TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABREVIATIONS

AI	Artificial Intelligence
AFNOR	Association Française de Normalisation
COSMIC	Common Software Measurement International Consortium
CRS	Course Registration Case Study
EDA	Exploratory Data Analysis
ÉTS	École de Technologie Supérieure
FP	Function Point
FPA	Function Point Analysis
FPS	Function Point Sizing
FSM	Functional Size Measurement
FUR	Functional User Requirements
HDF	Hierarchical Data Format
HTML	Hypertext Markup Language
ID	Identification
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ISO	International Organization for Standardization
IT	Information Technology
IWSM	International Workshop on Software Measurement
LED	Light Emitting Diode
KPA	Key Process Area
MENSURA	Conference on Software Process and Product Measurement
ML	Machine Learning
MPC	Measurement Practice Committee.
NESMA	Netherlands Software Metrics Users Association.
NFR	Non-Functional Requirement

OD	Operating Delay
OS	Operating Standby
R&D	Research & Development
Resto-Sys	Restaurant Management System
RFID	Radio Frequency Identification
SDLC	Software Development Life Cycle
SPT	Standard Production Time
SQL	Structured Query Language
Std	Standard Deviation
UML	Unified Modeling Language
URL	Uniform Resource Locator

LIST OF SYMBOLS

CFP	COSMIC function point
hrs	Hours
p	Probability of obtaining a test statistic (p-value)
z	Z-score / Z-critical
%	Percentage

INTRODUCTION

All companies in the software industry, of whatever size, are facing the challenge of having to estimate their software development efforts (Abran, 2015), and the functional size of a software is one of the estimation inputs that can be measured using international standards of measurement. It can be sized either at the beginning of the project, during the project life cycle and at the end of the project once the software has been built.

In the literature, most of the models for estimating software development efforts are based on data collected after the fact on completed projects (Abran, 2015) (Cheikhi & Abran, 2014) (Dumke & Abran, 2011) (Jayakumar & Abran, 2017). They correspond to a posteriori models (e.g., modeling the past) that were built with variables that are known and measured at the end of projects using a wide variety of algorithmic estimation techniques and, more recently, machine learning (ML) (García-Floriano, López-Martín, Yáñez-Márqu, & Abran, 2018) (Hosni, Idri, & Abran, 2018) (Idri & Abran, 2018).

The COSMIC-ISO 19761 measurement of the functional size of the software is one of the main variables for the construction of these posteriori estimation models (Abran, 2010) (García-Floriano, López-Martín, Yáñez-Márqu, & Abran, 2018) (Jayakumar & Abran, 2017). It makes it possible to quantify in a standardized, repeatable, and objective way the functions delivered in a software, and independently of development technologies.

When the estimation models are considered for a priori estimation at the beginning of the project, it is with the implicit assumption that the independent variables of the model are known and can be measured in a precise way from the beginning of a project. However, in practice this is far from being the case, and there is practically no work in the software engineering literature that explores this issue of a priori measures of the software to be developed, including from partial and modifiable information during the project, and their use within a priori estimation models (Almakadmeh & Abran, 2013) (Lopez-Martin & Abran, 2012) (Ungan, Trudel, & Abran, 2018).

This thesis reports on the research carried out to explore a posteriori measurement of functional size of software applications in the specific software development context of real-time embedded and of AI software. This research was conducted in a small size company designing software and AI solutions for the underground mining industry. This is followed next by the development of a sizing approximation technique applicable within the context of incomplete requirements (e.g. within an a priori estimation context), as well as of its usage with industry data.

The motivation of this research project is to assist engineers by providing the size information on the software requirements (in both priori and posteriori contexts) and to use this size to build realistic estimation models for their software projects and improve their planning. These functional size requirements can be documented at different levels of granularity when estimating future projects.

Research Issue

It is challenging for software engineers to identify all the functional requirements as well as the size and required effort of a piece of software before starting their project, in particular when:

1. A project is new and there is lack of information;
2. A project is complex, i.e. a lot of features need to be developed;
3. Time is limited, and it is not possible to identify all the functional requirements.

To help in tackling these challenges, the engineers can use their historical data (if documented well) to identify the functional requirements or use the historical data of other external projects to help determine the functional requirements.

