview of the components of the connectivity layer . . . . .	26
3.4	Overview of the components of the platform layer . . . . .	27
3.5	Overview of the components of the application layer . . . . .	28

3.6	The total cost structure for an IOT-application as described in section 3.1 . . . . .	30
3.7	Added and incremental cost for smart objects . . . . .	32
3.8	Added and incremental cost for communication layer . . . . .	33
3.9	Added and incremental cost for the platform layer . . . . .	34
3.10	Added and incremental cost for the application layer . . . . .	34
3.11	Final incremental cost model for all layers . . . . .	36
3.12	Schematic overview of methodology for the KPI suggestion tool	38
3.13	A typical user story to use the KPI suggestion tool . . . . .	39
3.14	Example of the front-end user input . . . . .	41
3.15	The EER model of the KPI net . . . . .	42
3.16	Conceptual overview of the working of the correlation matrix .	45
3.17	The three layer architecture for the back-end application . . .	47
3.18	Class diagram for the back-end . . . . .	48
3.19	A screenshot of the front-end output the user gets to see . . .	53
3.20	The two step method of XingYu [5] to cluster KPIs . . . . .	55
3.21	Methodology for the KPI suggestion tool with clustering . . .	56
3.22	Combining both the KPI suggestion tool and incremental cost analysis . . . . .	59
C.1	Steps to download the application from github . . . . .	98
D.1	Printscreen of the add KPI page . . . . .	102

D.2	Printscreen of the add Requirements page . . . . .	103
-----	--	-----





# List of Tables

2.2	Recommendation for good performance measures as stated by Neely [6] . . . . .	11
2.4	The six step implementation plan as proposed by Parmenter [7]	17
4.1	The input scores for the non-functional requirements as given by the Sundo organization . . . . .	65
B.1	Overview of the non-functional requirements present in the KPI net . . . . .	89
B.2	Overview of the KPIs present in the KPI net . . . . .	91



# List of Algorithms

1	Code responsible for the transformation of the user input in the correlation matrix . . . . .	45
2	Pseudocode for retrieving the top KPIs based on the user's non-functional requirement input . . . . .	52
3	Pseudocode for the KPI suggestion algorithm . . . . .	52

## List of Abbreviations

**API** Application Programming Interface

**CAPEX** Capital Expenditures

**CBS** Cost Breakdown Structure

**DAO** Data Access Objects

**DTOs** Data Transfer Objects

**DW** Data Warehouse

**GW** Gateway

**IoMT** Internet of Medical Things

**IoT** Internet of Things

**JSON** Javascript Object Notation

**KPI** Key Performance Indicator

**KPIs** Key Performance Indicators

**M2M** Machine-to-machine

**MCU** Microcontroller

**OPEX** Operational Expenditures

**ORM** Object-relational Mapping

**PCB** Printed Circuit Board

**RFID** Radio-Frequency Identification

**TEA** Techno-economic Analysis

**UIDs** Unique Identifiers

**V2X** Vehicle-to-Everything

# Chapter 1

## Introduction

### 1.1 Situation of this master dissertation

The last recent years, a new wave in the era of computing found its way to the spotlights. Internet of Things (IoT) emerged as a concept about 20 years ago and is now making headlines all around the world. In the internet of things (IoT) paradigm, people are surrounded with everyday devices connected to the internet. These devices take many forms. Going from smart thermostats over connected doorbells to connected healthcare applications. IoT opens up a whole range of new services and applications in many domains. The growth of IoT devices is spectacular. From 15.46 billion connected devices in 2015 to 26.66 billion nowadays [8] (including smartphones, connected cars...). Predictions say this number will increase up to 75.44 billion connected devices in 2025 [8]. Worldwide spending on IoT devices is also on the rise, with IDC's Worldwide Semiannual Internet of Things Spending Guide predicting that global spending in IoT will leap from over \$ 800 billion in 2017 to \$ 1.4 trillion by 2021 [9]. IotAnalytics [10] estimates the number of pure IoT applications (connected devices without smartphones, laptops, tablets etc.) around 7 billion nowadays and expects this number to grow to 21 billion by 2025. The different types of IoT devices cover a very wide range. Control and automation of lighting and heating (Smart Home), blood pressure and heart rate monitoring enabling remote healthcare (mHealth), vibration monitoring in a machine in order to perform preventive maintenance (smart industry) are only some applications for this very quickly expanding paradigm.

Key Performance Indicators (KPIs) are the critical (key) indicators of success towards an intended result. KPIs provide a focus for strategic and operational improvement, create an analytical basis for decision making and help focus attention on what matters most. They represent a set of measures focusing on the most critical aspects for an organization both in the present and in the future. Well chosen KPIs provide objective evidence of progress towards achieving a desired result, give insights for better decision making and offer a comparison in the degree of performance change over time. Dependent on the type of project, importance of different functional and non-functional requirements and domain of the project, different KPIs must be formulated.

## 1.2 Problem Description

The gain from IoT often lays in the fact that the data produced by the sensors on these devices allows to take appropriate actions. Based on well chosen Key Performance Indicators, data analytics are used to discover correlations between captured data points. Since data is a product that results from most IoT applications, it is often not yet available in a development phase. In addition, data analytics can only generate insights based on what is measured by the IoT device, possibly supplemented with external data. Thus, aspects not measured by the IoT device cannot be included in the analysis. Therefore, to maximize the expected impact of the IoT application (e.g. increased operational efficiency) whilst minimizing the costs for post-deployment data analysis and potential refactoring costs (e.g. adding a sensor in an existing and already deployed solution), one needs to have a clear view on what to measure and how to monitor certain aspects of the IoT application at an early stage of the IoT development.

Figure 1.1 gives an schematic overview of the ideal steps in the creation of an (IoT) application. The most left side of the figure represents the ideation phase where the idea is born. After the ideation phase, ideally, a Techno-economic analysis (TEA) follows. In this TEA, an organization takes several decisions towards an IoT solution. The first question to ask is whether or not to make the application an IoT application. For this, there must be obvious benefits of choosing for an IoT solution. If not, the overhead produced by creating a "smart" application may not be worth the benefits. When the question about whether or not to create an IoT solution is solved, an orga-

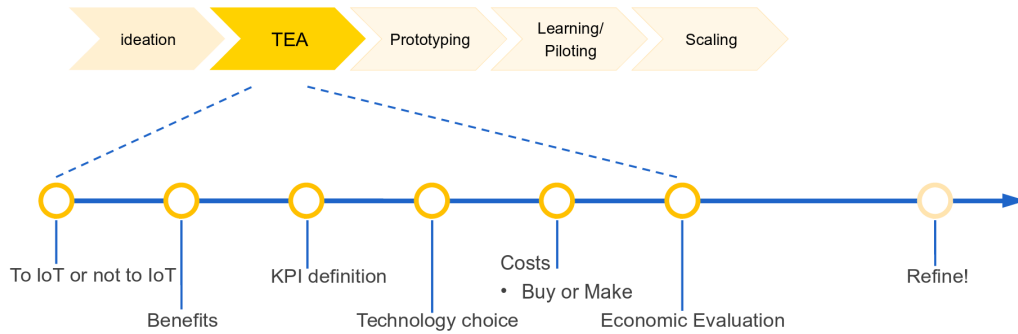


Figure 1.1: Ideal schematic overview of the creation of an (IoT) application

nization must define its KPIs to monitor the performance of the application. When all KPIs are defined, a cost structure of the application allows for an economic evaluation of the application. When this evaluation is positive, the TEA can be refined and the prototyping step starts. This thesis focuses on this techno-economic phase. After the TEA, a company typically creates a prototype of the product and starts to work on a pilot after which the final product is scaled to the needed amount. Whilst the phases after the TEA also possess great challenges, this is not the focus of this thesis. Nevertheless, a complete overview is given because bad chosen KPIs are carried along all other phases. Figure 1.2 illustrates the problem of choosing bad or missing KPIs. This problem emerges most of the time after the application is already deployed and scaled. This way huge economic efforts can be needed when one decides to define other KPIs than the ones primarily chosen in the TEA phase.

As described above, the main challenges IoT applications are facing nowadays concerning KPIs are the following:

- KPIs are determined and defined based on gut feeling nowadays.
- Data to evaluate the KPI is not accessible in development phase.
- Adding a KPI to an IoT application may require a huge economic effort after development phase.



Figure 1.2: Lifecycle of how KPIs are used [1]

### 1.3 Goal

The first goal of this thesis is to gain insights into the different costs in the development and deployment of an IoT application. The creation of a cost structure of an IoT application enables the possibility to understand how KPIs can affect the cost of an Internet of Things application. More specifically this work offers cost models that enable a comparison between the cost of adding KPIs in development phase and the cost of adding these KPIs in a later stage. The cost models also helps organizations to make a decision about the gains of using IoT technologies for the application. While the measurement of a KPI brings an added cost at first, this cost may later be justified by the gains of the measurement in later stages of the application.

Based on the above described cost structure a methodology for a KPI suggestion tool will be given. Via this tool, organizations describe the domain-aspects or settings of their IoT application. The tool then provides KPIs and metrics as source of inspiration and cross-checking the earlier formulated KPIs. Users see the added value of adding a KPI in the development of the application compared to the cost it will cause when they decide to ignore



the KPI for now and implement it in a later stage. Every KPI needs (a number of) data sources in order to measure this KPI. The KPI suggestion tool uses the number of common data sources between KPIs as a distance metric. The KPIs that are most relevant to the application based on the input of the organization and that have the closest distance to the most important KPIs for this input are suggested by the KPI suggestion tool.

## 1.4 Structure of this master dissertation

In what follows, a short overview of what the reader can expect in the remainder of this work is given.

Chapter 2 gives a literature study about IoT and KPIs and how these concepts influence our everyday life and the structure of organizations that work with them. Furthermore, a basic architecture for IoT applications is given. A literature study about different cost components for IoT applications gives some first insights in how an incremental cost analysis can be developed. At last, literature about how KPIs are suggested nowadays and models to predict KPIs for a new IoT application are given. This helps to develop a new methodology and KPI suggestion tool to predict KPIs for an organization.

Chapter 3 starts with the incremental cost analysis. The different cost components of an IoT application are given for every layer of the 4-layer architecture as described in chapter 2. These cost models help determine the incremental cost of adding KPIs in a later stage and help to compare the cost between adding KPIs in development phase and adding the same KPIs in a later phase. The second part of this chapter focuses on the KPI suggestion tool by explaining its methodology and architecture.

Chapter 4 starts by describing the Sundo use case. Sundo is a startup company that delivers a service to cities and inhabitants. Using smart sunscreen dispensers they allow cities to provide sunscreen for their inhabitants. This use case allows to validate the KPI suggestion tool and check if the tool proposes useful KPIs for this organization.

At last, chapter 5 provides a conclusion and wrap up of this work. Some important insights and future improvements are also listed.



# Chapter 2

## Literature study

### 2.1 What is IoT?

The Internet of Things (IoT) is the interconnecting of physical devices (also referred to as "smart" or "connected" devices). The basic idea of this paradigm is the pervasive presence around us as a variety of things or objects which are able to interact and cooperate with each other to reach common goals [11]. These things or objects are found in many different shapes. They are the smartphones we use daily, smart thermostats, connected doorbells, basically a broad range of sensors, actuators, Radio-Frequency IDentification (RFID) tags etc. Unique Identifiers (UIDs) provide these devices with the ability to be defined in an unambiguous way. These UIDs offer the smart devices the ability to transfer data over a network without requiring human-to-human or human-to-computer interactions [12]. The extensive set of applications for IoT devices is often divided into consumer, commercial, industrial and infrastructure spaces [12]. Notice some of these spaces may contain some overlapping devices/applications.

The consumer space covers the devices connected to the internet that are created for consumer use. These include connected vehicles, home automation, wearable technology, connected health and appliances with remote monitoring abilities [13]. Home automation includes lighting, heating and air conditioning, media and security systems. Examples of such systems are Amazon echo [14], Google Home [15] or Apple's Homekit [16]. Wearable

technology devices can be incorporated into clothing or worn on the body as implants or accessories. Activity trackers are an example of such kind of devices.

The commercial applications include medical and healthcare applications, also referred to as Internet of Medical Things (IoMT). Smart beds have found their implementation into some hospitals, detecting if they are occupied or if a patient is attempting to get up. Next to IoMT the commercial space for IoT application includes devices used for transportation, and vehicle-to-anything (V2X) communication which is the first step to autonomous driving and road infrastructure. Devices used to monitor and control the mechanical, electrical and electronic systems, used in various types of buildings and building automation systems, can help in building and home automation applications.

IoT applications offer great help in industrial sectors. Manufacturing has made huge improvements using IoT solutions. IoT even has such big influence on the industrial sector it has led to the fourth industrial revolution [17]. This new industry is called industry 4.0 or smart industry. An overview of the revolutions to industry 4.0 is given in figure 2.1

Machines augmented with wireless connectivity and sensors are connected to systems that can visualise the entire production line and make decisions on their own. The results of this approach are very promising. Applications include machines which can predict failures and trigger maintenance processes autonomously or self-organized logistics which react to unexpected changes in production. Smart industry allows better and faster optimisation for processes within an organization. Letting machines communicate directly to one another also reduces lead times, increases efficiency and reduces the risk on errors.

The last application space where IoT applications create an added value are infrastructure applications. Several planned or ongoing deployments of IoT applications enable better management of cities and systems. Smart cities use IoT applications to collect data and gain insights in this data. This way enabling a more efficient energy management, monitoring and managing of traffic streams, environmental monitoring etc.

As stated above IoT is ubiquitous nowadays. The use cases for IoT applications are almost endless and IoT has found its way into a very broad range of domains. The number of IoT applications will only grow in the future. Increasing technology such as 5G and the global adoption of Ipv6 will only encourage this growth making IoT even more indispensable in our

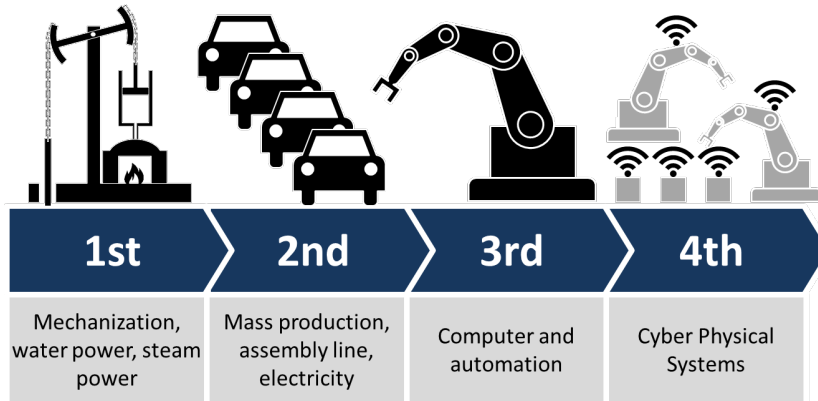


Figure 2.1: The four phases of the industrial revolution [2]

everyday life.

## 2.2 What are KPIs?

Key performance Indicators (KPIs) are measurable values that allow a company to value how effectively it is achieving its key business objectives. Many organizations use KPIs at multiple levels to evaluate their success at reaching targets. High-level KPIs may focus on the overall performance of the business, while low-level KPIs may focus on processes in departments such as sales, marketing, HR, support and others [18]. Well defined and meaningful KPIs are of great value to any organization. KPIs provide the opportunity to check whether or not the company is achieving what it wants to achieve.

To formulate KPIs, every company should start with stating what the organizational objects are, how they plan on achieving them and who can act on this information. This iterative process involves feedback from analysts, department heads, consultants and managers. Every KPI should be very clear and well defined in order to be useful. A lot of question lists exist online to help creating well defined KPIs. Some of the most important questions that lead to good KPIs are given below:

- What is your desired outcome?
- Why does this outcome matter?

- How are you going to measure progress?
- How can you influence the outcome?
- Who is responsible for the business outcome?
- How will you know you've achieved your outcome?
- How often will you review progress towards the outcome?

As stated by Neely et al. [6] good performance measures should follow the recommendations as stated in table 2.2

Questions and guidelines such as described above help organizations to define their KPIs in a proper way. But the problem of how to know if every possible KPI for the organization is found remains unsolved. This work proposes an incremental cost analysis by which organizations can estimate the cost of missing KPIs in development phase and thus having to implement these KPIs after the deployment of the application. The second part of this work focuses on a KPI suggestion tool to suggest KPIs the organization may not have thought of before. This way reducing the risk of missing KPIs and avoiding the economic effort this entails.

## 2.3 Cost analysis literature

This section handles the literature study about the cost analysis of IoT applications. To develop an incremental cost model, all costs of an IoT application must be known and described. Based on this literature study, an overview of all costs for an IoT application can be given which serves as the basis for the incremental cost analysis described in section 3.1

### 2.3.1 Architecture of IOT-applications

The architecture of an IoT application is the starting point to define different cost models for these applications. Ghulak et al. [19] gives a first architectural design for IoT applications. In their work, they compare two- and three-tier IoT architectures. The two-tier applications consist of an IoT

	<b>Recommendation</b>
1	Performance measures should be derived from strategy
2	Performance measures should be simple to understand
3	Performance measures should provide timely and accurate feedback
4	Performance measures should be based on quantities that can be influenced, or controlled, by the user alone or in co-operation with others
5	Performance measures should reflect the “business process” – i.e. both the supplier and customer should be involved in the definition of the measure
6	Performance measures should relate to specific goals (targets)
7	Performance measures should be relevant
8	Performance measures should be part of a closed management loop
9	Performance measures should be clearly defined
10	Performance measures should have visual impact
11	Performance measures should focus on improvement
12	Performance measures should be consistent (in that they maintain their significance as time goes by)
13	Performance measures should provide fast feedback
14	Performance measures should have an explicit purpose
15	Performance measures should be based on an explicitly defined formula and source of data
16	Performance measures should employ ratios rather than absolute numbers
17	Performance measures should use data which are automatically collected as part of a process whenever possible
18	Performance measures should be reported in a simple consistent format
19	Performance measures should be based on trends rather than snapshots
20	Performance measures should provide information
21	Performance measures should be precise – be exact about what is being measured
22	Performance measures should be objective – not based on opinion

Table 2.2: Recommendation for good performance measures as stated by Neely [6]

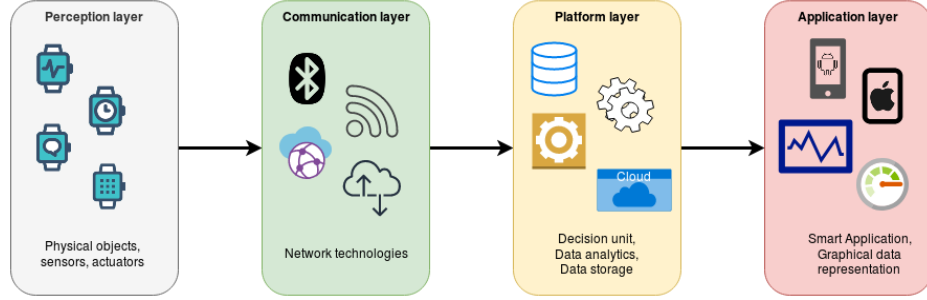


Figure 2.2: The 4-layer architecture used in the remainder of this work

device tier (the connected devices) and a server tier. In a three layer architecture a middle gateway (GW) device tier connects both tiers. Ghulak also proposes a classification of hardware features which give us a first direction to develop a cost structure for IoT-applications. Niyato et al. [20] uses this architecture and split the server tier into two separated tiers: data storage tier and data processing tier resulting in a four-tier architecture. Munjin et al. [21] introduces marketplaces for IoT-applications by looking at analogies with the marketplaces for smartphone applications. Munjin presents a review of IoT application platforms in order to show the relationship between the software applications and the IoT cloud based services.