This ETS research project was initiated within the context of the following ETS R&D project funded by MITACS[1] and a private organization in Montreal: “A priori effort estimation models of embedded software development for underground mines communications”. When this R&D project was initiated in 2018, there was virtually no work in the software engineering literature that explores the issue of a priori measures of software based on partial information, and their uses in estimation models.

In this underground mining environment, new software solutions are developed to optimize underground communications, improve safety and operational performance. The research issues that have been identified in this specific development environment were:

Issue #1: lack of benchmark data based on estimated size and effort for an adequate approximation for future projects. This lack of data inhibits the ability to estimate for a new software solution the required effort, duration, money, and resources (mostly people);

Issue #2: no usage of any standard to measure the functional size of the requirements, for benchmarking studies as well as for estimation purposes;

Issue #3: lack of documentation of the functional user requirements (FUR) in this development environment of embedded software and of AI applications. The FUR need to be for maintenance and estimation purposes;

Issue #4: the size of AI projects had not been measured yet either in research or industry using function point (FP) techniques. For instance, in a 2020 Software Project Management course offered to graduate students, only the measurement of real-time and business applications was taught in the course curriculum.

---

[1] Nonprofit national research organization

Research Objectives

The initial objectives of this research were to operationalize and experiment in the context of the private organization funding this R&D project the "a posteriori" measurement of functional size of software proposed in (Abran, 2015) and next evaluate its contribution within an "a posteriori" and "a priori" estimation with historical data in the specific context of real-time embedded and AI software developed for underground mines. The software projects developed by in that organization during a period of 12 months were to be measured with COSMIC - ISO 19761. The initial objectives of that R&D funding proposal were:

Initial objective 1: a first sub-project for the posteriori measurement with the COSMIC - ISO 19761 standard of the software developed in that organization, followed by the construction of a posteriori estimation models for these projects, and evaluation of their performance as estimation models;

Initial objective 2: a second sub-project to determine at which stage of the development cycle each of these details was known and could have been measured in a priori context rather than in a posteriori context. Analysis of measurement patterns to identify scale factors and origins of these factors. From the scale factors developed in the previous step, the construction of a historical a priori database;

Initial objective 3: a third sub-project for the construction of a priori estimation models of these projects with these a priori measures and evaluating their performance as a priori estimation models.

Within the detailed planning and execution of this ETS R&D project, a number of additional practical constraints and new research challenges have been identified, and in particular:

1. Functional size measurement: there was no documentation of the software functional requirements, either in an 'a priori' context at a project stage, nor in an "a posteriori" context

of an implemented software. Only actual lines of code were available, of which the functional requirements had to be extracted for measurement purposes.

2. Types of software: in addition to the type of real-time embedded for underground mines, the organization had started to develop AI software and ML algorithms to analyse data and develop predictive models. When this R&D started, there were no literature exploring whether or not the COSMIC standard could be used for such types of software.

Under these constraints of 1 and 2, this raised then two research questions (RQ) for this thesis:

RQ 1: Can the COSMIC functional size be used to measure real-time embedded software?
RQ 2: Can the COSMIC functional size be used to measure AI software and ML algorithms?

3. In the literature, the posteriori estimation models using linear regression are typically built using a relatively small set of projects developed by teams within a timeframe of weeks or months, and a significant variation in functional sizes; however, the data made available for this thesis research were small programming tasks carried mostly by individuals, over a period of a few days maximum, and for a period on only three months (instead of the targeted 12 months of data) within the context of a 'Release' of changes to existing set of real-time embedded software, where a 'change' corresponded to either the addition or modification of a 'feature' or a correction of a 'defect'. This then raised two additional research questions for this thesis:

   RQ 3: Do linear regression models based on functional size work well for set of very small projects developed by individuals and when their size is very small and within a very small range?
   RQ 4: If the answer to RQ 3 does not provide good posteriori models, can descriptive statistics be used to provide relevant posteriori estimation models that could be used in a priori context?