To talk about the architecture in an unambiguous way, figure 2.2 shows the 4-layer structure used in the remainder of this work based on the above described literature. As security is gaining importance in IoT applications [22] sometimes a five layer architecture is proposed. This fifth layer is called the security layer and is present in all other layers of the 4-layer architecture. Nowadays security is mostly implemented in every layer separately. Yet, it is possible to provide end-to-end security from the perception layer to the application layer as suggested by PubNub [23]. This way much more secure applications can be build. Figure 2.3 gives this 5-layer architecture. However, at the moment the 4-layer architecture is mostly used. For this, the cost analysis provided in section 3.1 works with this 4-layer architecture as provided in figure 2.2

### 2.3.2 Cost structure of IOT-applications

This section provides a more in depth exploration about the costs of every component based on the above described 4-layer architecture for IoT-



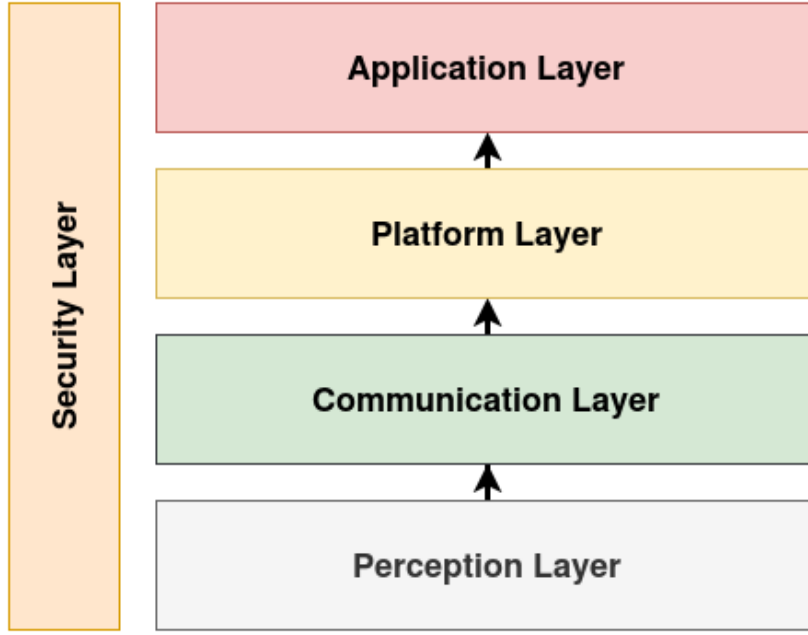


Figure 2.3: The 5-layer architecture of an IOT-application

applications. Section 3.1 uses the results found in the literature study from this section to build cost models for each layer of this 4-layer architecture. Li et al. [24] models a service composition of IoT into a finite state machine which can be transformed into a markov decision proces. This markov decision proces is then extended with a cost structure that gives a first insight in how such a cost structure represents different quality attributes. Combined with the work of Yang et al. [25], which describes how IoT service composition is driven by user requirements, these two papers can give a first attempt for an incremental cost model based on service requirements of the IoT application. The work of Lucero [26] gives more insights in the impact of IoT applications and analyzes the role that IoT platforms play in IoT-applications. The work provides more insights in the important aspects of an IoT platform and helps develop a cost structure for the platform layer as described above. Several cloud solutions concerning storage, such as google cloud engine [27], AWS [28] and Microsoft Azure [29], were consulted to get better insights in the cost of the platform layer for IoT-applications. These cost drivers are also listed in the work of Niyato et al. [30]. In his work, Niyato describes how service providers should address data management in IoT through using smart data pricing. The work proposes a new pricing scheme for IoT service providers. Niyato [30] proposes different pricing schemes for different user needs. These schemes help to determine the cost drivers in our

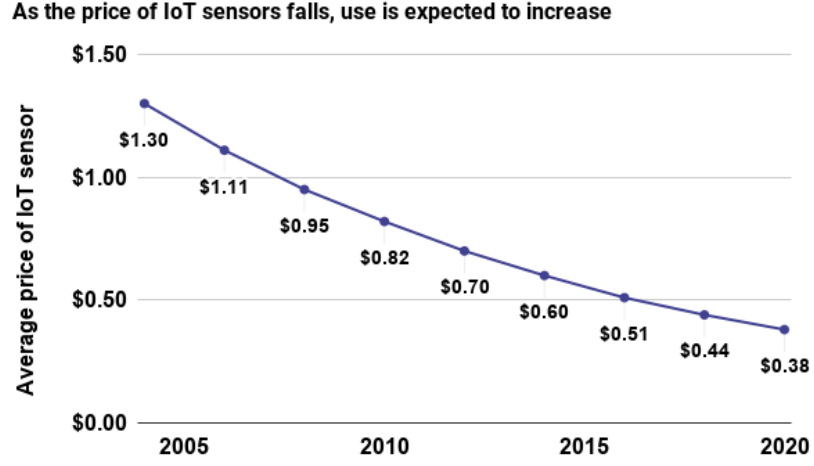


Figure 2.4: The average price of an IOT-sensor is falling [3]

cost model. Concerning the perception layer (which includes the IoT sensors) Leonard [31] states that the average sensor prices are falling. Figure 2.4 shows the trend of the average sensor price. Because of this trend a first assumption is the perception layer will not be a major cost driver in our cost model of IoT-applications. Communication cost mainly defines the cost of the edge layer (also called communication layer later on). A lot of protocols exist for IoT communication, all with their own prices and specifications. Dhillon et al. [32] discusses the need for a wide-area machine-to-machine (M2M) Wireless network. More relevant for this thesis, Dhillon also provides an overview of the current existing communication standards for IoT-applications.

Based on the above found literature, a first impression of the different cost components for each layer in the 4-layer architecture is obtained.

## 2.4 KPI suggestion tool literature

This section handles the literature study about the KPI suggestion tool. First a state of the art analysis shows how KPI suggestion is done nowadays for IoT-applications. In section 2.4.2 literature about how the current state of the art can be improved is combined. Based on this section, chapter 3 presents a new methodology for KPI suggestion in development phase.

### 2.4.1 State of the art analysis

The cost analysis described above shows the importance of defining useful KPIs and gives more insights in the cost of an IoT system. Based on the literature found, missing a KPI can have a huge impact on the cost of an application. To suggest useful KPIs, a methodology for mapping KPIs onto the application requirements is needed. For this purpose, a literature study was done. Besides of this literature study, multiple interviews with IoT experts within organisations were taken. The questionnaire used for these interviews can be found in appendix A. An interview with the team of Sofie Van Hoecke (professor at UGent) gave more insights in the need for a KPI suggestion tool. Her team faces the problems related to missing sensors in applications almost daily. They perform data analysis on big data but are often confronted with the fact that certain necessary sensors and/or data points are not present to make a correct prediction for the KPIs objective. Interviews with experts in the IoT-domain reinforced the need for a KPI suggestion tool. These experts were IoT responsables in big companies such as Deloitte, SAP and Daikin. Other interviews with people who deal with KPIs in their job gave some insights into how KPIs are defined. These people include staff of Imec, Rombit, Televic rail and the retail factory. In total, 17 interviews with people using KPIs were taken. The following remarks were the most important to vindicate the need for a KPI suggestion tool:

- Nowadays, KPIs are defined most of the time based on gut feeling.
- In a lot of cases and applications, data (sources) appears to be missing to measure the wanted KPIs
- The costs of adding data sources to an already deployed application turned out to be quite large

A lot of literature exists about how KPIs can be defined in certain domains. However, as IoT is evolving rapidly the last recent years, there is less literature about how KPIs should be defined for IoT-specific applications. A lot of tools and methodologies to determine KPIs based on the output of other KPIs and/or measured data points already exists. Lei Shi et al. [33] proposes a methodology for KPI identification of KPIs based on Knowledge capturing. Based on measured data points they extract KPIs. As this thesis focuses on KPI definition in development phase this data will most of the time not be available, but an extraction based on data of other

similar applications could lead to some KPIs for the new application based on this method. Parmenter [7] talks in his book about Key performance indicators about how winning KPIs should be defined and proposes a six-stage implementation plan to discover useful KPIs. His six-step procedure (given in table 2.4) provides guidances towards defining KPIs for organizations and how organizations should implement these KPIs in their structure. Until now several other guidelines and best practices about implementing KPIs on to IoT applications have been described. Hwang et al. [34] uses business process modelling to define and select useful KPIs for IoT applications. He proposes a KPI evaluation tool that can change the dynamic structure of business processes. Rakar et al. [35] proposes a methodology for KPI definitions in production process management. Wang pang et al. [36] develops a KPI evaluation system to manage business processes. While Marr et al. [37] gives a review of the existing approaches for measuring knowledge based assets. Marr introduces the knowledge asset map which integrates existing approaches in order to achieve comprehensiveness. The problem of misleading KPIs due to little domain knowledge beforehand is described by Roubtsova et al. [38] The same kind of problem description is stated in the introduction. Roubtsova [38] presents a way to validate the effectiveness of KPIs before implementation. The results of this paper can be very important for this thesis as it will allow to verify whether the suggestion tool returns sensible KPIs. Bauer et al. [39] suggests a way to reduce a network existing of a lot of KPIs to a handful set of useful KPIs. They use the correlation between the KPIs to detect which KPIs are linked together and which are further away from each other. Bauer proposes a validation method to check whether the chosen KPIs are relevant for the business processes of organizations before implementation. Wilkinson [40] shows in a case study how useful KPIs can be found between all KPIs measured in his paper. These techniques can be useful to filter the best KPIs for an application out of a long list of different KPIs. Although a lot of best practices exist, a KPI suggestion tool to suggest KPIs in development phase, as is the goal of this thesis, is not available today [41]. Peral et al. [42] presents a new approach to drive data mining techniques to obtain specific KPIs for business objectives in a semi-automated way. Using data mining techniques on a data warehouse (DW), useful KPIs are extracted.

1	Getting the CEO and senior management committed to the change	The senior management team must be committed to developing and driving through the organization KPIs and any balanced scorecard that includes them. In addition, timing is everything.
2	Up-skill in-house resources to manage the KPI project	The success of a KPI project rests with trained home-grown staff who have been reassigned so that they are full time on the project.
3	Leading and selling the change	All major project implementations are deeply affected by the success or failure in leading and selling the change.
4	Finding your organization's operational critical success factors	Critical success factors (CSFs) are operational issues or aspects that need to be done well day-in and day-out by the staff in the organization.
5	Determining measures that will work in your organization	Many performance measures are created from a flawed process. Numerous methodologies, including the balanced scorecard, appear to simply say the measures are a by-product of the exercise. Frequently the task of finding measures is carried out at the last minute by staff who do not have a clue about what is involved in finding a measure that will create the appropriate behavioral response.
6	Get the measures to drive performance	In order to get measures to drive performance, a reporting framework needs to be developed at all levels within the organization.

Table 2.4: The six step implementation plan as proposed by Parmenter [7]

### 2.4.2 KPI suggestion tool

A lot of techniques found in literature for KPI definition in other domains could be modified to map onto IoT applications nevertheless. Peral [42] proposed a new methodology for extracting relevant KPIs based on the business strategy model of a particular enterprise/activity. A six stage procedure, which exists of 2 business model strategy phases, 2 KPI definition phases and 2 extraction phases, allows an organization to identify new KPIs based on current implemented KPIs. KPIs are organised in a DW where they are modelled based on dependencies such as same data sources. As the goal of the tool this thesis will deliver, is to suggest KPIs before the application is developed, there are no current implemented KPIs. Yet, the idea rose that if we can find enough KPIs from other similar applications the methodology could be relevant for the KPI suggestion tool. KPILibrary [43] contains a lot of useful KPIs with their metrics and formulas currently implemented in applications all over the world. So does the work of Kang et al. [44]. To create the tool, the work of Karlson et al. [45] was used to see how KPI dashboards are created nowadays and where attention is paid to. Samsonowa et al. [46] proposed a way to use clusters to find similarities in performance measurements in different companies. Using this method, similar KPIs of other applications can be found for a new application. Fukuda et al. [47] proposes a way to analyze and design a KPI net based on data mining on existing KPIs. This KPI net, in Fukuda's work is a graph structure with weighted correlations between KPIs on the edges and KPIs as vertices. Based on the work in this paper, a KPI net can be constructed based on previous KPIs of other organizations. This KPI net serves as input for KPI suggestions for new applications. In the paper of Fukuda, correlations between KPIs are tracked to indicate how strong KPIs are linked. Willaert et al. [48] defines a methodology for performance measurement that supports a process-oriented vision on an organization. His methodology combines a bottom-up approach, where operational measurements are being collected on the level of the end-to-end customer focused processes, and a top down approach, in which process goals are cascaded down to the company's core processes. Abe et al. [49] suggests the use of a KPI-network to keep correlated KPIs that can later be used to define useful KPIs for an organization. In their work, Stricker et al. [4] used the fact that several links exist between KPIs. In his work, Stricker defines high level KPIs and searches for correlations between KPIs to construct a KPI network to maximize the potential of a simulated assembly line. The idea of this network is adopted in this work and expanded to multiple domains for IoT applications. Figure 2.5 gives an idea how Stricker stored the

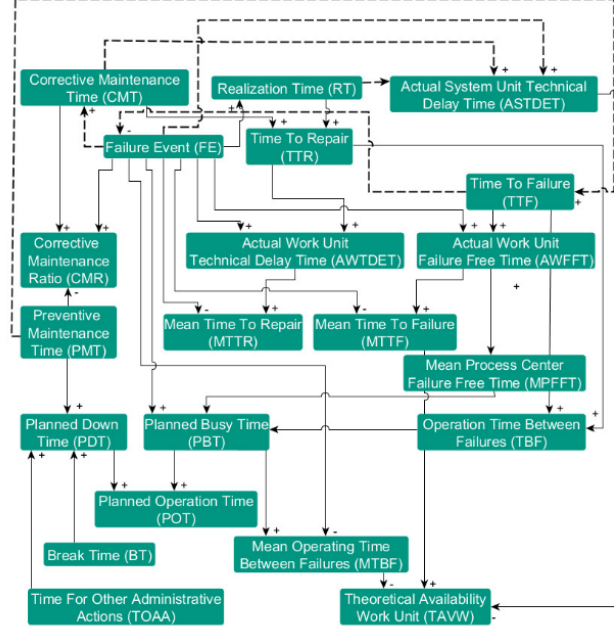


Figure 2.5: The correlated KPIs as used in the work of Stricker [4], this idea is used as basis for the correlations in the KPI net

correlations between KPIs . In this figure the '+' sign stands for directly proportional correlations while the '-' sign stands for inversely proportional correlations. This thesis combines the ideas from these last two mentioned papers to construct a KPI network which will serve as input for the algorithm to suggest the most relevant KPIs. Although Stricker uses the KPI network only to predict machine downtime, his proposed method could serve in a far more general way. As Bauer [39] proposes a way to extract the best set of KPIs from a long list of KPIs, his techniques can be applied on the KPI network.





# Chapter 3

## Methodology

To create a KPI suggestion tool that allows organizations to predict useful KPIs, the problem must be broken down into several smaller steps. The first part of this chapter focuses on the incremental cost analysis for IoT applications. This cost analysis lists all cost of an IoT application based on the 4-layer architecture as described in chapter 2 and figure 2.2. Organizations who want to develop a new IoT application can use this model to discover the costs they need to take into account. Next to that, they can estimate the cost of missing KPIs for their application. The analysis provides a model to estimate the cost of adding KPIs to an application in development phase and compare this cost with the cost to implement the same KPI in a later phase. This cost increases each phase. Adding the KPI in development phase clearly requires a less economic effort compared to adding the same KPI after the prototyping phase. While adding the KPI after deployment and scaling phase requires an even greater effort.

The second part of this chapter focuses on the KPI suggestion tool. This tool takes user input about the application and provides useful KPIs together with their data sources. This way organizations discover "low hanging fruit" KPIs that are less obvious to implement in development phase. The tool can help organizations to prevent the problem of missing KPIs in development phase. It suggests KPIs the organization may not have thought of at first.

In the last section of this chapter, a guidance is provided about how both the incremental cost analysis model and the KPI suggestion tool can

be used by an organization to know which KPIs he wants to implement in his application in development phase. Combining both methods enables organizations to fully understand which KPIs they may need for their IoT application. Based on the incremental cost analysis, they can predict whether or not adding the KPI in a later stage will require a large economic effort.

## 3.1 Incremental Cost Analysis

### 3.1.1 Introduction

This section provides information about, and the creation of, an incremental cost analysis for IoT applications. This incremental cost analysis needs a first model to get an overview of all costs of an IoT application. To list all cost components of an IoT application, the 4-layer architecture of figure 2.2 is used as a starting point. The cost components for every layer are now investigated separately. The cost models in the cost structure section are mainly based on the work of De Cock [50]. The section on the incremental models investigates the costs of adding a data source to an IoT application in a later stage than the development phase. While the models provided in the following sections help to identify these costs, it is the responsibility of the organization to estimate how large the cost will be when a KPI needs to be implemented after the development phase. To help identify the different cost components for expanding the IoT application, the incremental analysis compares the cost between adding a KPI (with given needed data sources to measure this KPI) in development phase and adding the same KPI to the application in a later phase.

### 3.1.2 Cost Structure

Figure 2.2 gives a first overview of the architecture of an IoT application. A cost breakdown structure (CBS) based on this architecture gives the cost for every layer separately and looks for common costs between layers. Two categories of costs are distinguished. The first category consists of the costs needed for the design and development of the application. This includes buying/making sensors, creating a database, creating a platform to monitor

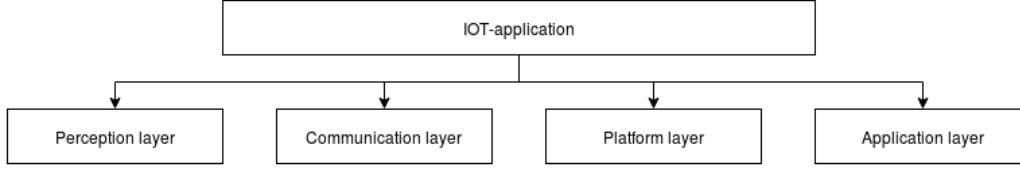


Figure 3.1: A schematic overview of the 4-layer architecture

the IoT application. . . These are the Capital Expenditures (CAPEX) and are needed at the start of the development. The second category, the Operational Expenditures (OPEX), includes the costs of maintaining the IoT application. These costs mainly depend on the scale on which the IoT application is used. Both CAPEX and OPEX have an impact on the incremental cost analysis this master dissertation provides. Section 3.1.3 gives an overview on how these two categories affect the incremental cost model for an IoT application.

To give an overview of the CBS of an IoT application, the architecture from figure 2.2 is used as a starting point. Figure 3.1 gives a more schematic overview of this architecture. In the following sections the different cost drivers for each layer are determined and displayed.

### 3.1.2.1 Perception Layer

The perception layer allows the interaction with the outside world. This layer contains the sensors and other data providers that are used to measure KPIs and application-specific data. This work combines all these sensors, actuators, data providers etc. and aggregates them under the term: smart objects. Definition 3.1.1 gives the definition of smart objects. The perception layer can contain one or several of these smart objects depending on the application.

**Definition 3.1.1.** Smart Object: A smart object is an object that enhances the interaction with not only people but also with other smart objects. These are products, assets and other things embedded with processors, sensors, software and connectivity that allow data to be exchanged between the product and its environment, manufacturer, operator/user, and other products and systems. Connectivity also enables some capabilities of the product to exist outside the physical device, in what is known as the product cloud. The data collected from these products can then be analyzed to inform decision-making, enable operational efficiencies and continuously

improve the performance of the product. [51]

Definition 3.1.1 allows us to define the different cost components of the perception layer in more detail. As definition 3.1.1 states the smart object is the combination of hardware, software, and connectivity. Figure 3.2 gives an overview of the different components of these smart objects based on the work of Klubnikin [52] and interviews with people responsible for the creation of IoT applications in companies like Delaware, Philips and SAP. Figure 3.2 consists of four main branches. These branches each bring their own costs and are discussed in the remainder of this section.

The first branch is the import cost. When smart objects, or components of smart objects, are bought abroad (A lot of companies tend to buy many sensors in China due to economic reasons), it is possible an import cost needs to be paid. This import cost belongs to the CAPEX of the application as it must be paid only in the development phase of the application.

The second cost branch is the cost of the smart object itself. An organization can either choose to buy, rent or make the smart object. Buying or renting smart objects leads to a smaller startup cost. The only cost that is made is the cost of the smart object itself as sold/rented by the manufacturer. An organization does not need to worry about any technicalities of the smart object, only how to send the data to their platform layer. When the smart object is bought this cost belongs to the CAPEX. In case of renting, the cost is classified under OPEX. When an organization cares a lot about the KPIs for his IoT application, they will almost always prefer to make the smart object themselves instead of buying or renting. As in this case, the smart object is much more customizable. Notice that the "make" branch does not mean the organization needs to create the smart object themselves, they can also choose to let another manufacturer create the smart object for them. In both cases, some main cost components are introduced.

The cost of making smart objects is divided in two main categories. The first category is the cost needed to buy all the equipment of the smart object. This includes Sensors, hardware, a printed circuit board (PCB), Microcontroller (MCU), the casing of the smart object, the energy source and the actuators to react upon events. As most of these costs only have to be made once (per smart object) they all belong to the CAPEX of the application. The second category is given on the right hand side of the "make" branch in figure 3.2. These are the costs for assemblage and certification of the hard-

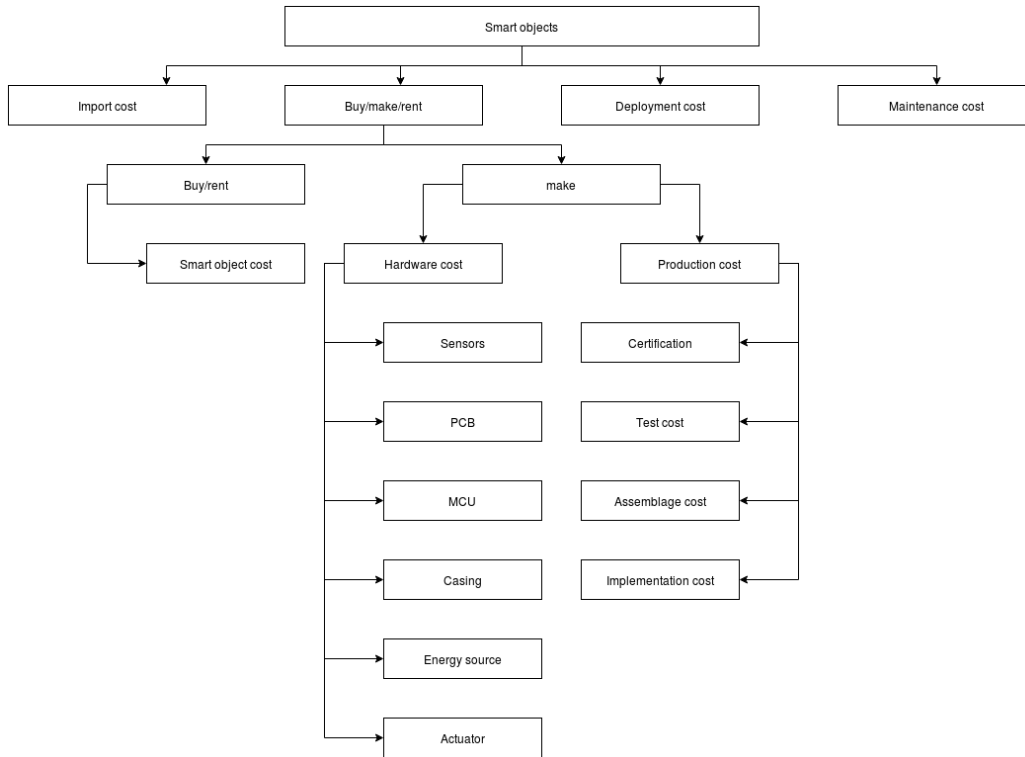


Figure 3.2: Overview of the components of the smart objects layer

ware, implementing and testing of the firmware. These costs also belong to the CAPEX of the IoT application as they are typical costs at the startup phase for any application.

The deployment cost includes the installation and roll-out cost for the smart object. The deployment cost belongs to the CAPEX of the application. When the application needs to be deployed more than once depending on the customer needs, the deployment cost can also belong to the OPEX of the application.

The last cost is the maintenance cost. This is the cost to keep the application up and running. The maintenance cost includes repairing components of the smart object, upgrading the firmware on the smart objects... The maintenance cost clearly belongs to the OPEX of the application.

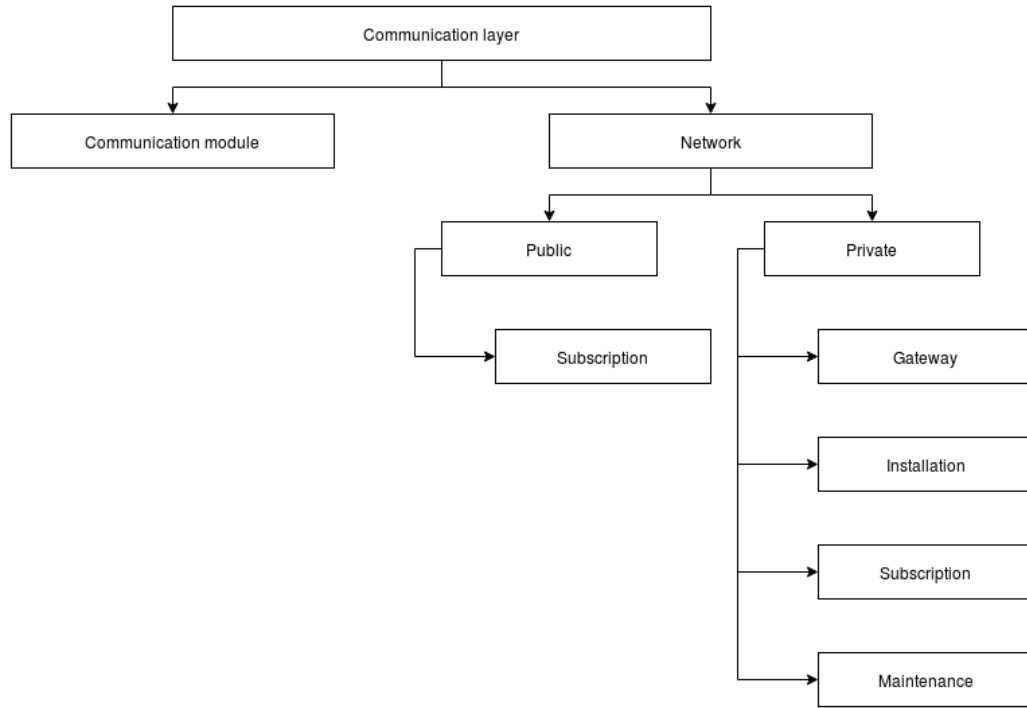


Figure 3.3: Overview of the components of the connectivity layer

### 3.1.2.2 Communication layer

The communication layer provides communication between the perception layer and the platform layer. This means it takes care of the data provided by the smart objects and sends this data to a server where it can be processed and/or stored. Figure 3.3 presents the cost components in the communication layer. Communication for IoT applications is possible using either public or private networks. With public networks, the end user does not have to provide gateways (GW) and installation of the network. On the other hand, public networks may not have coverage everywhere the user wants to install his IoT application. When this is the case, a private network is needed. The costs for having a private network are listed in figure 3.3. In this model the private network branch also includes a subscription cost. This is because even when an organization provides its own network coverage, it will still need a network operator to send its messages. Some open networks like TheThingsNetwork [53] exist for free but others are on paid subscription. The subscription and maintenance components belong to the OPEX of the application. The GW and installation components belong to the CAPEX.

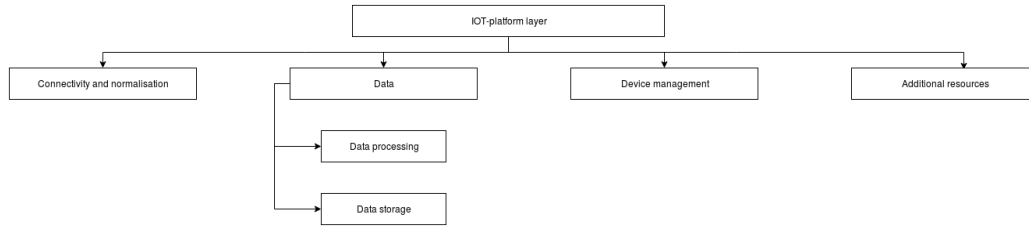


Figure 3.4: Overview of the components of the platform layer

### 3.1.2.3 Platform layer

The platform layer contains the data storage and data processing for the data received from the perception layer through the communication layer. The platform layer stores data from and metadata about the application. Device management of the different gateways enables the application to send uplink packets to the different nodes it wants to communicate with. A lot of big players offer off the shelf IoT platforms. Examples are Google Cloud, Microsoft Azure and Amazon Web Service (AWS). Figure 3.4 gives the different cost components of the platform layer. As these off the shelf platforms offer a pay-as-you-go service the cost components belong to the OPEX of the application. An organization can also choose to develop its own IoT platform which handles the different components of the platform layer. In this case an extra development cost should be added to the CAPEX of the application. As this is very rare, this cost component is neglected in figure 3.4. Notice that an organization is not forced to use one single platform. Although not recommended due to economic reasons - most platforms offer discount for IoT packages of services - an organization can choose to use different services from different platforms.

### 3.1.2.4 Application layer

The application layer is, as the name mentions, very specific for each end application. The layer is a connection of IoT technologies and sector professional technologies. Its key issue is to share the information produced by the smart objects and secure the information safety. It provides specific services to the end users through analyzed and processed data. For KPI definition and monitoring the application layer will mostly be some kind of dashboard where all KPIs are listed. Although this layer is very use case dependent,

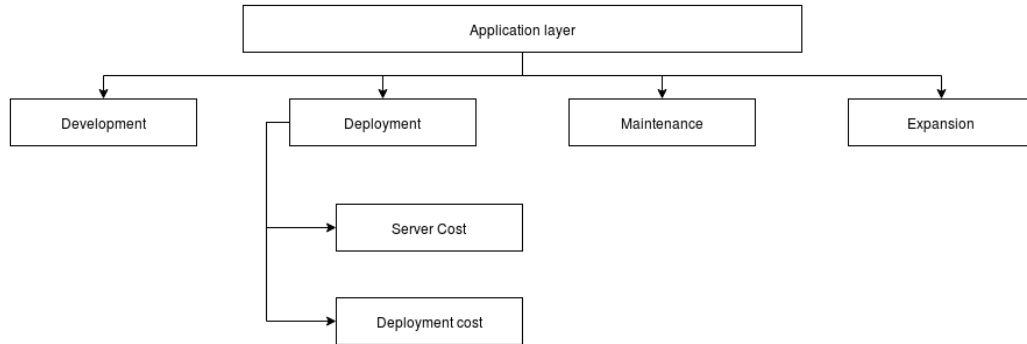


Figure 3.5: Overview of the components of the application layer

some main cost components can be discovered. Figure 3.5 lists these different components. The main cost component for the application layer is the development of the application. The organization can choose to either make the application themselves or to buy the software for the application from an IT company. In both cases the development cost of the application belongs to the CAPEX of the application. A second cost is the deployment of the application. For this a server (in the cloud or on a local machine) is needed if the application must be used by multiple users which is almost always the case. In the deployment component in figure 3.5 an extra deployment cost component is given. This is the cost for actually deploying the application. On application level this usually means using a git repository and a jenkins server to automate the deployment. As with the deployment cost in the perception layer, the deployment cost belongs to both the CAPEX and to the OPEX of the application. A third cost component of the application layer is the maintenance of the application. This includes bug fixing and code refactoring. As this cost must only be made after deployment (otherwise it is still counted as development cost) it is in the OPEX of the application. A last cost component which is actually not very connected to IoT-applications but more with software in general is expanding the application by adding new features. Yet, this cost is given in figure 3.5 for completeness. This expansion cost clearly belongs to the OPEX of the application.

### 3.1.3 Incremental Cost Structure

While the previous section listed the different costs for an IoT application this section focuses on the incremental cost analysis for adding KPIs to an already



deployed application. Figure 3.6 gives an overview of all costs described in the previous section. For each layer the costs introduced by adding a KPI to the IoT application are now identified. This section differentiates two types of costs for adding a KPI to the application.

- **Added cost:** The added cost is defined as the cost for an organization, introduced by adding a new KPI to the application after it is deployed. These are costs that one needs to make no matter in what stage they decide to add the KPI. So the cost stays the same for the decision of both adding the KPI in development phase or implementing the KPI in a later stage. In the visualisation of the models in the following sections, this cost is represented in green.
- **Incremental cost:** The incremental cost on the other hand is the cost that could have been avoided by implementing the KPI in the development phase of the application. This means adding a KPI after the deployment introduces a cost whereas this cost would have been avoided by adding the KPI in development phase. This cost is represented in red in the visualisation of the models in the following sections.

The following sections discuss if there is an added or incremental cost for each component in each layer.

#### 3.1.3.1 Perception layer

The first layer to examine is the perception layer. Implementing new KPIs introduces new data sources to measure these KPIs (given the new KPI can not be measured with the data sources already present in the perception layer). It may be possible to add a new data source from external sources such as an external API. This may not always be possible however and the need of adding an extra sensor may be present. When this is the case the smart object must be adapted. When the smart object is bought from a manufacturer this may introduce a problem as these are mostly not customizable. For this, a new smart object must be bought. This is clearly an incremental cost as the organization did now buy two smart objects compared to one object when the KPI was implemented in development phase. When the application is made by the organization, an added cost must be taken into account in the sensor cost component. It is important to realize this added cost would be

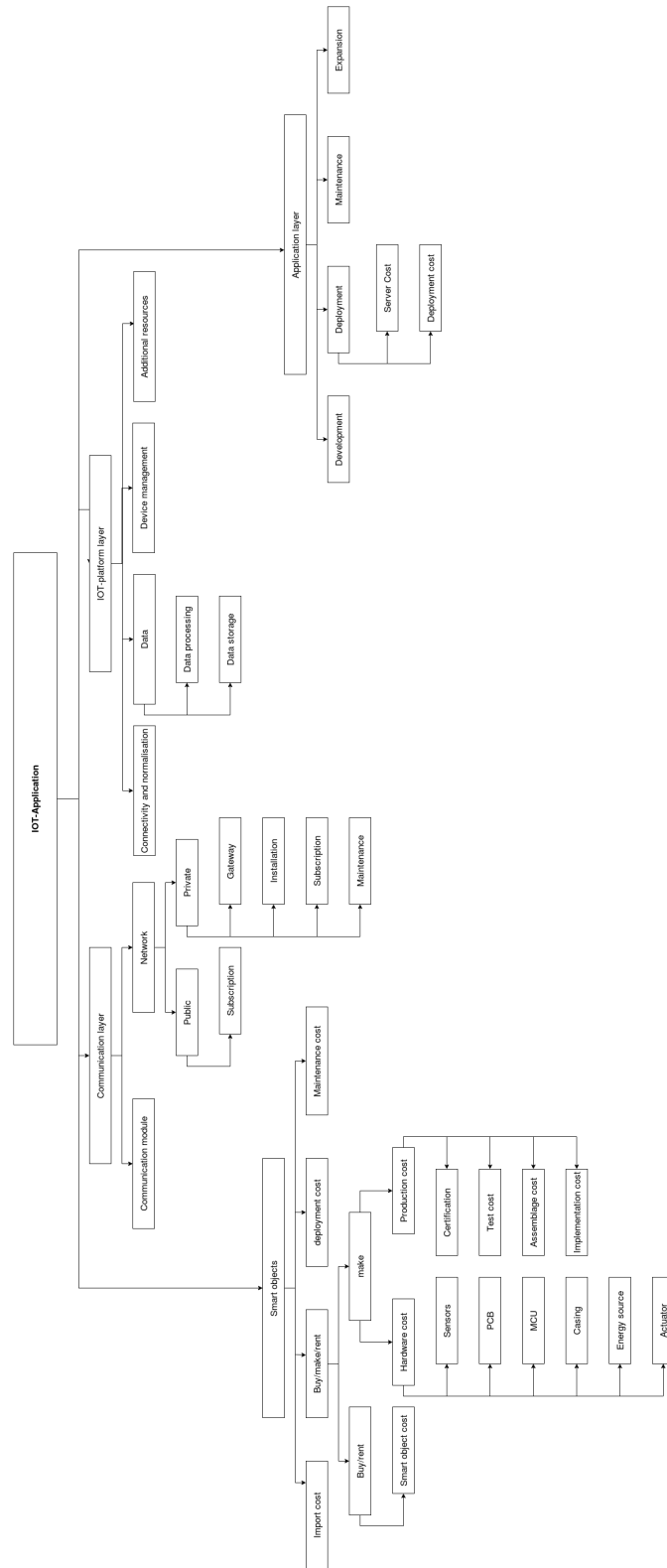


Figure 3.6: The total cost structure for an IOT-application as described in section 3.1

the same cost when the organization decided to implement the data source in development phase. Next to the added sensor cost the PCB needs to be redesigned. This cost is an incremental cost as the design of the PCB does now take place twice. Adding the KPI and thus the data source in development phase of the application would have avoided this cost. Most of the times the MCU for the smart object does not need to change when adding a new data source to the IoT application so this will not bring an incremental cost. When an extra data source is needed, the casing of the smart object may need to change. This certainly is an incremental cost as the design of the casing does now need to be done twice. The decision to redesign the case depends whether or not the data source must be integrated in the case (for example a light sensor) or if the data source does not fit the current casing. In these cases, the casing requires a redesign. The casing component is red in the model in figure 3.7 as an organization needs to think for itself if the casing needs to be adapted. When the new data source requires a lot of energy, the energy source may need to change. This leads to an incremental cost as the old energy source is now unnecessary. Of course it is possible that the energy source can be reused in other smart objects but that is for the organization to decide and out of the scope of this work.

Next to the cost for the hardware, an extra data source will also entail extra costs in the production process of smart objects. Adding a physical component to the smart object creates an incremental cost for the certification as this means the smart object must achieve a new certification. This way the certification cost, which is mostly a rather expensive cost, must be paid twice. The test cost brings an added cost because integration tests with the different other data sources should be taken into account. This cost however, when designed well, is the same as the test cost for adding the data source in the development phase. Adding a new data source that can perfectly integrate with the already existing smart object will create an added cost for both the assemblage and implementation cost. Note this remark only counts when the application is well designed from the start with attention to extensibility. The last cost component that changes, is the deployment cost for the application. When the existing smart objects do not need to be upgraded, for example when the new smart object with the extra data sources is just a newer version of the old smart object, this does not provide an extra cost. When this is not the case and every existing smart object of the application needs to implement the new data source, this needs to be done twice and thus ensures an incremental cost. When this last scenario is the case, an extra incremental logistic cost must be taken into account to retrieve all smart objects and upgrade them. In figure 3.7 this is called the

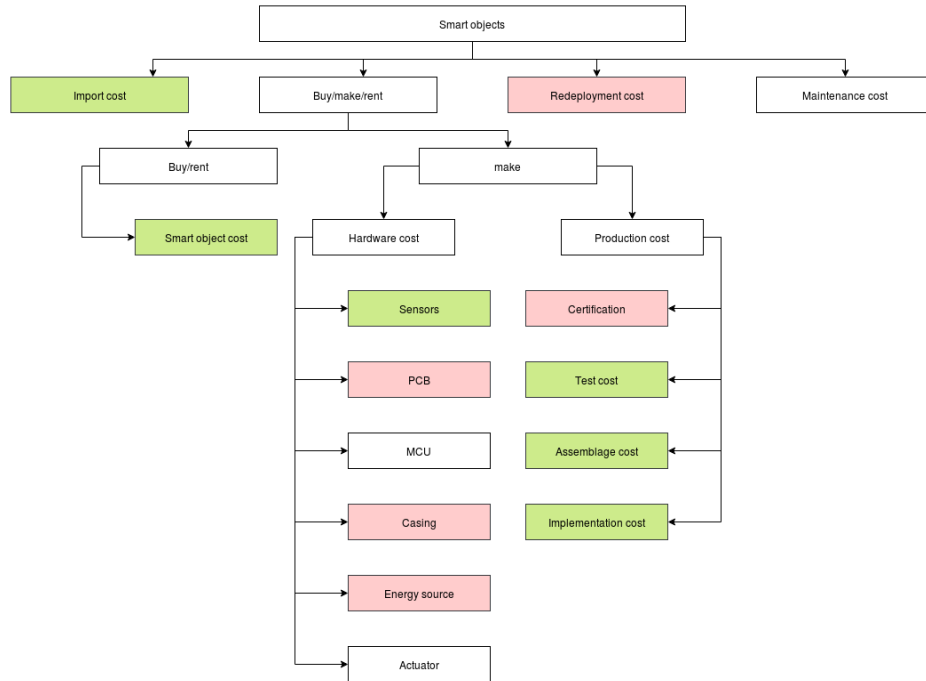


Figure 3.7: Added and incremental cost for smart objects

redeployment cost and will usually be even larger than the first deployment cost.