4. In this organization there was no a priori or a posteriori documentation available to explore the development of scaling factors for their usage in ‘a priori’ context: therefore, another alternative set of data had to be identified to explore this research issue. Since there is very good documentation and detailed a posteriori COSMIC measurement results in COSMIC case studies, these were used in this thesis research work to explore the development of scaling factors. This led to a fifth research question:

   RQ5: Can the detailed documentation and functional size measurement results in COSMIC case studies be used for the development of size scaling factors applicable early in a software development life cycle?

To answer the research questions, the following specific research activities were planned:

1. For RQ 1 & RQ 2: Identify the FUR of software and ML solutions developed for underground mines, size them with the COSMIC-ISO 19761 international standard and document the corresponding measurement results;

2. For RQ 3 & RQ 4: Measure the size of the projects in a specific software release and then using the available information on effort for each project, develop a posteriori estimation models for these projects;

3. For RQ 5: Design an approximation technique-based ISO/IEC/IEEE 25148 standard on requirements engineering and an iceberg analogy to assign scaling factor to early functional requirements.

Methodology and Contributions

To answer the research questions and achieve these research objectives, the following research methodology was designed and adopted:

1. By using the information in the software user guides, the knowledge repository where the company archives its project documents and complementary information from the software developers:

a. Document the identified FUR of software solutions and AI algorithms already programmed by the researcher at this organization development site and;
b. Size and document their detailed functional size measurement results using the COSMIC – ISO 19761 standard, both from the software and from the data scientist viewpoints.

2. For a specific three-month software release:

a. Manual measurement with the COSMIC-ISO 19761 standard of the software projects within that release;
b. Constructing a posteriori estimation models for these projects and evaluating their performance as estimation models with:
   - regression analysis,
   - an EDA approach and the identification of outliers and extreme data through box plots and Grubbs test.

3. Develop an early software size approximation technique with function point sizing using two COSMIC case studies, through the classification of the functionality at three levels of increasing details and the derivation of scaling factors.

Thesis Structure

This thesis contains 5 chapters and is structured as follows:

Chapter 1 presents a literature review on software functional size, software size approximation and software sizing with the COSMIC-ISO 19761 measurement method.
Chapter 2 presents the identified FUR of developed software solutions with their corresponding size measurement results that needed to answer the RQ 1.

Chapter 3 provides answer to RQ 2 by presenting the identified FUR of developed ML algorithms to provide AI solutions for the company's underground mining projects.

Chapter 4 presents an analysis of three months of historical data for a software release, proposing estimation models using descriptive statistics to answer the RQ 3 and RQ 4.

Chapter 5 presents a new approximation technique on early sizing in software projects. This new early approximation technique of software sizing was applied to two COSMIC case studies to address to RQ 5.

The conclusion presents a summary of the results of this thesis research work, as well as the contributions and suggestions for future work.

CHAPTER 1

LITERATURE REVIEW

1.1. Software Functional Size

Software is composed of different components and the size of these components is dependent on the functionality required (Singh, 2017). According to ISO, functional size must be independent of quality and technical aspects of the software (Abran, Al-Sarayreh, & Cuadrado-Gallego, 2013) and it does not have any fixed proportional size relationship among those components (Singh, 2017).

Functional size measurement (FSM) is one of the techniques to measure the functionalities a software delivers to user (s) (ISO/IEC 14143-1 Standard, 2007) and it is applicable in software project management for different purposes: for example, to obtain system related technical indicators, development effort, manage scope in project, productivity studies, benchmark for future projects and normalize quality and maintenance ratios.

FSM is based on requirements and can provide size information early in the life cycle of the project (Ungan, Trudel, & Abran, 2018). When organizations collect the information from the past projects, they can build their own productivity and estimation models. This brings them the required data to measure their productivity from the past projects to find out their performance and how much it differs from the past projects. Having your own organization productivity model is a key advantage in the market and in the organization (Abran, 2015).

Research on FSM began with Allan Albrecht's invention of Function Point Analysis (FPA) in 1979 (Lokan, 2005). One of the most important phases of FPA is recognizing system components that bring functionality to users (Zelkowitz, 2005). Abran and Robillard (1996) investigated the structure of FPA considering mathematical operations and scale types in the measurement process.