Figure 3.7 gives a visualisation of the above described added and incremental costs.

### 3.1.3.2 Communication layer

More data sources will cause more communication. For the communication layer this means the subscription with our network provider may change. A new KPI may need more bits to send its relevant data or may need to send more frequently in order to properly measure the KPI. An example of this last requirement could be when the end user decides to implement a heartbeat in his application to measure the uptime as a KPI. This is an added cost as almost all providers allow to simply upgrade the subscription to the new needs of the organization. The gateway, installation and maintenance cost will not be affected by adding a new KPI to the application.

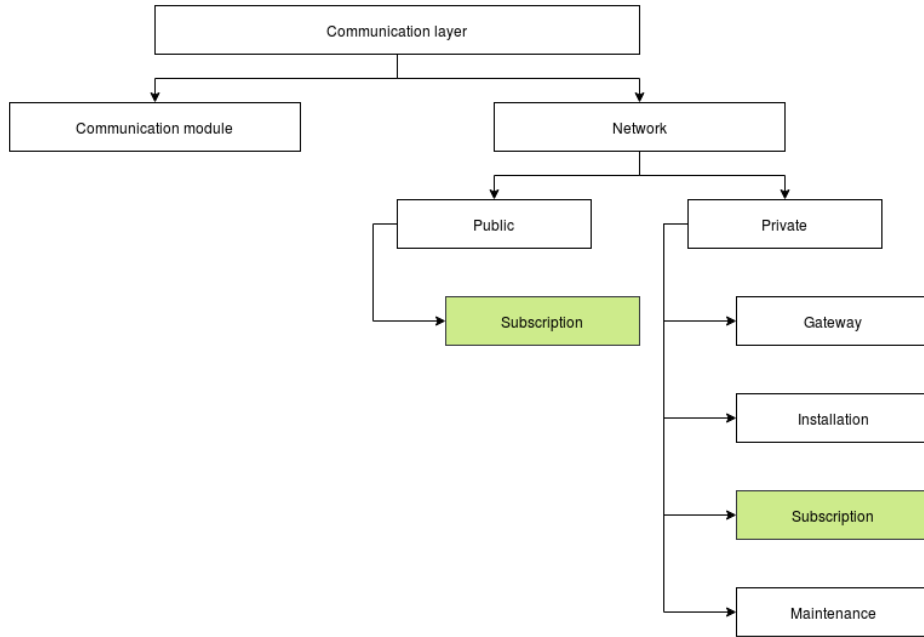


Figure 3.8: Added and incremental cost for communication layer

### 3.1.3.3 Platform layer

The platform layer is affected by the addition of a new KPI in several ways. First of all the database design needs to be rethought. When the database is properly designed in development phase this should not be a huge effort. This means it introduces an added and no incremental cost because it will not require a much larger effort compared to when the KPI was implemented in development phase. When the data from the new KPIs must be preprocessed this is also an added cost for the cost model. The device management cost component will not change when new KPIs are added to the application. So won't the connectivity and normalisation component. As the additional resources are very use case specific, there is no telling if this component will change by adding new KPIs over time. It is therefore neglected in this work.

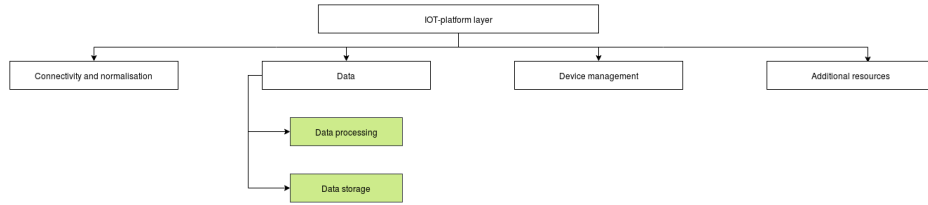


Figure 3.9: Added and incremental cost for the platform layer

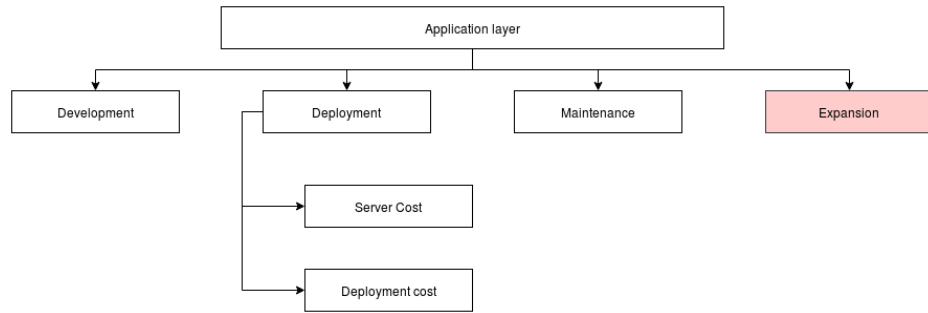


Figure 3.10: Added and incremental cost for the application layer

### 3.1.3.4 Application layer

Adding KPIs to the application will add an extra cost to visualize the new data sources in the application layer. This cost will be an incremental cost compared to the cost for implementing the KPI in development phase. This incremental cost is classified in the expansion component of the cost model. The reason this is an incremental and not an added cost is due to the nature of software development. Extending software requires a much bigger effort than implementing the new feature in development phase [54].

### 3.1.3.5 Final model

Figure 3.11 presents the final combined model for the four layers as described above. Organizations can now use this cost model to validate if it is worth adding extra data sources in development phase. Doing this increases the risk these data sources will never be used. Yet, the benefits that when they are used and thus the incremental costs are avoided, can be very high. The green components in the model indicate an extra cost when adding KPIs. This cost is the same as the cost compared to when the KPI would have

been included in development phase. The green components are meant to inform organizations what costs they should take into account when adding new KPIs. The red components on the other hand, stand for an incremental cost. This means missing a KPI will have a bad impact on the costs of these components. The size of the impact depends on the scale on which the application is deployed, the task of the application, the domain the application is operating in etc. It is the task of the organization to estimate this impact based on its application knowledge. The white components are cost components that should not change when adding KPIs to the IoT application. This does not mean these components can never change if a KPI is added, they are just very unlikely to change. This final model should inform organizations about the cost of creating an IoT application and what the consequences are when they miss a KPI in an IoT application after deployment. Section 3.2 provides a methodology and a KPI suggestion tool to suggest useful KPIs to an end user. The outcome of this tool is used together with this incremental cost model to make a more considered decision whether or not to implement the KPI in development phase.

### 3.1.4 Future Improvements

The above described models list the main incremental and added cost components of an IoT-application for adding KPIs. The models indicate the different cost components of an IoT application and illustrate which costs to expect when adding KPIs to these applications. The models however, do not give any clues about how large these costs are or which components are the largest cost drivers. With the current model, an organization can identify these costs but needs to estimate the size of this cost for itself. As a first improvement the different costs and their magnitude should be listed and described. This way organizations can, instead of only identifying the cost based on these models, also predict very precise how much missing a KPI for an IoT application will cost and which are the main cost drivers.

In addition, the costs for adding KPIs to an IoT application will not have the same magnitude in every phase of the creation process (displayed in figure 1.1). As shortly mentioned in the introduction of this section, the cost for adding a KPI in the prototyping phase will normally be much lower compared to the cost of adding the same KPI after the scaling phase. Future work could give an idea about what the magnitude of these costs are for every phase of the creation process.

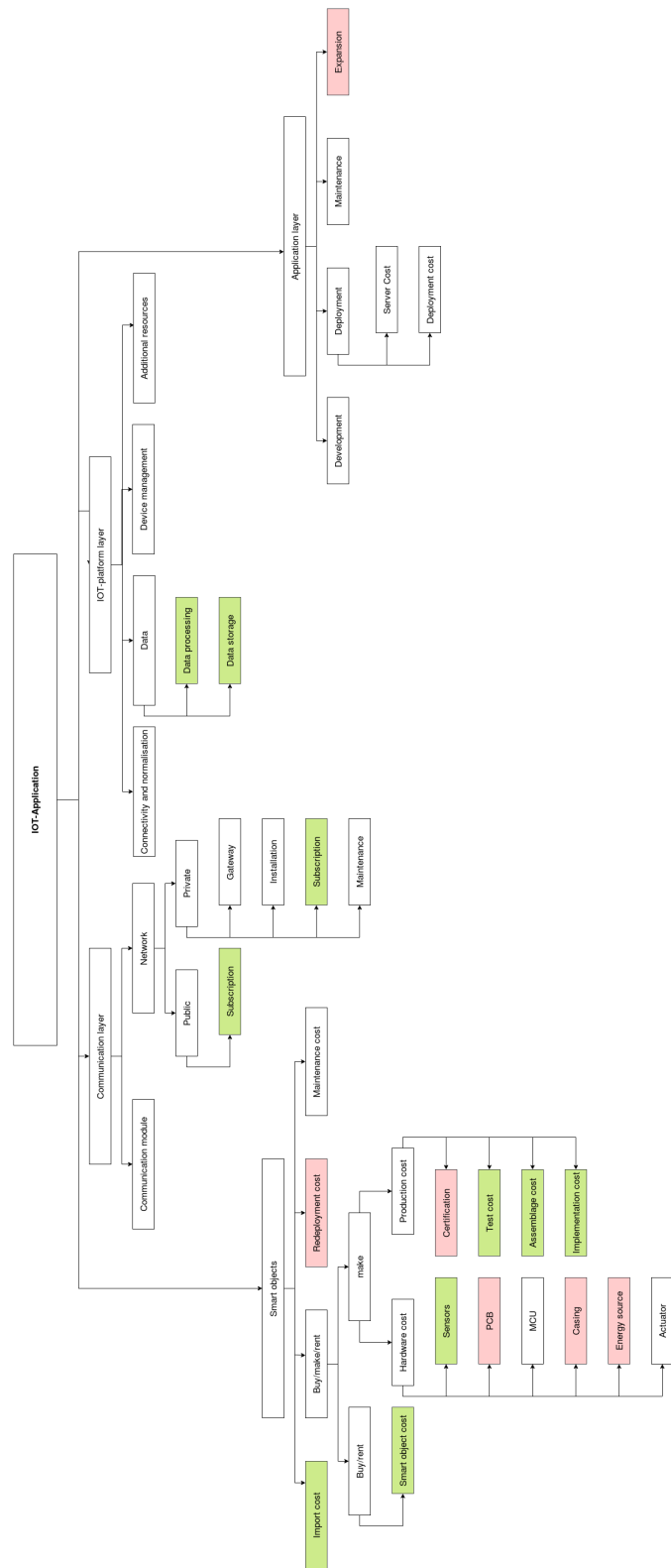


Figure 3.11: Final incremental cost model for all layers



## 3.2 KPI Suggestion Tool

### 3.2.1 Introduction

This section handles the methodology for the KPI suggestion tool. This tool takes user input in the form of non-functional requirements. Non-functional requirements are defined as requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviors. Examples are: security, availability, stability... While functional requirements are determined by the business rules and domain of the application, an example of a functional requirement for an application would be: An administrator should be able to create new customers. Notice in the remainder of this work whenever the term requirements is used, these are non-functional requirements. Except of course when it is specifically stated that functional requirements are meant.

Based on the non-functional requirements of an IoT application, the tool suggests useful KPIs to an end user. The tool should be used by organizations who want to discover new KPIs they did not think about at first. This in order to avoid the incremental costs described in section 3.1. The architectural design of the tool is given in figure 3.12. The methodology consists of 6 main parts. Detailed information about each part is given in the next sections.

The first part of the tool is the user input in the lower left corner of the figure. Here, the user estimates the non-functional requirements of his application on a likert scale (a rating scale). These non-functional requirements are then processed using a correlation matrix. This correlation matrix consists of predefined correlation scores between each requirement-KPI pair (see section 3.2.2.3). The correlation matrix gives a mapping between the non-functional requirements and the KPIs and tells how important each requirement is for each KPI. When combined, the user input (in the form of non-functional requirements) and the correlation matrix provide a first score for each KPI. This way the application discovers a list of top suggested KPIs for the application.

The second part of the suggestion tool is the KPI net. This KPI net is a relational database where all data is stored. It contains each KPI, the data sources for every KPI, the non-functional requirements described above and

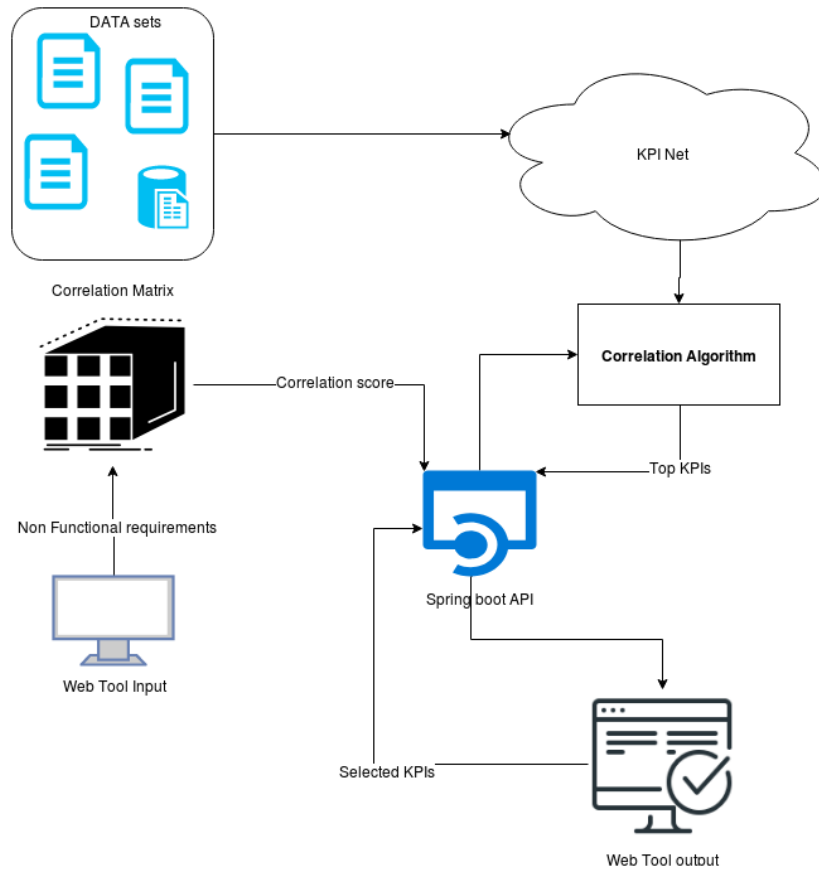


Figure 3.12: Schematic overview of methodology for the KPI suggestion tool

the correlations between these requirements and the KPIs.

The KPI net and the first list of top suggested KPIs (as described above) are both sent to the back-end API. Based on these list of top KPIs, the back-end searches for other suggested KPIs which have a lot of common data sources with the list of top KPIs. These other suggested KPIs are called the "low hanging fruit" KPIs as they require minimal effort by the end user to implement. The back-end also provides API endpoints to add KPIs, requirements and correlations to the KPI net.

The result of the back-end, this is the list of top KPIs and the suggested low hanging fruit KPIs, is sent to the user as output of the tool. The user can now select which KPIs he wants to implement. Based on this selection the back-end selects new suggested KPIs. This is an iterative process that continues until the user is satisfied with the current suggested KPIs.

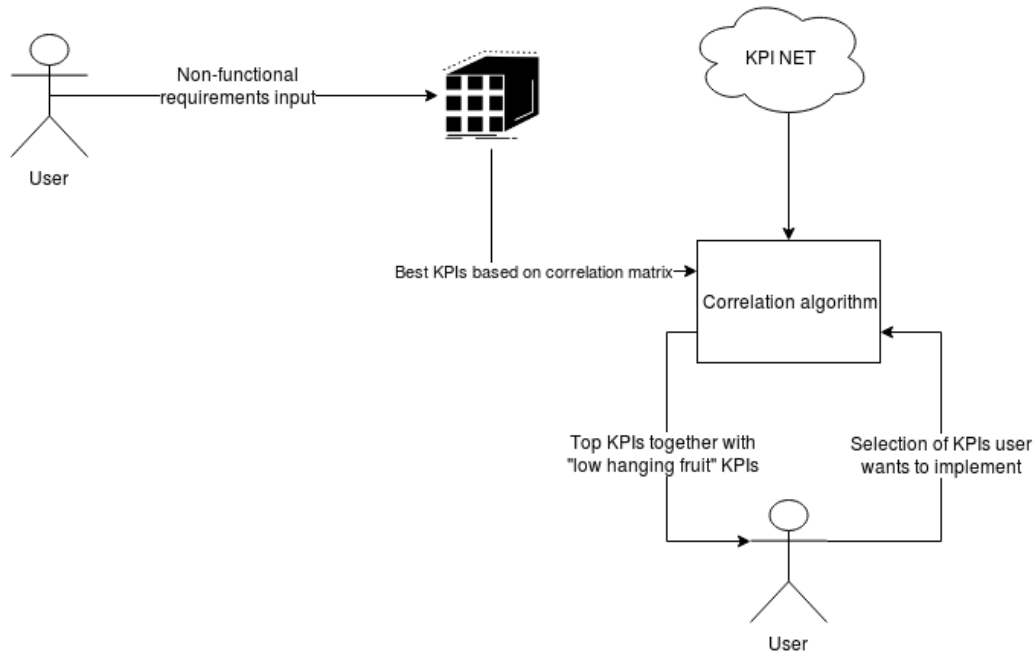


Figure 3.13: A typical user story to use the KPI suggestion tool

Figure 3.13 gives a typical user story of how the application is used. The user rates his non-functional requirements first. These non-functional features are rated on a likert scale from 1 to 7 and define the importance of every non-functional requirement present in the KPI net. When the requirements are rated, the correlation matrix will select the best KPIs based on this input. The algorithm returns these KPIs together with the needed data sources for every KPI. This data goes through the correlation algorithm (present in the back-end of the application). This algorithm uses the data stored in the KPI net to select other KPIs with the lowest cost based on the list of top KPIs. The user can then look at this result and select which KPIs and thus data sources he will use after which the correlation algorithm checks for new discovered lowest cost KPIs. This last iteration is repeated till the user decides he has the best set of KPIs.

In the following sections, the implementation and working of each of the different parts of the application is described in more detail. In the end some ideas for future improvements of the KPI suggestion tool are listed.


## 3.2.2 Components

### 3.2.2.1 User input front-end

The first part of the web tool takes non-functional requirements about the application as user input. It tries to find the most useful KPIs based on this input. Because it is impossible to ask the user to rate all functional requirements, the choice was made to focus on non-functional requirements. These are more general and more easy to predict for the end user. All non-functional requirements are stored in the KPI net (see section 3.2.2.2). A description for each requirement is also provided in the front-end user input page. This way, there is no doubt about the definition of a non-functional requirement. In total the application contains 15 non-functional features (listed in appendix B). The API of the suggestion tool provides an endpoint that allows to add more non-functional requirements for future improvements (see appendix D). Figure 3.14 gives an example of what the front-end user input web page looks like. The end user rates every non-functional requirement for his application on a likert scale from 1 to 7. During the development of the application the idea rose that it could be useful to provide meaningful labels instead of just the numbers 1 to 7. These meaningful labels could provide more context to the end users of the tool. For security for example, such labels could go from basic security to bank level security in multiple levels. Another example could be the availability that is usually expressed in number of nines. The scale then goes from 99% availability to 99.999999% availability. In the end, this was not adopted because not all applications will use the same metrics to define the importance of a requirement. An organization may for example value the measurement of availability very important (score 7 on the likert scale) while it only needs 99% availability for its application. This is not a contradiction and is hard to determine when working with meaningful labels. Every user gets a fixed number of points to distribute among the given features. Users can only submit their input when this number of points is greater or equal than zero, this to force the user to think about which requirements are really important for his application.

The front-end application asks end users to rate a fixed set of non-functional requirements on a likert scale from 1 to 7. This to indicate how important each non-functional requirement is for the application. These scores are processed later on in the correlation matrix (section 3.2.2.3) to find the list of top KPIs.

**Welcome to the KPI Selection tool!**



Indicate for each non-functional requirement how important this requirement is for your application.

You have 1 points remaining

Disaster recovery	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Availability	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Fault tolerance	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Maintainability	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Performance	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Response time	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Scalability	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Security	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Robustness	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>

Clear

Figure 3.14: Example of the front-end user input

### 3.2.2.2 KPI net

The KPI net is in fact a relational mySQL database. The architectural design of the database is given in figure 3.15. The remainder of this section discusses the different tables and their columns.