(Fetcke, 2001) proposed a generalized representation of available FSM methods focusing on commonalities and differences and concluded that the approach could be applicable using the experience of past projects and allowing FSM to be automated. (Berg, Dekkers, & Oudshoorn, 2005) applied FSM to UML-based user requirements and proposed a requirement space in four refinement levels: in their study, the FPA and COSMIC CFP were used and three case studies were investigated: the results were relatively on mark for both methods.

1.2. Software Sizing with COSMIC Method

One of methods that is applicable to measure the functionality of a software from different domains is the COSMIC method (ISO/IEC 19761 Standard, 2011) accepted as an International Standard: 'ISO/IEC 19761 Software Engineering, a functional size measurement method'. COSMIC FSM measures the functional size of the software derived from its FUR. COSMIC Function Point (CFP) is a unit of measurement that represents the data movement for software and 1 CFP is equal to the size of a single data movement of a single data group (COSMIC, 2017).

Function point sizing (FPS) quantifies the functional size of software and is used for various purposes in software project management, including effort estimation, project planning, project monitoring, productivity studies and benchmarking (ISO/IEC/IEEE 29148 Standard, 2011), (Desharnais & Abran, 2003). This makes FPS a tool of choice for planning techniques that require an early view of the software to be developed. Despite being available earlier than other sizing methods, a precise application of FPS requires that the functional requirements of the software be detailed, and its architecture defined (Santillo, 2000).

FSM patterns are applicable to the COSMIC method and useful for measurers to address different issues such as: early sizing estimation, measurement errors and decreasing measurement effort. A study by Abran et al. (2002) discussed an estimation model for a maintenance project to implement functionality in the existing software. In their study, two

estimation models were investigated: a regression model with functional size only and a multiple regression model with two independent variables.

(Trudel, 2012) investigated the efficiency and effectiveness of COSMIC to identify defects in functional requirements. The FSM brings added value and saves the rework cost if the identified defects by measurer are corrected early in the development cycle. In another study (Trudel, Desharnais, & Clout, 2016) focuses on the application of FSM pattern with real-time embedded systems and suggests the potential benefits for the industry.

(Huijgens, Bruntink, Deursen, Storm, & Vogelezang, 2016) presented an exploratory study on FSM based on code and discussed challenges, obstacles, and opportunities the experts experience on software project code. 87% of the FSM experts have the same opinion that the FSM is an important tool for decision making and COSMIC with 25% is the most preferred FSM method.

Today, many software development companies have adopted the agile process. A study by (Hussain, Kosseim, & Ormandjieva, 2013) presented an approximate COSMIC functional size from informally written textual requirements and build a historical database by manually measuring the COSMIC functional size from textual requirements. (Sellami, Haoues, Borchani, & Bouassida, 2018) studied functional changes in agile projects and proposed a guide for decision makers in software development projects when they encounter changes in software requirements during agile sprints. The tool is based on user stories and the COSMIC sizing method and examined 15 software development projects.

COSMIC method is applicable in different domains such as (COSMIC, 2017): business applications, real-time software and several case studies are proposed by the COSMIC group in the mentioned domains, but not with AI applications. (Soubra, Abran, Stern, & Ramdan-Cherif, 2011) proposed an FSM procedure for real-time embedded software based on COSMIC and documented the requirements with the Simulink tool. The automated tools reduce the measurement variances caused by interpretations made by different measurers. In parallel to

this research project, Lesterhuis and Abran (2019) used the COSMIC method to extract the ML system requirements of an ML image classifier software in a neural network: it presents the ML functionalities and allows the ML expert to have more freedom to address additional ML challenges.

1.3. Early Sizing and Size Approximation

Early in the software development life cycle (SDLC) the software requirements are not fully described, and it is unrealistic to expect this set of requirements to describe the full scope of functionality of the software as a whole. There are obviously many functional unknowns early in the life cycle. Over time, more detailed requirements will be identified and part of them might be changed as the software development life cycle progresses. When the requirements are fully recognized and understood by the risk management experts, the uncertainty dwindles: this was graphically represented as a Cone of Uncertainty by (Boehm & Abst, 2000) - Figure 1.1:

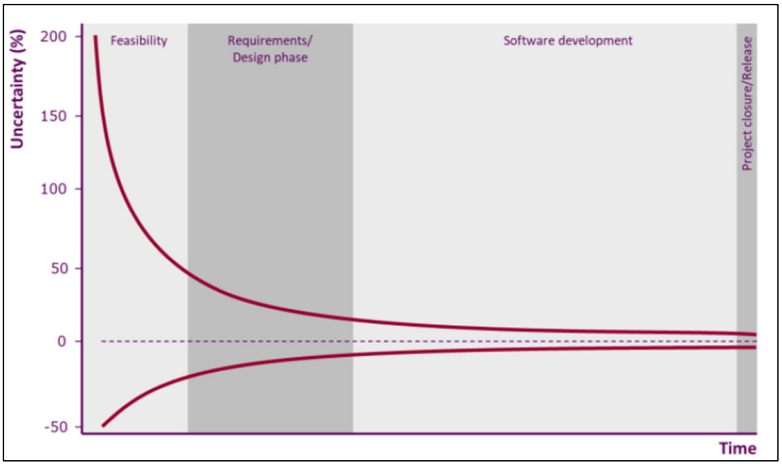

Figure 1.1. Cone of uncertainty (Boehm & Abst, 2000)

where the uncertainty progressively decreases as the project progresses and additional more precise information becomes available where:

At time = project closure, everything is known about the software. When all the requirements information as implemented in the software code is available, the size can be measured with high accuracy.

At time = feasibility study, the information available on the requirements is typically high level and without much details. From a functional perspective many un-knowns remain. With imprecise and incomplete inputs, there cannot be accurate measurement. Rather, the expected functional size at project closure can only be estimated and, as with any type of estimate, this comes with a range of uncertainty that will vary depending on the quality, completeness and stability of the set of requirements (Yılmaz, Ungan, & Demirörs, 2011);

At time = between feasibility and project closure, the completeness of the information and requirements will progressively improve, which will impact the measurement results until all requirements have been specified in detail.

At various points later in the project life cycle, the lists of software functions are detailed, programmed, tested and implemented; the same applies to system non-functional and quality requirements, some of which may later be allocated to additional software functional requirements (Al-Sarayreh, 2011), (Abran, Al-Sarayreh, & Cuadrado-Gallego, 2013), (Meridji, Al-Sarayreh, Abran, & Trudel, 2019), (Al-Sarayreh & Abran, 2010).

Some COSMIC-based size approximation techniques have been proposed that can be used as an input for early management activities (Desharnais & Abran, 2003), (Santillo, 2000), (Vogelezang & Prins, 2007), (Almakadmeh & Abran, 2013), (Almakadmeh, 2013), (COSMIC, 2015). The bands-based techniques were empirically investigated by Lavazza and Morasca (Lavazza & Morasca, 2019). Seven approximation techniques have been proposed by (COSMIC, 2015) including: 1) average functional process, 2) fixed size classification, 3) equal size bands, 4) average use case, 5) functional size measurement patterns, 6) early and quick COSMIC sizing and 7) easy function points. The nature of the gaps and source of them between earlier size and final size is discussed in an exploratory analysis study by Abran et. al. (2018)

and how the hidden functionality can cause a large gap between the initial size and final size of a piece of software (Ungan, Trudel, & Poulin, 2017).

In this research project, we will conduct size measurement using the COSMIC method in the context of AI and real-time embedded software for underground mines. Mining is an industry with great traditions where change is at times very gradual. There is less literature about the software size in the field of software solutions for underground mines whereas the software devices have brought many innovations in the mining sector including facilitating data collection, improving safety, automating services, and making tasks and procedures integrated. The lack of literature and historical data as well as documentation in the specific context of software for underground mines have brought challenges for engineers and software developers when they want to implement and size a new software project with requirements that are imperfect and incomplete.

The same is true in the AI applications and ML algorithms in the specific industry contexts without relevant measurement tools to tackle to the problem at hand. There is a lot of literature in the context of AI and ML algorithms and the mathematical aspects of them; however, there are literature shortcomings about the functional size measurement of software to be developed in order to implement an ML algorithm.

CHAPTER 2

MEASUREMENT OF SOFTWARE FOR UNDERGROUND MINES

This chapter presents the COSMIC measurement results of the FUR in the domain of real-time embedded software in underground mines and explore whether or not the COSMIC standard could be used for real-time embedded software which addressed to RQ 1. The research objective in this chapter is to size with COSMIC and extract the FUR of software solutions and then document the measurement results.