The KPI table contains, as the name states, all KPIs of the application. In total 46 KPIs were found in different sources (see chapter 2). Appendix B lists these KPIs. As mentioned above an API endpoint in the back-end allows for future additions of KPIs. The direction column in this table defines which is the best value for the KPI. This can either be "min", "max", "optimal value", or "undefined". The KPI machine downtime for example will have "min" as direction while the KPI signal strength will have "max" as value for the direction column. The unit column describes which unit the KPI is measured in. Examples of units are percentages, absolute value, time. . . These last two columns are mainly for clarification reasons for the end users.

The requirements table contains all requirements for the application. The requirements, together with their description, that are now in the database were found in different sources [55–60] that describe best practices and most common use cases for these non functional requirements. Each requirement is linked to each KPI in the requirement\_correlation table. A correlation score tells the application how important a non-functional requirement is for a KPI.

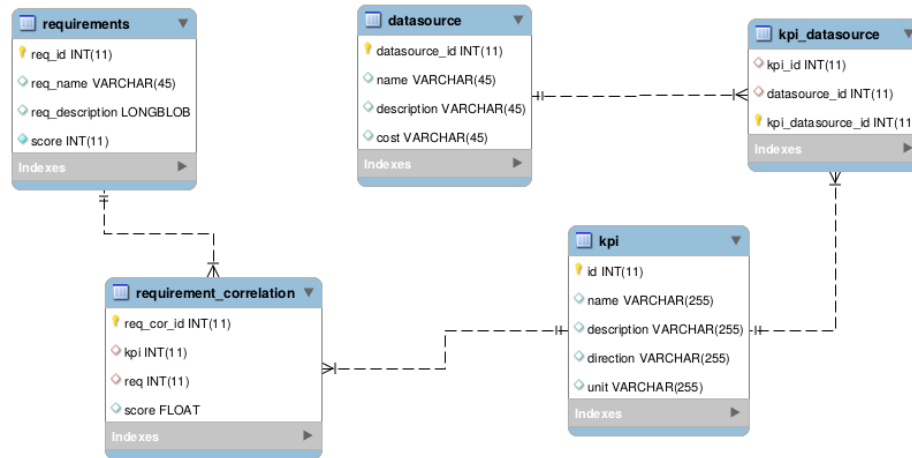


Figure 3.15: The EER model of the KPI net

The table contains an entry for each requirement-KPI pair. The correlation for the requirement "availability" and the KPI "% of device up" for example will be very high whereas the correlation between that same requirement and the KPI "Air quality" will be rather low. This table is the backbone table for the correlation matrix (section 3.2.2.3). Section 3.2.2.3 gives more info about how the correlation scores between KPIs and requirements are determined.

The data source table makes sure all data sources in the database are defined unambiguous. The link to capture which KPIs uses which data sources is defined in the kpi\_datasource table.

The KPI net contains all data the suggestion tool uses in order to suggest useful KPIs to the end user. These suggestions are based on the input the tool receives from this end user.

### 3.2.2.3 Correlation matrix

The correlation matrix is a very important part for the web tool as it transforms the non-functional requirements, received from the user input, into a score for each KPI. The correlation matrix receives the scores for each non-functional requirement as given by the user on the input page. Although the correlation matrix is implemented in the back-end of the application, it is given as a separate component in figure 3.12 due to its importance for the

suggestion tool. Listing 3.1 gives an example input the correlation matrix receives from the front-end. This input is in JSON format and is converted in the back-end by using the data stored in the KPI net mentioned above.

Listing 3.1: example of json input for correlation matrix

---

```

1  [
2    {
3      "id": 1,
4      "name": "Security",
5      "description": "Security is freedom from, or
                       resilience against, potential harm caused by
                       others. Beneficiaries (technically referents)
                       of security may be of persons and social
                       groups, objects and institutions, ecosystems
                       or any other entity or phenomenon vulnerable
                       to unwanted change.",
6      "score": 2
7    },
8    {
9      "id": 2,
10     "name": "availability",
11     "description": "The degree to which a system is
                     in a specified operable and committable state
                     at the start of a mission, when the mission
                     is called for at an unknown, i.e. a random,
                     time. Simply put, availability is the
                     proportion of time a system is in a
                     functioning condition",
12     "score": 3
13   },
14   ...
15   {
16     "id": 15,
17     "name": "Throughput",
18     "description": "In general terms, throughput is
                     the rate of production or the rate at which
                     something is processed.",
19     "score": 5
20   }
21 ]

```

---

Algorithm 1 gives the algorithm used to transform the input into KPI scores in pseudo code. In the algorithm the variable "kdao" is responsible for retrieving all KPIs from the KPI net. The "cdao" object retrieves all correlation coefficients stored in the KPI net. The correlation coefficients are determined using the following methodology: at the creation of the tool, every KPI-requirement pair received a score on 5 indicating how important the requirement is for this KPI. These scores are based on a large amount of different sources. We call this score the importance score. These importance scores are then normalized so every KPI-requirement pair received a score between 0 and 1. This normalization is done to make sure the relative weight for the KPI-requirement is used. This way KPIs with less obvious requirements can also be selected. The formula to calculate the correlation coefficient between requirement  $r$  and KPI  $k$  as described above is given in formula 3.1.

$$\rho(k, r) = \frac{score(k, r)}{\sum_{\delta \in \Omega} score(k, \delta)} \quad (3.1)$$

With

$\rho(k, r)$  : Correlation coefficient between KPI  $k$  and requirement  $r$   
 $score(k, r)$  : The importance between KPI  $k$  and requirement  $r$   
 $\Omega$  : The collection of all requirements

As each requirement is rated on a Likert scale from 1 to 7 (see section 3.2.2.1), the score for each KPI is then calculated according to formula 3.2.

$$S_k = \sum_{r \in \Omega} C(r, k) * I(r) \quad (3.2)$$

With

$S_k$  : The score for KPI  $k$   
 $\Omega$  : The collection of all requirements stored in the KPI net  
 $C(r, k)$  : The correlation score between requirement  $r$  and KPI  $k$   
 $I(r)$  : The input score for requirement  $r$  as rated by the user

Figure 3.16 gives a conceptual overview of how the requirement scores, as provided by the end user, are transformed into the output which contains a score for every KPI. After this transformation, all KPIs have a score based on the non-functional requirements. The back-end of the application will now search for other suggested KPIs based on the KPIs that have the best scores.



---

**Algorithm 1** Code responsible for the transformation of the user input in the correlation matrix

---

```

1: List<KPI> kpis = kdao.findAll()
2: for KPI k in kpis do
3:   score = 0
4:   for Requirement r in requirements do
5:     RequirementCorrelation rc = this.cdao.findByReqAndKpi(r,k)
6:     if rc != null then
7:       score += rc.getCorrelationScore * r.getScore()
8:     end if
9:   end for
10:  # Normalize score for every KPI
11:  score /= requirements.totalScore(k)
12:  k.setScore(score)
13: end for
14: return kpis

```

---

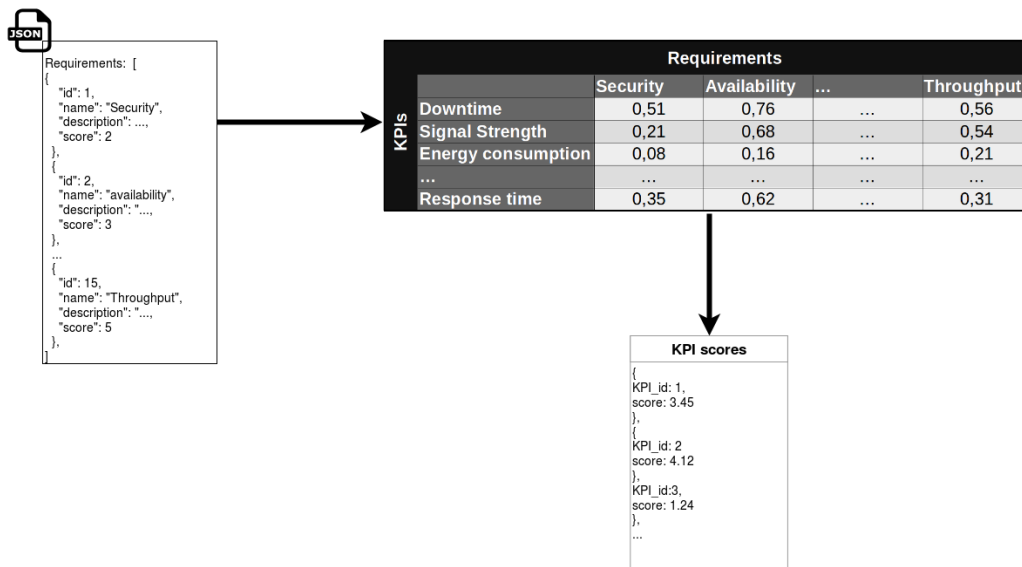


Figure 3.16: Conceptual overview of the working of the correlation matrix

### 3.2.2.4 Back-end of the application

The back-end of the KPI suggestion tool is a stand-alone application. This way other applications can use the stored KPIs in the future. An Application Programming Interface (API) provides the stored KPIs together with the distances between these KPIs. As mentioned before, the back-end also provides endpoints to expand the application. This allows to add KPIs, requirements and data sources for future improvements. The back-end is written in java code. It has a three layer architecture (see figure 3.17). The first layer is the data layer. This layer enables communication with the KPI net and converts the data stored in the KPI net to java entities. These entities are represented in the models package in the class diagram (displayed in figure 3.18). Every entity inherits from the `AbstractEntity` class in order to make an abstraction of entities for future improvements and extensions. The data layer also contains a Data Access Object (DAO) package which provides an abstract interface to connect with the database itself. The back-end uses the hibernate framework as ORM library [61] to make connection to the database. This framework allows us to make an abstraction of the complex queries and focus on the data. This way the classes in the DAO package are really simple. The logical layer is responsible for the suggestion of the KPIs. This layer is the engine of the application. It takes user input from the presentation layer and processes this input with the data from the data layer. In the logical layer, all code concerning KPI suggestion is present. It includes the services to process the requirement scores (based on the correlation matrix), get the list of top suggested KPIs as will be explained in the next section and calculates the alternative suggested KPIs (also explained in the next section). The logical layer also provides a comparator which allows to rank the KPIs according to their score. The presentation layer, at last, offers the API endpoints to the front-end application. These endpoints enable the communication between the back-end and the front-end application. They also allow users to add and alter KPIs, non-functional requirements and data sources for the KPI net. The data in the KPI net can be exposed to third parties using this presentation layer. The presentation layer also includes the data transfer objects (DTOs). These DTOs are simple representations of the entities in the data layer. Mappings between these DTOs and the entities enable the opportunity to modify the data layer without the need to change the presentation layer and thus the front-end application. The DTOs also provide abstraction of the back-end entities for the front-end. This three layer architecture offers the possibility to separate the logic to predict KPIs from the database objects and the API. It also allows to change the structure of

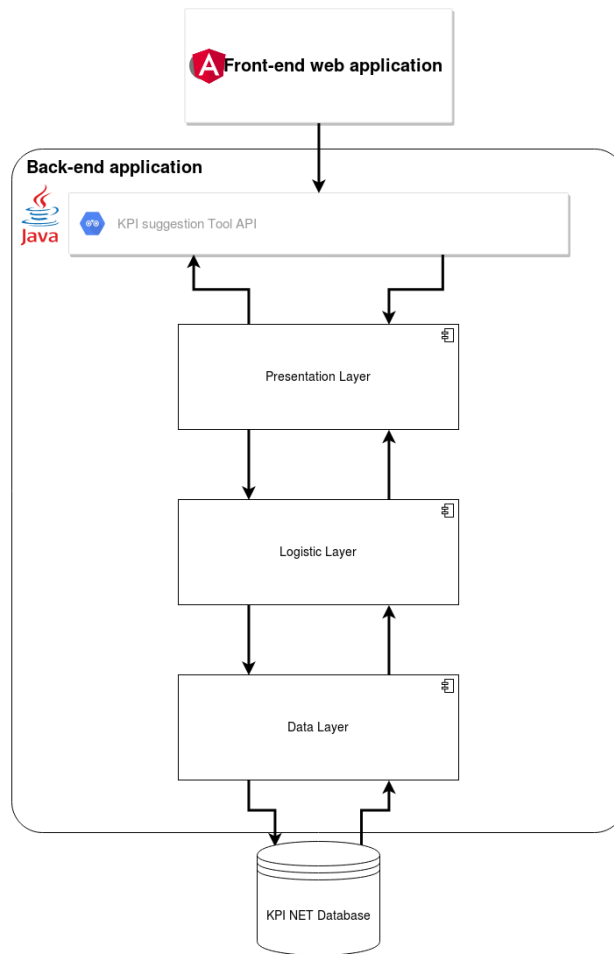


Figure 3.17: The three layer architecture for the back-end application

the KPI net without the need to rewrite all code concerning the KPI suggestions. A schematic overview of the three layer architecture is given in figure 3.17, while figure 3.18 shows the class diagram for the back-end. Notice the different fields and methods are neglected in this figure as its main purpose is to give a better understanding of the architecture and dependencies of the back-end application and not to perfectly visualize the UML diagram of the application.

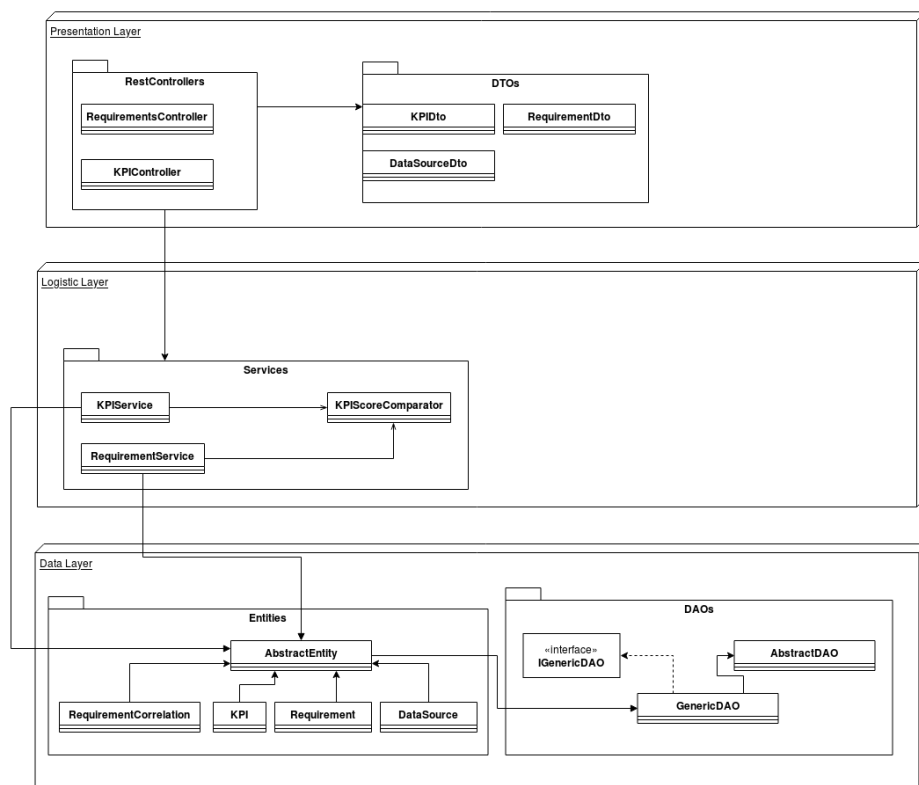


Figure 3.18: Class diagram for the back-end

### 3.2.2.5 KPI calculation algorithm

As mentioned above, all code concerning the calculation of the suggested KPIs is contained in the logical layer of the back-end application. Algorithm 2 contains the pseudo code for the selection of the top KPIs. The algorithm makes use of a `treeSet` which is an ordered set. The `kpiList` variable is the output of the correlation matrix as described in section 3.2.2.3. The score a KPI can achieve lies between 1 and 7. This because the correlation matrix only contains correlation coefficients between 0 and 1 and because the maximal input score for a non-functional requirement is 7. Taking the product of these two values can thus only generate values scores between 1 and 7. A threshold with a value of 3.8 is defined based on several test iterations in order for a KPI to be selected as a top KPI for the application. Notice this threshold is based on the current content of the KPI net. It is possible a better threshold value can be found in future improvements of the algorithm. If no KPI achieves the threshold value, the algorithm returns the 'x' KPIs with the best score (x being a fixed variable, in this case 3). Other KPIs are suggested based on this selected top KPIs. Algorithm 3 provides the pseudo code for the suggestion of KPIs. The algorithm checks how many data sources are needed for each KPI to add them to the application. The cost to add a KPI to the application is then represented by this amount of data sources divided by the score of the KPI for the application. Both components are multiplied with a weight coefficient. This weight coefficient enables the possibility to give more importance to the cost of the KPI (in this case, the number of common data sources). The cost gets divided by the score to prevent the tool from suggesting KPIs that have a lot of data sources in common but are no use for the application. To prevent the suggestion of KPIs that need no extra data sources (cost= 0) but are totally irrelevant, formula 3.3 is used in the algorithm.

$$R_c(k) = \frac{a(1 + c(k))}{b * S_k} \quad (3.3)$$

With

- $R_c(k)$  : The relative cost of KPI k
- $c(k)$  : The cost of the KPI
- $S_k$  : The score of KPI k based on the requirements
- $a$  : The weight coefficient for the cost
- $b$  : The weight coefficient for the score

Notice the formula uses  $1 + c(KPI)$  in the numerator in order to always retrieve a cost larger than zero. If this would not be used, total irrelevant KPIs that do not need extra data sources will always be suggested (as they would always have a relative cost of 0). Based on the relative cost described in equation 3.3, the KPIs with the lowest relative cost are selected as suggested KPIs.

The algorithm finally returns the list of top KPIs first selected by using the correlation matrix, together with the suggested KPIs with the lowest relative cost. The amount of suggested results is a fixed variable. No threshold was used here as the relative score of the KPIs can be very different depending on the application and top KPIs. A Normalization factor could be used to keep the score between 0 and 1 but by testing the application, we noticed even then the relative score is very dependent on the application. Listing 3.2 gives an example of the output the algorithm returns to the front-end application.

Listing 3.2: example of json output sent to the front-end

---

```

1 {
2   top : [
3     {
4       "id": 9,
5       "name": "Machine downtime",
6       "description": "Security is freedom from, or
                        resilience against, potential harm
                        caused by others. Beneficiaries (
                        technically referents) of security may be
                        of persons and social groups, objects
                        and institutions, ecosystems or any other
                        entity or phenomenon vulnerable to u",
7       "score": 4.17,
8       "rel_cost": 0,
9       "selected": false,
10      "data_sources_id": [8,11,13]
11    },
12    {
13      "id": 16,
14      "name": "Active users",
15      "description": "Number of users that are
                      using the application at the moment",
16      "score": 3.98,
```

```

17         "rel_cost": 0,
18         "selected": false,
19         "data_sources_id": [6]
20     },
21     ...
22 ],
23 suggested: [
24     {
25         "id": 7,
26         "name": "Signal Strength",
27         "description": "received signal strength
28                         indicator is a measurement of the power
29                         present in a received radio signal.",
30         "score": 2.87,
31         "rel_cost": 0.34843,
32         "selected": false,
33         "data_sources_id": [6]
34     },
35     {
36         "id": 31,
37         "name": "% of devices up",
38         "description": "The fraction of devices
39                         compared to all devices that are actually
40                         up and running",
41         "score": 1.71,
42         "rel_cost": 0.58479,
43         "selected": false,
44         "data_sources_id": [8]
45     },
46     ...
47 ]
48 }

```

---

### 3.2.2.6 User output frontend

Listing 3.2 gives an example of the JSON code the front-end takes as input to display the results of the KPI suggestion tool. 3.19 shows a screenshot of this front-end output. The end user sees all suggested KPIs and can select which

---

**Algorithm 2** Pseudocode for retrieving the top KPIs based on the user's non-functional requirement input

---

```

1: x = 3
2: Set<KPI> ts = new TreeSet<>(kpiList)
3: Set<KPI> result = new TreeSet<>()
4: KPI best = ts.pollLast()
5: result.add(best)
6: # If no KPI has a higher score than the min. threshold, return fixed
   number of best KPIs
7: if best.getScore() <= threshold then
8:   for i = 0; i < x; i++ do
9:     result.add(ts.pollLast())
10:  end for
11: else
12:   while ts.last().getScore() > threshold do
13:     result.add(ts.pollLast())
14:   end while
15: end if
16: return result

```

---



---

**Algorithm 3** Pseudocode for the KPI suggestion algorithm

---

```

1: List<DataSource> sources = new ArrayList<>()
2: Set<KPI> ts = new TreeSet<>()
3: ts.useComparator(KPIRelativeScoreComparator)
4: for KPI k in selected do
5:   sources.addAll(k.getSources())
6: end for
7: for KPI k in kpiList do
8:   cost = 0
9:   for DataSource ds in k.getSources() do
10:    if ds not in sources then
11:      cost++
12:    end if
13:   end for
14:   float relativeScore = a*(1+cost)/b*k.getScore();
15:   k.setRelativeScore(relativeScore);
16:   ts.add(k);
17: end for
18: return ts

```

---



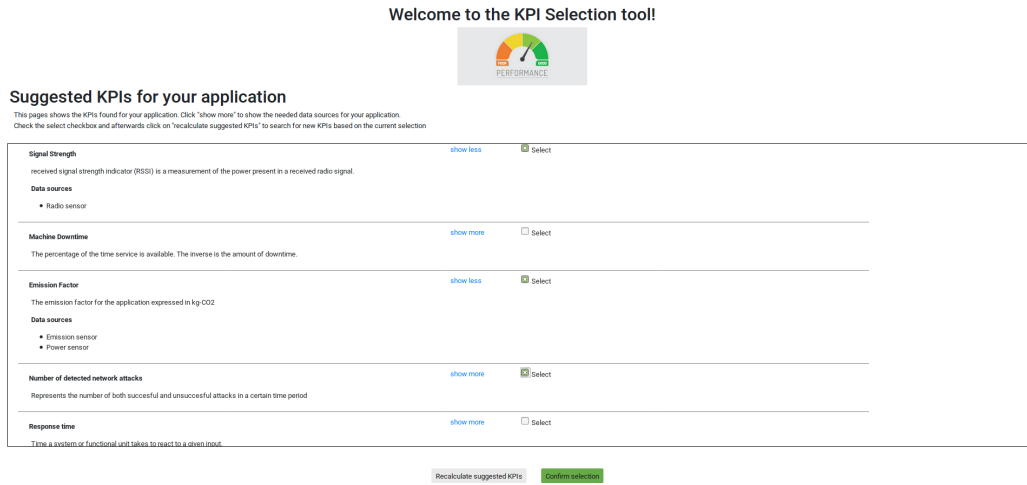


Figure 3.19: A screenshot of the front-end output the user gets to see

KPIs he wants to implement. Based on this selection, the back-end can run the algorithm again. This to obtain more suggested KPIs. The algorithm now uses the KPIs the user selected as the list of top KPIs (as described in section 2.4.2) as new starting point. The user repeats this process until he is satisfied with the final result.

### 3.2.3 Future improvements

The KPI suggestion tool as described above is a first implementation. Although this implementation gives results that make sense and are useful as described in the validation chapter (chapter 4), some improvements can be made to give even better results in the future. The improvements have not been implemented because of a lack of data sources. It turned out in the construction of this work that, although many companies were very enthusiastic about the creation of a KPI suggestion tool, not many were willing to give the KPIs they were using for their applications. This section handles the improvements that should be made when more data about KPIs is available.

### 3.2.3.1 KPI clustering

A first improvement is touched shortly in section 3.2.1. The first idea was to use data sheets generated by KPIs to define the distance between KPIs. XingYu et al. [5], proposes such an approach for cellular networks. Based on data analysis on his results, he noticed his approach can very well divide data into clusters and each cluster can effectively reflect the relationship between two KPIs. His methodology consists of two main steps (displayed in figure 3.20):

**1. Define correlations between each two KPIs:** The first step of XingYu's approach is to use the data sets of each two KPIs and define a correlation coefficient between them based on a Pearson product moment correlation coefficient (eq. 3.4).

$$\rho_{X,Y} = \frac{COV(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \quad (3.4)$$

With

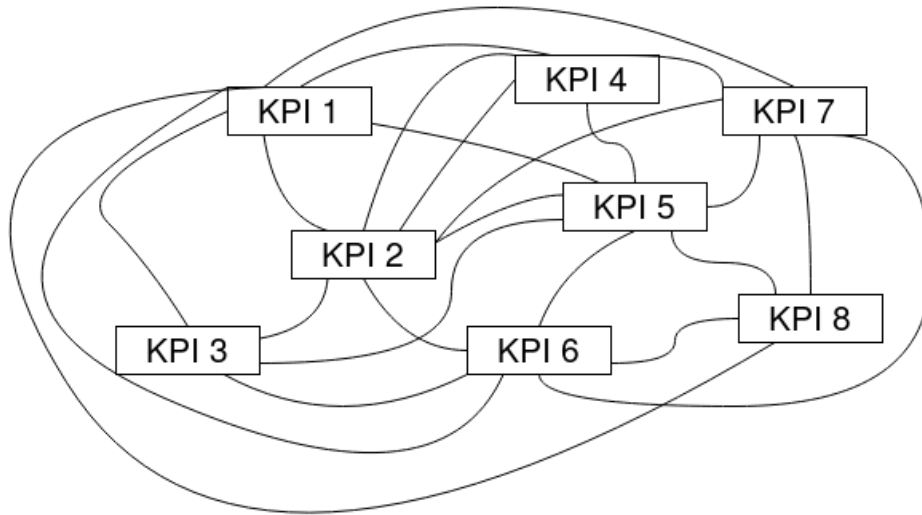
$\rho_{X,Y}$  : Correlation coefficient between KPI X and Y  
 $COV(X,Y)$  : Covariance between datasets of X and Y  
 $\sigma_X$  : Standard deviation of dataset X  
 $\sigma_Y$  : Standard deviation of dataset Y

**2. Define clusters based on Clustering by Fast Search and Find Density Peaks (CFSFDP) algorithm:** This algorithm uses the correlation score between each two KPIs and defines clusters based on the density peaks in the data. The algorithm uses a variable k which defines the number of clusters to be found. In the basic version of the algorithm, this k is predefined and fixed but with some adjustments proposed by Mehmood et al. [62] the algorithm is able to determine the best k dynamically based on the data.

With the above described method, the clustering can add an extra cost to the KPIs aside from the data sources they have in common. Equation 3.3 can then be adapted to the formula given in equation 3.5. Although a lot of research has been done about clustering KPIs, this improvement is not implemented in the KPI suggestion tool due to a lack of datasets.

$$R_c(k) = \frac{a(1 + c(k)) + b(\delta(k))}{c * S_k} \quad (3.5)$$

- 1 Calculate the correlations between every pair of KPIs



- 2 Use this correlation coefficient as distance and cluster KPIs based on this distance

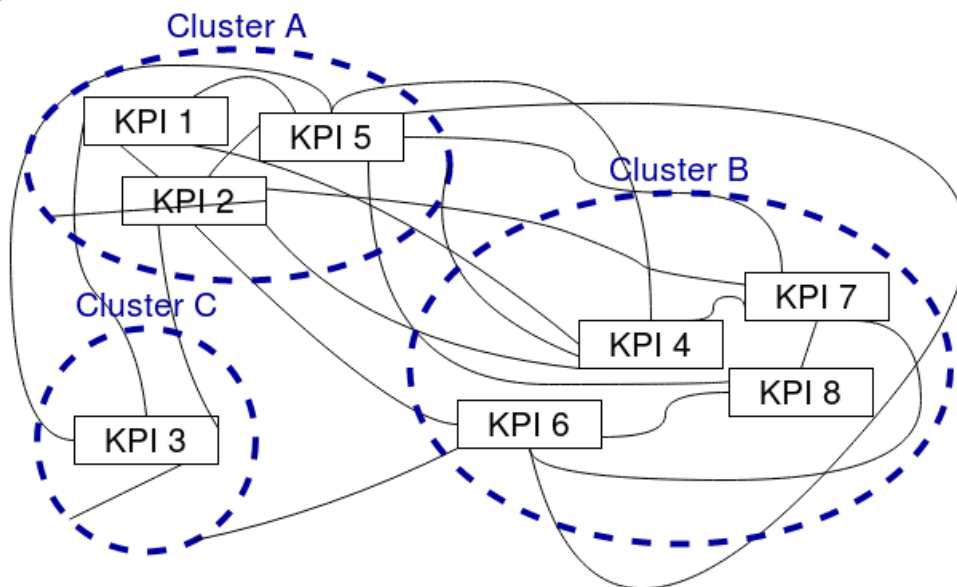


Figure 3.20: The two step method of XingYu [5] to cluster KPIs

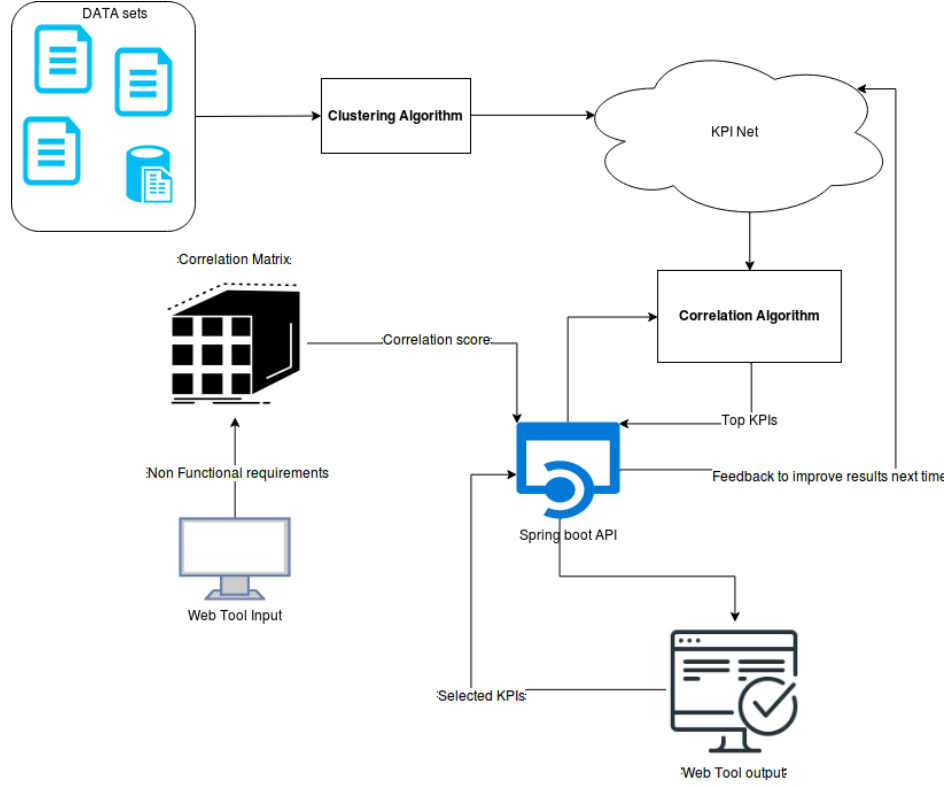


Figure 3.21: Methodology for the KPI suggestion tool with clustering

With

- $R_c(k)$  : The relative cost of KPI k
- $c(k)$  : The data source cost of KPI k
- $S_k$  : The score of KPI k based on the requirements
- $\delta(KPI)$  : The distance function for KPI k
- $a$  : The weight coefficient for the cost
- $b$  : The weight coefficient for the distance function
- $c$  : The weight coefficient for the score

In the above equation the distance cost  $\delta(k)$  is the sum of the correlation coefficients (as described in equation 3.4) between KPI k and each KPI from the top suggested KPIs list

Figure 3.21 gives an overview of the different parts of the KPI suggestion tool with clustering.

### 3.2.3.2 KPI expansion

The KPI net contains 46 KPIs at the moment. Although this is a good first start, the application will always benefit from the addition of new KPIs (provided the correlation scores between this KPI and the requirements make sense). The application provides an API endpoint for this purpose. To add a KPI one needs to provide the name of the KPI, a description, the needed data sources and the scores for each requirement of this KPI. Appendix D shows how KPIs can be added to the KPI net. In addition to this, the importance scores to determine the correlation coefficient between the KPIs (see section 3.2.2.3) require a review. It must be stated that, although a lot of these importance scores are based on literature, they still have been filled in as interpreted by the author of this work.

### 3.2.3.3 Machine Learning approach to improve correlations

The correlation scores between KPIs and requirements as described in section 3.2.2.3 are now based on literature. In the future it would be beneficial to provide some sort of learning system that changes these correlations based on selected KPIs by the end user. The importance score (see section 3.2.2.3) could for example be increased every time a KPI gets selected by the end user. Such an implementation needs profound research to make sure the scores don't change too drastically based on a single user's choice of KPIs and can be reset when needed. Equation 3.6 provides a first suggestion on how selected KPIs can affect the importance score, and thus the correlation coefficient, once the KPI is selected by the user in the final iteration:

$$\rho(k, r) + = \frac{I(r)}{7} \quad (3.6)$$

With

$I(r)$  : The input score for requirement  $r$  as rated by the user  
 $\rho(k, r)$  : Correlation coefficient between KPI  $k$  and Requirement  $r$

We divide by 7 because each requirement is rated on a 1 to 7 scale by the user. This way the importance score will increase by 1 at most. The correlation coefficient is still calculated using equation 3.1.

Figure 3.21 displays this machine learning approach by the arrow running from the Spring boot API to the KPI net.

### 3.3 Combining both methodologies

This section describes how both the KPI suggestion tool and the incremental cost models described above can be combined and complement each other. Figure 3.22 gives a schematic overview of how organizations should use both methodologies together.

First of all organizations must provide the non-functional requirements for the application they want to create. The KPI suggestion tool suggests the most useful KPIs for this application based on this input. The user can iterate over the outcome of the tool as described in section 3.2.2.6 till he is satisfied with the suggested KPIs.

Once the organization knows which KPIs he may need, he can now use the incremental cost model to make an estimate about the cost for the addition of each KPI. The incremental cost model provides a tool for organizations to compare the cost for the addition of the KPI now and in the future. This way they can decide to implement the KPI in development phase. This is recommended if the use of the KPI seems very likely or if the incremental cost of missing the KPI seems very large. If the use of the KPI seems not that likely and the incremental cost seems rather low, an organization may decide to not implement the KPI in the development phase of their application.

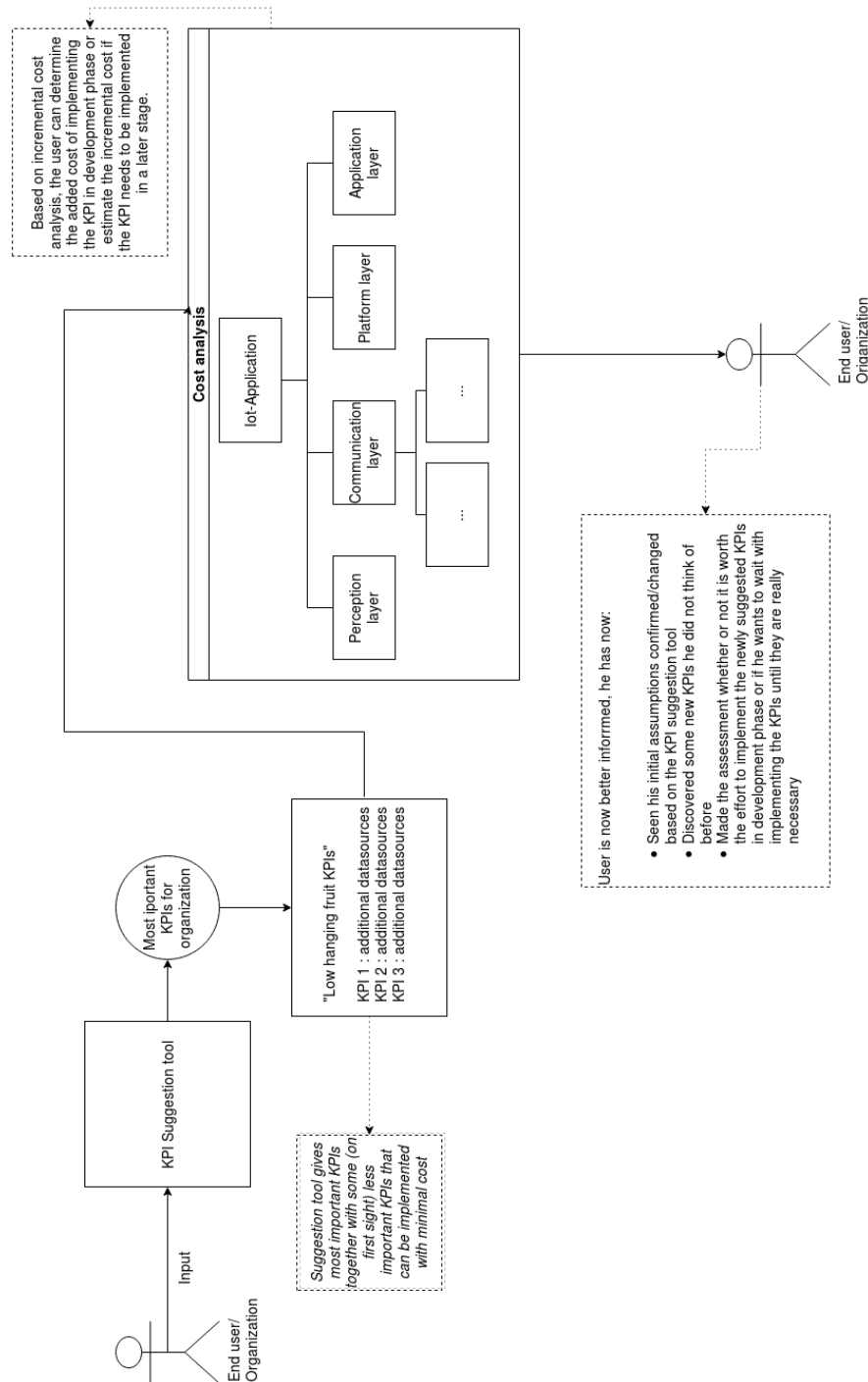


Figure 3.22: Combining both the KPI suggestion tool and incremental cost analysis





# Chapter 4

## Validation

In this chapter, a validation of the above described methodology is done. This validation has the purpose of evaluating how well the web tool performs for new IoT-applications. The application used for validation is the Sundo use case, further described in section 4.1. Section 4.2 provides the strategy used to validate the Sundo use case against the methodology for the KPI suggestion tool. The last section of this chapter focuses on the results of this validation strategy.

### 4.1 The Sundo use case

Sundo is a startup company located in Ghent. Driven by the fact that skin cancer is one of the fastest growing cancers, the persuasion of Sundo is that more active prevention against the dangers of the sun should be taken. People typically protect themselves against the sun when going to the beach or on a vacation, but when walking outside in the summer for casual occasions, this protection is forgotten a lot. This while the danger of burning and thus damaging your skin is still present. Creating awareness of this problem is a first step in this prevention process. However, nowadays not much actions follow on this awareness. For this, Sundo wants to bring sunscreen to the people.

By providing smart sunscreen dispensers to cities, Sundo tries to get the inhabitants to protect themselves better and more often against exposure to harmful sunbeams. Accommodated with sun information like the actual UV-index and temperature, these dispensers offer inhabitants the possibility to shield themselves against the sun. With a user friendly online dashboard, the city can monitor the consumption of each dispenser in a simple way. In addition, automatic notifications about dispensers that are almost empty offer the possibility to plan optimised routes to refill these empty dispensers. At the moment Sundo has an agreement with 3 cities to launch their application. An agreement with two sunscreen suppliers to deliver the sunscreen for their dispensers is also obtained. The launch of the Sundo application is planned in April this year (2020).

As their application is a new IoT application that is never created before, it is an ideal use case to validate the methodology against. Sundo has now reached the prototyping phase in the creation cycle of an IoT application. For this, it may be useful to discover new KPIs before the application is launched and before the scaling phase is reached.

## 4.2 Validation Strategy

In order to test the KPI suggestion tool, the importance of every non-functional requirement as described in section 3.2.2.1 was estimated for the Sundo use case. With this input, the comparison between the KPIs suggested by the KPI suggestion tool and the KPIs already thought of by the Sundo company is made. Two main questions must be solved to validate the methodology described in chapter 3:

1. How well does the KPI suggestion tool perform as a validation tool to check the currently determined KPIs by the organization?
2. How well does the KPI suggestion tool perform as a suggestion tool to predict new, useful KPIs for an IoT application?

As the first of these two questions provides a nice-to-have, a positive outcome on the second question really means an added value for organizations. Notice that, due to the fact that the incremental costs of adding a KPI to

an application can be very high (especially after the scaling phase), the KPI suggestion tool can be of great value even if only one extra KPI is found.

The KPI suggestion tool processes the provided input and displays the suggested KPIs as described in section 2.4.2. Based on these suggested KPIs and the list of KPIs already thought of, the following questions were asked:

1. How many KPIs did the company think of before using the KPI suggestion tool?
2. How many KPIs did the web tool suggest that are not on the list with KPIs the company already thought of?
3. How many KPIs are not in the suggestion of the web tool that are on the list of KPIs the company thought of?
4. How many of the KPIs from the previous question are very use case specific (In this case: are KPIs related to the sun or to sunscreen)?
5. How many KPIs described in question one seem useful for your application?

To be a solid validation tool for KPIs an organization already thought of, the score for question three should be rather low compared to the score on question one.

To be a solid suggestion tool for new KPIs for an IoT application, the score for question three and five should be high.

Question four is mainly used to filter out the KPIs that are very use case specific and, for this, very hard to predict for the KPI suggestion tool. Notice organizations are much more capable of finding the use case specific KPIs for their application. This because these KPIs are linked to the functional requirements in a lot of cases.

The validation strategy consists of two main parts. The first part only uses the outcome of the KPI selection tool without further iterations on the selected KPIs. This to validate how well the tool performs the first iteration and to check whether or not it is useful to perform multiple iterations on the calculation of multiple KPIs.

The second validation step then, does use these iterations on the output until the end user is completely satisfied with the found KPIs. This validation

step shows whether or not it is useful to keep the iteration in the output of the suggestion tool in future improvements. It also allows to gain first insights in which direction the iteration step of the suggestion tool should be improved. In order to verify these assumptions, the following questions were answered on top of the ones asked primarily:

1. How much iterations did you need until the suggested KPIs were complete to your feeling or until you had the feeling no more useful KPIs would be found by the application?
2. How many more KPIs did you found that seem useful to your application after the first iteration?

The following section now describes the outcome of this validation strategy for the Sundo use case.

### 4.3 Validation Outcome

Table 4.1 gives the scores of the non-functional requirements used as input for the KPI suggestion tool (see appendix B for the description of the non-functional requirements). These are the scores rated on the likert scale from 1 to 7 for each non-functional requirement. Based on this input the following KPIs were given by the application in the first validation step (without iterating on the selected KPIs):

- Mean Time Between Failure (MTBF)
- % of devices up
- Average Incident Response Time
- First Response Time
- Energy Consumption
- Signal to Noise Ratio (SNR)
- Mean Time To Repair (MTRR)

Requirement	Score
Security	1
Availability	6
Channel Capacity	3
Backup	3
Data Integrity	4
Disaster Recovery	1
Durability	5
Fault Tolerance	3
Maintainability	7
Performance	5
Response Time	6
Stability	3
Scalability	3
Robustness	6
Throughput	6

Table 4.1: The input scores for the non-functional requirements as given by the Sundo organization

- Average Product Waste

The Sundo organization listed 11 KPIs before using the application. 7 of them were very use case specific (average temperature, average UV-index etc.). The four KPIs that were more general were:

1. received signal strength indicator (RSSI)
2. Signal to Noise Ratio (SNR)
3. Remaining battery life
4. Number of users/day

Three of these four KPIs were not found in the suggested KPIs from the webtool. This raises the assumption that the tool needs extra expansion in order to perform as an reliable validation tool for KPIs of organizations. Notice that, although the energy consumption proposed by the KPI suggestion tool is not the same as the remaining battery life described by Sundo, they resemble a lot.

The following KPIs were marked as worth investigating for future use:

- % of devices up
- Mean Time Between Failure (MTBF)
- Average Incident Response Time
- Energy Consumption
- Average Product Waste

This proves the suggestion tool is certainly capable of suggesting new KPIs that could be useful for an organization in development phase. As stated in section 4.2, the added value of only finding one KPI can be very large due to the incremental costs missing a KPI entails.

Based on the above selection new iterations were done, leading to the following selected KPIs when finished:

- Mean Time Between Failure (MTBF)
- % of devices up
- Average Incident Response Time
- Energy Consumption
- Signal to Noise Ratio (SNR)
- Mean Time To Repair (MTRR)
- Average Product Waste
- Received Signal Strength Indicator (RSSI)

In total 3 iterations were done. Because no new useful KPIs were selected after iteration 3, the above was considered as the final result. The only useful KPI that was added compared to first iteration was the Received Signal Strength Indicator (RSSI). The reason the KPI suggestion did not find much new KPIs after the iteration phase is twofold:

1. The KPI suggestion tool did a pretty good job in the first phase which makes it harder to find new useful KPIs based on the selected KPIs
2. The KPI net does not contain enough specific data sources in order to really determine the distance between KPIs, this reinforces the fact that a KPI net expansion is needed as future improvement.

For future improvements the KPI net should be expanded with more KPIs and data sources as already mentioned in section 3.2.3. This way more iterations will help to suggest more low hanging fruit KPIs based on the selection made by an organization.

The KPIs found in this validation step can certainly create an added value for the Sundo organization. However, these results need the remark that the Sundo organization did not do a deep investigation into the KPIs of their application beforehand. This way it is much easier for the tool to predict new, useful KPIs as the list of already determined KPIs for the Sundo use case was rather limited. This said, the following conclusion can be drawn from the validation strategy:

The Sundo use case shows the KPI suggestion tool is able to find and suggest useful KPIs for an organization. This validation also shows the tool should not be used as the only source to determine all KPIs for an organization. After all, the tool is (at the moment) unable to give 100% guarantees all KPIs for an organization's application will be found. Yet, the tool can be used by an organization as starting point to predict useful KPIs or it can serve as an expansion to the KPIs already defined by the organization. Even when only one useful KPI is discovered by the use of this KPI suggestion tool, this can infer a big economic saving for an organization. For this, it is save to say the tool offers an added value for organizations that will only grow when expanding the KPI net and implementing future improvements.





## Chapter 5

### Conclusion

Internet of Things (IoT) emerged as a concept about 20 years ago and is now making headlines all around the world. More and more applications are becoming smart applications connected to the internet. The data provided by these applications is stored on servers or in the cloud and provided to the end user in order to monitor an application using their smartphone or laptop.

Key Performance Indicators (KPIs) are a well known concept in the business world. They help an organization to measure its most important targets and indicate how well the organization scores on these targets. Based on these KPIs they can take actions to improve the value of their company.

As these KPIs are dynamic, an organization does not always know all KPIs on startup. Adding and re-evaluating these KPIs allows organizations to keep improving, innovating and set new targets.

In IoT applications the measurement of these KPIs is mostly provided by sensors in the IoT application. When this is the case it is much harder to adjust the set of KPIs as this includes adding new sensors to the application. Especially when the application is deployed on a large scale this entails a huge economic effort for the company. This work starts by listing the cost components for an IoT application. Based on these cost models, the required effort to add a new KPI to the IoT application is estimated. The cost models differentiate between the added cost and the incremental cost. Adding a KPI

to the application will bring an extra cost in some cost components. When this cost is the same as the cost of implementing the KPI in development phase, this is called the added cost. This cost is inevitable and is specific to the fact that KPIs are dynamic. It is however possible that costs, due to adding KPIs to the application, could have been avoided by implementing these KPIs in development phase. This is called the incremental cost and allows a company to get an impression which effort it takes to add a new KPI to the application after this development phase.

The cost models presented in this work are based on a 4 layer architecture for IoT applications. The first layer is the perception layer which includes the data sources needed for the application. This perception layer communicates with the platform layer through a communication layer. The platform layer is, among other tasks, responsible for preprocessing and storing the data it receives from the perception layer. At last the application layer presents the stored data to the end user in such a way it creates a useful application. The cost models presented in this work should be used together with the companies knowledge about the application. The models give a first impression on which cost components will create an incremental cost when KPIs are appended to an application after the development phase. The models don't say much about how large this cost will be as this depends on multiple variables. The number of devices, cost of the needed data sources, and type of application are only some variable factors that determine the magnitude of the cost for adding new KPIs. The first part of the methodology of this work, has as main purpose to offer the end user an overview of which types of costs he will face when missing a KPI in development phase.

As the first part of the methodology gives an overview of the different cost components for adding KPIs to an IoT-application, the second part provides the end user with a KPI suggestion tool to predict which KPIs are useful for his application. The tool starts with the non-functional requirements of the application and processes this input together with a prestored KPI net to enrich the user with useful KPIs.

When combined, the incremental cost analysis and KPI suggestion tool offer a great added value for companies. The suggestion tool provides an additional resource to help companies determine KPIs they may not have thought of before. This way the chance of missing KPIs in development phase decreases for a company. When the suggestion tool suggests KPIs but the company is not sure whether or not this KPI will be useful in the future, it can use the incremental cost analysis to compare the cost of adding the

KPI in the development phase of the application to the cost of adding the KPI in a later phase.

The Sundo use case is used to validate the proposed methodology. Sundo is a startup company providing smart sunscreen dispensers to cities. This to offer inhabitants the possibility to protect against harm of the sun. As Sundo creates a new IoT application it is a perfect use case to validate the methodology against. Using the KPI suggestion tool, 8 new KPIs will be investigated, and possibly implemented in the Sundo application. This shows the KPI suggestion tool can help to improve the definition and determination of useful KPIs for on organization.

This work gives a first methodology and suggestion tool to predict useful KPIs. In future work the KPI net should be expanded and more literature and investigation should be done into how KPIs correlate with the non-functional requirements for an application. When more datasets of KPIs can be found, the distances between the KPIs in the KPI net could be expanded. At the moment, the only distance between KPIs is their common data sources. This metric could be combined with the correlation found between KPI-pairs by investigating data sets from two KPIs.

A feedback loop, that allows the suggestion tool to learn and improve from previous selected KPIs could be a first step to predict KPIs using a machine learning approach.

At the moment the cost models described in this work allow to identify the different cost components for an IoT application. Next to this, the incremental cost analysis describes which components are affected when a KPI is added to the application. The models however contain no clue about how large this cost is and how the phase in which the application situates itself affects this cost. Future work should list these costs and supply an impression of the magnitude of the cost for each component described in the models. This magnitude should be examined for every phase for the creation cycle of an IoT application.



# Chapter 6

## Future improvements

The most important future improvements for both the incremental cost models and the KPI suggestion tool are given in chapter 3 in detail. This chapter summarises these improvements. The first section handles the future improvements for the incremental cost models while the second section provides the improvements for the KPI suggestion tool.

### 6.1 Incremental cost analysis

The cost models, described in section 3.1, list the main incremental and added cost components for adding KPIs to an IoT application. The models indicate the different cost components of an IoT application and illustrate which costs to expect when adding KPIs to these applications. The models however, do not give any clues about how large these costs are or which components are the largest cost drivers. With the current model, an organization can identify these costs but needs to estimate the size of this cost for itself. As a first improvement the different costs and their magnitude should be listed and described. Future work should also investigate which parameters cause the increase in (incremental) costs for every cost component. Some first ideas of these parameters are given below.

A first parameter is the scale on which the application is already deployed.

It is clear that it will be harder to implement a new KPI when there are 1000 devices of the application compared to when only one device is deployed.

A second parameter is the geographical distribution of the application. Adding a sensor to an IoT application that is distributed very locally (for example only in one company) will be much easier than adding this sensor to an application that is distributed worldwide.

A third parameter could be the value of the application and the fact whether or not the application needs to be redeployed to implement a new KPI. When the implementation of a new KPI causes the application to be redeployed, this means that the application can not be used for some time. When the application creates very high value products, this can lead to a very high loss in production.

The above are only some suggestions of parameters that should be looked into. In future work, a profound investigation into all parameters that affect the incremental costs, is needed. This way organizations can, instead of only identifying the cost based on these models, also predict very precise how much missing a KPI for an IoT application will cost and which are the main cost drivers.

A last suggested improvement is to identify the magnitude of the incremental costs for every phase of the creation process for IoT applications. the costs for adding KPIs to an IoT application will not have the same magnitude in every phase of this process. So will the cost for adding a KPI in the prototyping phase normally be much lower compared to the cost of adding the same KPI after the scaling phase. Future work could give an idea about what the magnitude of these costs are for every phase.

## 6.2 KPI suggestion tool

As seen in the validation chapter 4 the KPI suggestion tool is capable of suggesting useful KPIs for an IoT application. Yet, to validate KPIs for an organization it is (at the moment) not 100% reliable. In order to improve the suggestion capabilities and to increase the reliability of the tool as a validation tool, following improvements are suggested:

A first improvement would be to expand the KPI net. The KPI net contains 46 KPIs at the moment. Although this is a good first start, the

application will always benefit from the addition of new KPIs (provided the correlation scores between this KPI and the requirements make sense). The application provides an API endpoint for this purpose.

A second improvement is to use other cost functions than only the common data sources between KPIs. Clustering of the KPIs allows to define a cost penalty between each pair of KPIs. A first suggestion for this clustering is described in chapter 3. Data sets, generated by the measurement of KPIs, could be used to determine the correlation coefficient which counts as the distance between two KPIs. The Clustering by Fast Search and Find Density Peaks (CFSFDP) algorithm can now determine clusters in the KPI net. Based on these clusters an extra cost could be taken into account in the suggestion of low hanging fruit KPIs.

A last improvement given in chapter 3 is to make use of a machine learning approach to improve the KPI suggestion on previous inputs. It would be beneficial to provide some sort of learning system that changes the correlations between KPIs and requirements based on selected KPIs by the end user. The importance score (see section 3.2.2.3) could for example be increased every time a KPI gets selected by the end user. Such an implementation needs profound research to make sure the scores do not change too drastically based on a single user's choice of KPIs and can be reset when needed.





# Bibliography

- [1] KpiLifeCycle. Chart-KPI-steps\_credits\_flock-associates800x535-1.jpg (JPEG Image,  $800 \times 535$  pixels).
- [2] Industry\_4.0.png (PNG-image,  $1332 \times 647$  pixels).
- [3] As\_the\_price\_of\_IoT\_sensors\_falls\_use\_is\_expected\_to\_increase.png (PNG-image,  $600 \times 371$  pixels).
- [4] N. Stricker, M. Micali, D. Dornfeld, and G. Lanza. Considering Interdependencies of KPIs – Possible Resource Efficiency and Effectiveness Improvements. *Procedia Manufacturing*, 8:300–307, 2017.
- [5] Xingyu Guo, Peng Yu, Wenjing Li, and Xuesong Qiu. Clustering-based KPI data association analysis method in cellular networks. *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, (November 2017):1101–1104, 2016.
- [6] Andy Neely, Huw Richards, John Mills, Ken Platts, and Mike Bourne. Designing performance measures: A structured approach, 1997.
- [7] Parmenter David. *Key Performance Indicators: developing, implementing, and using winning KPIs*, volume 53. 2013.
- [8] Statista. • IoT: number of connected devices worldwide 2012-2025 — Statista.
- [9] Vasu Pasupuleti. The Importance of Business and Performance KPIs for IoT Applications — Blog — Appdynamics.

- [10] Knud Lasse Lueth. State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating. *IoT Analytics*, pages 1–7, 2018.
- [11] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 2010.
- [12] Internet of things - Wikipedia.
- [13] How IoT’s are Changing the Fundamentals of “Retailing”.
- [14] Amazon.com Help: Alexa & Alexa Device Terms.
- [15] Explore what you can do with Google Nest devices - Google Nest Help.
- [16] HomeKit: het complete overzicht voor slimme apparaten in huis.
- [17] Industry 4.0 - Wikipedia.
- [18] Klipfolio. What is a KPI? Definition, Best-Practices, and Examples, 2017.
- [19] Alexander Gluhak, Srdjan Krco, Michele Nati, Dennis Pfisterer, Nathalie Mitton, and Tahiry Razafindralambo. Challenges for Iot Experimentation. (November):58–67, 2011.
- [20] Dusit Niyato, Xiao Lu, Ping Wang, Dong In Kim, and Zhu Han. Economics of Internet of Things: An information market approach. *IEEE Wireless Communications*, 23(4):136–145, 2016.
- [21] Dejan Munjin and Jean Henry Morin. Toward Internet of Things application markets. *Proceedings - 2012 IEEE Int. Conf. on Green Computing and Communications, GreenCom 2012, Conf. on Internet of Things, iThings 2012 and Conf. on Cyber, Physical and Social Computing, CP-SCom 2012*, pages 156–162, 2012.
- [22] Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zuallakernan. Internet of things (IoT) security: Current status, challenges and prospective measures. *2015 10th International Conference for Internet Technology and Secured Transactions, ICITST 2015*, pages 336–341, 2016.
- [23] PubNub. A New Approach to IoT Security. 2015.

- [24] Lixing Li, Zhi Jin, Ge Li, Liwei Zheng, and Qiang Wei. Modeling and analyzing the reliability and cost of service composition in the IoT: A probabilistic approach. In *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, pages 584–591, 2012.
- [25] Zhen Yang and Deshi Li. IoT information service composition driven by user requirement. *Proceedings - 17th IEEE International Conference on Computational Science and Engineering, CSE 2014, Jointly with 13th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2014, 13th International Symposium on Pervasive Systems,,* pages 1509–1513, 2015.
- [26] Sam Lucero. IoT platforms: enabling the Internet of Things. *IHS Technology*, Whitepaper(March):1–19, 2016.
- [27] Google. Prijzen voor Cloud SQL — Cloud SQL Documentation — Google Cloud.
- [28] Amazon. Microsoft {SQL} {S}erver - {SQL} {S}erver Home.
- [29] Microsoft. Pricing – Azure SQL Database Managed Instance — Microsoft Azure, 2019.
- [30] Dusit Niyato, Dinh Thai Hoang, Nguyen Cong Luong, Ping Wang, Dong In Kim, and Zhu Han. Smart data pricing models for the internet of things: A bundling strategy approach. *IEEE Network*, 30(2):18–25, 2016.
- [31] Matt Leonard. Declining price of IoT sensors means greater use in manufacturing — Supply Chain Dive.
- [32] Harpreet S. Dhillon, Howard Huang, and Harish Viswanathan. Wide-Area Wireless Communication Challenges for the Internet of Things. *IEEE Communications Magazine*, 55(2):168–174, 2017.
- [33] Lei Shi, Linda Newnes, Steve Culley, James Gopsill, Simon Jones, and Chris Snider. Identifying and visualising kpis for collaborative engineering projects: A knowledge based approach. In *Proceedings of the International Conference on Engineering Design, ICED*, volume 3, pages 377–386, 2015.
- [34] Gyun Hwang, Jeongcheol Lee, Jinwoo Park, and Tai Woo Chang. Developing performance measurement system for Internet of Things and smart factory environment. *International Journal of Production Research*, 55(9):2590–2602, 2017.

- [35] Jovan Rakar, Zorzut. Assessment of production performance by means of KPI. (September), 2004.
- [36] Pan Wang and Wei He. Research on key performance indicator (KPI) of business process. In *Proceedings of the 2012 2nd International Conference on Business Computing and Global Informatization, BCGIN 2012*, pages 151–154. IEEE, oct 2012.
- [37] Bernard Marr, Gianni Schiuma, and Andy Neely. Intellectual capital – defining key performance indicators for organizational knowledge assets, oct 2004.
- [38] Ella Roubtsova and Vaughan Michell. A method for modeling of KPIs enabling validation of their properties. In *ACM International Conference Proceeding Series*, 2013.
- [39] Kent Bauer. KPI Reduction the Correlation Way. *DM Review*, 15(2):64, 2005.
- [40] Bruno Gries and John Restrepo. KPI measurement in engineering design - A case study. *ICED 11 - 18th International Conference on Engineering Design - Impacting Society Through Engineering Design*, 1(August):531–537, 2011.
- [41] Sonja Meyer, Andreas Ruppen, and Carsten Magerkurth. Internet of things-aware process modeling: Integrating IoT devices as business process resources. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7908 LNCS:84–98, 2013.
- [42] Jesús Peral, Alejandro Maté, and Manuel Marco. Application of Data Mining techniques to identify relevant Key Performance Indicators. *Computer Standards and Interfaces*, 54:76–85, 2017.
- [43] KPILibrary. Key Performance Indicators (KPI) Examples, Dashboard & Reporting.
- [44] Ningxuan Kang, Cong Zhao, Jingshan Li, and John A. Horst. A Hierarchical structure of key performance indicators for operation management and continuous improvement in production systems. *International Journal of Production Research*, 54(21):6333–6350, nov 2016.
- [45] Karola Karlson. What is a KPI Dashboard? (The Complete Guide) — Scoro.

- [46] TATJANA SAMSONOWA, PETER BUXMANN, and WOLFGANG GERTEIS. DEFINING KPI SETS FOR INDUSTRIAL RESEARCH ORGANIZATIONS — A PERFORMANCE MEASUREMENT APPROACH. *International Journal of Innovation Management*, 13(02):157–176, jun 2009.
- [47] Mari Fukuda, Jun-Jang Jeng, and Yinggang Li. United States US 20090281 845A1 (12). Technical Report 10.
- [48] P Willaert, J Willems, and S Viaene. 740 2006 IRMA International Conference Process Performance Measurement: Identifying KPI's that Link Process Performance to Company Strategy. Technical report, 2006.
- [49] Mari Abe, Jun Jang Jeng, and Yinggang Li. A tool framework for KPI application development. In *Proceedings - ICEBE 2007: IEEE International Conference on e-Business Engineering - Workshops: SOAIC 2007; SOSE 2007; SOKM 2007*, pages 22–29. IEEE, oct 2007.
- [50] Glenney De Cock. Techno-economische kostmodellering voor IoT-technologieën via implementatie van een online selectie- en modelleringstool. 2019.
- [51] Wikipedia. Smart Object, 2019.
- [52] Andrei Klubnikin. Internet of Things: How Much Does it Cost to Build IoT Solution? *R-Style Lab*, pages 1–10, 2016.
- [53] The Things Network. The Things Network, 2018.
- [54] Adam Trendowicz. Software Cost Estimation, Benchmarking, and Risk Assessment: The Software ... - Adam Trendowicz - Google Boeken, 2013.
- [55] Rachel Davies. Non-Functional Requirements: Do User Stories Really Help? *Methodsandtools.com*.
- [56] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Non-functional requirements. In *Csc.calpoly.edu*, page 699, 2002.
- [57] Plh Eide. Quantification and Traceability of Requirements. *Quantification and Traceability of Requirements*, 2005.
- [58] Scott Ambler. Technical (Non-Functional) Requirements: An Agile Introduction. *Agile Modelling*.

- [59] Joe Saur. Effective requirements practices. *ACM SIGSOFT Software Engineering Notes*, 27(3):93, 2002.
- [60] Hiroshi Wada, Junichi Suzuki, and Katsuya Oba. Modeling non-functional aspects in service oriented architecture. In *Proceedings - 2006 IEEE International Conference on Services Computing, SCC 2006*, pages 222–229, 2006.
- [61] Overview (javax.persistence-api 2.2 API).
- [62] Rashid Mehmood, Rongfang Bie, Libin Jiao, Hussain Dawood, and Yunchun Sun. Adaptive cutoff distance: Clustering by fast search and find of density peaks. *Journal of Intelligent and Fuzzy Systems*, 31(5):2619–2628, oct 2016.

# Appendices





## Appendix **A**

### Questionnaire for IoT Companies

Below, the questionnaire used for the interviews with several IoT companies is given. This questionnaire was used in the literature phase in order to gain insights in the following questions:

- Is the research question relevant?
- Could a KPI suggestion tool create an added value for companies?
- How are KPIs determined nowadays in organizations?
- Why are KPIs measured in organizations?
- What is important when selecting KPIs?
- Which domains are very important to develop KPIs for?
- Discover domain specific KPIs

Date: .....

# Methodology for KPI-selection and incremental cost analysis for lot-applications in development phase

Name Company: \_\_\_\_\_

Do you see an added value in the research question of this master dissertation?

How are KPIs estimated in development phase in your organization? Do you have best practices/ strict rules in order to determine KPIs for a new application?

Did you ever encounter problems facing KPIs that could have been avoided by using a KPI selection tool like the one proposed in this work? If yes, which ones?

Date: .....

What are the most important problems you're trying to tackle/ the most important goals you try to obtain when developing an IoT application (Think of minimizing system downtime, increasing coverage of the application...)

Which KPIs would you describe as being very common/important when developing an Iot application? (Both domain related as in general)

Can you give a correlation between different domains and KPIs in general at first sight?

Date: .....

If the scope of this master dissertation is not feasible and it turns out to be impossible to develop a webtool for all possible Iot applications, which domains seem most useful/achievable to develop the webtool for?

**Useful:**

**Achievable:**

Extra suggestions:

# Appendix B

## Content of the KPI net

### B.1 Non-functional requirements

Table B.1: Overview of the non-functional requirements present in the KPI net

requirement name	requirement description
Security	Security is freedom from, or resilience against, potential harm caused by others. Beneficiaries (technically referents) of security may be of persons and social groups, objects and institutions, ecosystems or any other entity or phenomenon vulnerable to unwanted change
availability	The degree to which a system is in a specified operable and committable state at the start of a mission, when the mission is called for at an unknown, i.e. a random, time. Simply put, availability is the proportion of time a system is in a functioning condition

Channel capacity	Channel capacity, in electrical engineering, computer science, and information theory, is the tight upper bound on the rate at which information can be reliably transmitted over a communication channel.
backup	backup, or data backup is a copy of computer data taken and stored elsewhere so that it may be used to restore the original after a data loss event.
data integrity	Data integrity is the maintenance of, and the assurance of the accuracy and consistency of data over its entire life-cycle,[1] and is a critical aspect to the design, implementation and usage of any system which stores, processes, or retrieves data.
disaster recovery	Disaster Recovery involves a set of policies, tools and procedures to enable the recovery or continuation of vital technology infrastructure and systems following a natural or human-induced disaster.
durability	Durability is the ability of a physical product to remain functional, without requiring excessive maintenance or repair, when faced with the challenges of normal operation over its design lifetime.
fault tolerance	Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the sever
maintainability	maintainability is the ease with which a product can be maintained in order to: correct defects or their cause,repair or replace faulty or worn-out components without having to replace still working parts, prevent unexpected working condition, meet new re
performance	performance is the amount of useful work accomplished by a computer system.
response time	response time is the time a system or functional unit takes to react to a given input.
stability	Stability is the degree in which the system must not alter over time.

scalability	Scalability is the property of a system to handle a growing amount of work by adding resources to the system.
Robustness	robustness is the ability of a system to cope with errors during execution and cope with erroneous input.
Throughput	In general terms, throughput is the rate of production or the rate at which something is processed.

## B.2 KPIS

Table B.2: Overview of the KPIS present in the KPI net

Name	Description	Direction	Unit
Mean Time Between Failure (MTBF)	The predicted elapsed time between inherent failures	max.	time
Mean Time To Repair (MTTR)	Represents the average time required to repair a failed component or device.	min.	time
Availability	Availability is a function of the total service time, the mean time between failure (MTBF), and the mean time to repair (MTTR).	max	percentage
Package loss	percentage of packets transmitted over the network that did not reach their intended destination. A 0 percent package loss indicates no packets were lost in transmission.	min	percentage
Downtime	The formula derives the percentage of the time service is available. The inverse is the amount of downtime.	min	percentage

Signal strength	received signal strength indicator (RSSI) is a measurement of the power present in a received radio signal.	max	DB-microvolts
Average incident response time	The average amount of time (e.g. in minutes) between the detection of an incident and the first action taken to repair the incident.	min	time
Unplanned availability	Percentage of outage (unavailability) due to incidents in the IT environment, relative to the service hours.	min	percentage
Uptime	The time during which the system is operational. Often defined in terms of percentages	max	percentage
Energy consumption	Evaluate the consumption per sector over time, so as to identify a trend of consumption and be able to predict future needs more accurately.	min	Energy in kWh
Power cuts & average duration	measures the number of time your facility suffers a power outage, and how much time it represents at the end of the month.	min	Events
Energy production costs	The production costs represent the net present value of the unit-cost of electricity of a certain energy source	min	Minutes
number of detected network attacks	Represents the number of both succesful and unsuccessful attacks in a certain time period	min	days
Risk level matrix	Risk Level Matrix – Mearures various risks and their likelihood.	min	score
Emission factor	The emission factor for the machine	max	Kg-CO2
Air pollution		max	score
Air quality	The quality of the air in the room		



% of devices up	measure how well you're actually delivering that service, and you have to drive up this quality-metric in order to scale		percentage
Active users	Number of users that are using the application at the moment	optimal val.	users
Activated users	How many users are signed up for the application	optimal val.	users
response time	the time a system or functional unit takes to react to a given input.	min	time
First response time	number of minutes, hours, or days between when a customer submits a support ticket and when a customer support representative provides an initial response	min	time
Average number of incidents per device	Average number of incidents per device	max	percentage
efficiency	Number of produced units per fixed time period	optimal val.	units
Average Product Waste	Amount of the end product the application that goes to waste.	min	percentage
% of repeat incidents	Percentage of incidents that can be classified as a repeat incident, relative to all reported incidents within the measurement period. A repeat incident is an incident that has already occurred (multiple times) in the measurement period.	min	percentage
Average time to procure	Average time to procure an item. Time lag between request for procurement and signing of contract or purchase.	min	time

% of availability SLAs met	Percentage of availability Service Level Agreements (SLAs) met.	max	percentage
% of availability SLAs not met	Percentage of response-time SLAs not.	min.	percentage
% of unauthorized changes	number of unauthorized implemented changes relative to all implemented changes within a given time period. An unauthorized change can be detected through consolidation of the CMDB. A change in infrastructure for which there is not a change registered is considered as unauthorized.	min	percentage
Subscriptions' NET Growth	how many new subscriptions am I adding NET on a monthly basis, otherwise known as 'Net New Adds', as all IoT	max.	users
Average Revenue Per Unit (ARPU)	the average revenue you've already received from your customers per application unit.	max.	revenue
Life Time Value	The revenues attributed to the entire relationship with a customer	max.	revenue
Planned Busy Time	The planned time during which a machine is busy.	min	time
Planned operation time (POT)	The scheduled time during which a machine can be utilized.	max	time
Planned run time per item (PRI)	The planned time to produce one piece or part.	min	time
Planned unit setup time (PUST)	The planned time for a machine to setup for an order.	min	time

Actual unit processing time (AUPT)	The time necessary for production and setup on a machine for an order.	min	time
Actual production time (APT)	The actual time in which the machine is producing for an order, which only includes the value-adding functions.	min	time
Actual unit idle time (AUIT)	The actual time when the machine is not executing order production even if it is available. This can also be referred to as actual unit delay time (ADET)	min	time
Processed quantity (PQ)	The quantity that a work unit has processed	optimal value	absolute value
Failure event (FE)	The count over a specified time interval of the terminations of the ability for a machine to perform a required operation.	min	failures
Blocking time (BLT)	The idle time of an equipment during events that parts cannot go downstream.	min	time
Starving time (STT)	The idle time of an equipment during events that parts cannot arrive from upstream.	min	time
Throughput rate (TR)	The process performance indicator in terms of produced good part quantity of an order	max	percentage
Production process ratio (PR)	The efficiency of production when considering the actual unit setup time, delay time, transportation time, and queuing time.	max	percentage



## Appendix C

# Deployment of the application

This appendix describes how the KPI suggestion tool can be downloaded and run locally. The tool is located on a github repository and can be run using docker.

## C.1 Download the application

The application is located on a github repository. This repository needs to be cloned. On windows and mac OSX, users can use sourcetree as GIT GUI (<https://www.sourcetreeapp.com/>). On a linux machine users can use the git clone command. The URL to the webtool is given below:

```
https://github.ugent.be/vidsmet/kpiSuggestionTool.git
```

When using sourcetree follow the steps listed below as in figure C.1:

1. Click on the "Clone/New" button.
2. Click on "Clone Repository".
3. Fill in the remote repository URL (<https://github.ugent.be/vidsmet/kpiSuggestionTool.git>) and all other details.

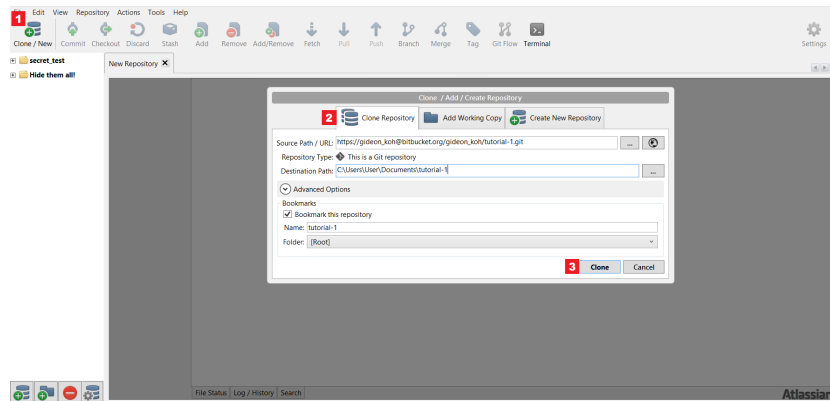


Figure C.1: Steps to download the application from github

#### 4. Click "Clone".

When using linux, open a new terminal, create an empty destination folder and use the following command:

```
git clone https://github.ugent.be/vidsmet/kpiSuggestionTool.git
```

The source files should now be in the newly created folder.

## C.2 Install docker

Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries. To run the application, an installation of docker is needed first. The installation manual can be found on the official docker page for every operating system:

- Microsoft Windows 10 Professional of Enterprise 64 bit:  
<https://docs.docker.com/docker-for-windows/install/>
- Apple MacOS Sierra 10.12 or higher:  
<https://docs.docker.com/docker-for-mac/install/>
- Linux CentOS:  
<https://docs.docker.com/install/linux/docker-ce/centos/>

- Linux Debian:  
<https://docs.docker.com/install/linux/docker-ce/debian/>
- Linux Fedora:  
<https://docs.docker.com/install/linux/docker-ce/fedora/>
- Linux Ubuntu:  
<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

For linux users Docker Compose also needs to be installed (on windows and mac this is included in the Docker installation). Docker compose allows to bundle several docker images which is needed to run the front-end and back-end application simultaneous. Docker compose can be installed on Linux using the manual from the official docker compose installation page (<https://docs.docker.com/compose/install/>)

## C.3 Deploy the application

Once Docker and Docker compose are installed the application can be deployed. For this open a new terminal and navigate to the local git folder on your computer. Starting from this folder, complete the following steps:

1. Navigate to the docker folder (`cd ./docker` on Linux)
2. Execute the `docker-compose build` command. This will build the Docker image for the application
3. Execute the `docker-compose up` command. This will launch the Docker image build in the previous step
4. The application is now available at `localhost:8080`. Use your browser to navigate to `localhost:8080`

The application is now fully deployed and can be used to suggest KPIs for your application





## Appendix D

### Expanding the KPI net

As mentioned in the methodology chapter. The KPI net should be expanded as future improvement. The front-end offers end users the possibility to add KPIs, data sources and non-functional requirements. Figure D.1 gives a printscreen of what these pages look like. To add a KPI the user needs to go to

**<https://localhost:8080/addKPI/>**

and fill in all details the application needs. This includes the name and description of the KPI, the scores for the correlation between the new KPI and the non-functional requirements and the data sources needed to measure the KPI.

Using the same method non-functional requirements can be added (as given in figure D.2) using the following url:

**<https://localhost:8080/addRequirement/>**.

For this the user needs to score all KPIs that are currently in the KPI net for the back-end to recalculate the correlations.

Frontend - Mozilla Firefox

localhost:4200/addKPI

Zoeken

80%

Welcome to the KPI Selection tool!

**Add a KPI to the KPI net**

Name:

Description:

**Non-functional Requirements Score**

For every non-functional requirement, rate how much the requirement should affect the KPI from 1 to 5. The scores are normalized by the back-end of the application

Security	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	
availability	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	
Channel capacity	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	

**Needed Data Sources**

Select the datasources that are needed to measure the KPI

If the datasource is not in the list, it should be added [first here](#). Double click a data source to select.

Type to filter data sources

- Heat sensor
- Click
- Vibration sensor
- CO2-sensor
- Event counter
- Infrared sensor

Clear All fields

Add KPI

Figure D.1: Printscreen of the add KPI page

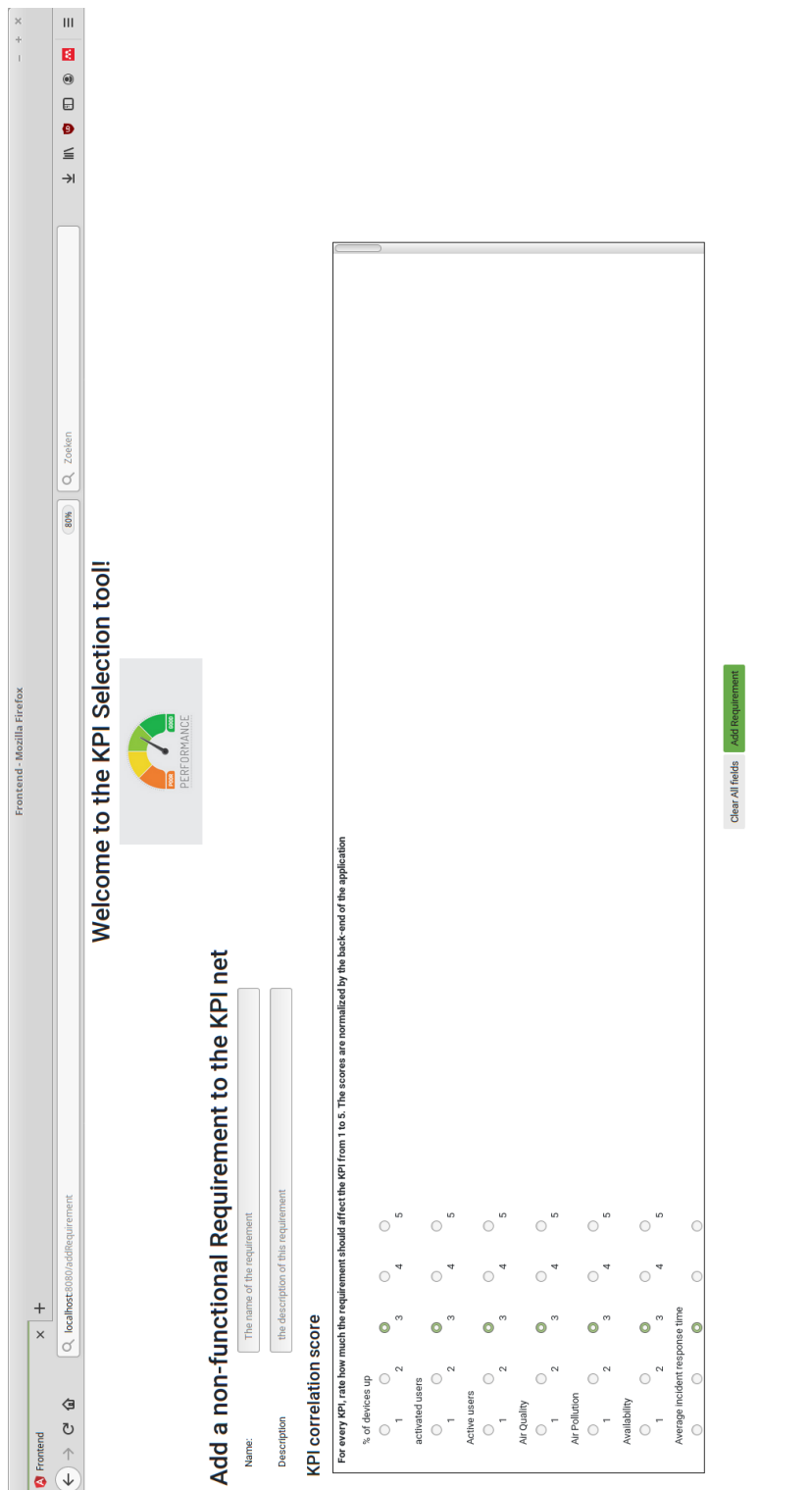


Figure D.2: Printscreen of the add Requirements page