The software is deployed in the mining hardware devices to collect data and manipulate the big data for monitoring, operating, predicting and maintenance. The FSM had never been used in this software development company with actual data from the industry projects, and no documentation of the FUR and software size information were available. The FUR had therefore to be identified from the available software user guides and through a number of measurement iterations, more FUR were revealed by software developer's contributions.

Sections 2.1, 2.2 and 2.3 present an introduction to three software in underground mines. Section 2.4 presents the scope and purpose of measurements along with context diagrams of these three software. Sections 2.5 and 2.6 present the software requirements and the measurement results. More details on software specification and measurement results are available in Appendix I.

2.1. Recorder Unit software

The recorder unit is a stand-alone data recorder and wireless router installed in the mine that records, stores, and transmits data collected from sensors to the communication server. There is currently a total of 195 sensor's readings received by the recorder. It is compatible with mining equipment and well suited to the harsh mining environment. The recorder software consists of three elements:

1. Recorder board (BOT): acquires unformatted data every half second (time is configurable in configuration file) transferred through a Bus cable from different sensors.

This configuration file (with ibc extension) contains the recording and data processing parameters and is generated by a desktop application called configuration tool. The configuration file is sent directly to the recorder board using USB or other applications and is stored on hard disk in BOT. The settings of configuration file in it determines what such streams mean and what data should be recorded based on the configuration. For instance, the settings in configuration file indicate recoding of the engine pressure coming from sensors.

2. Router board (TOP): receives the data from BOT and stores them on hard disk. A Bus cable connects the BOT to the TOP. When the network connection is available, it transfers the data to the communication server.
3. Two hard disks: to store the data and retrieve it when it is requested by the software.

2.2. Cap Lamp Software

Cap lamp is one of the important personal protection equipment in mines. New IoT cap lamps are improving miners' safety while they are working under the ground and they became one of the safety devices that provides communication with outside of the mine. It allows mines and industries to monitor their personnel, enhance safety and save lives in hazards.

The cap lamp is placed in a charger base and it has several buttons which trigger the software to start performing. The cap lamp is connected to a set of wireless nodes for transmitting signals to the server on the surface. The cap lamp has the following software elements:

1. Setting the clock (time) and serial numbers;
2. Automatic self-test mechanism;
3. Display network status and LEDs;
4. Send distress signal, man down alarm and request to cancel the alarms to the server;
5. Receive the acknowledgements and evacuation notice from server.

2.3. Tag Reader Software

Tag reader is a device that provides association between personnel tag (RFID cards) and personal safety device (e.g. cap lamp). Before going down into the mine, the miners should scan their personnel tags followed by the personal safety device that they are using for the shift. Tag association is an important safety procedure and unassigned personal safety device alarms are raised when personal safety device tags are tracked underground. Associations are made at the beginning of a shift and then broken at the end of the shift. The tag reader has the following software elements:

1. Scan the RFID and serial number (tag ID) of the personal safety device;
2. Receive the acknowledgements and the messages from the server through the tag reader gateway;
3. Ability to restart the association process.

2.4. Scope and Purpose of Measurement

The purpose of measurement is to determine the amount of the functionality of software based on its FUR. The scope of the measurement is the detailed functionalities the software delivers to user(s).

The software interacts with its functional users across the software boundary. Some COSMIC definitions adapted from COSMIC guideline (COSMIC, 2017):

A functional user is a type of user that acts as sender or recipient of data and identified in the FUR of software being measured. Table 2.1 presents the functional users of the software.
A functional process is a unique elementary part of the requirements initiated by a functional user.

A persistent storage is a storage that allows the functional process to store the data and retrieve data by another functional process. The software interacts with its persistent storage within the software boundary.

Table 2.1. The functional users for the three embedded software measured

Software	Functional users
Recorder unit	- USB - Sensors - Communication server
Cap lamp	- Charger base - Buttons (lamp, Clock/ display and Lateral buttons) - Accelerometer sensor - LEDs (main, auxiliary, green and red) - Wireless node
Tag reader	- RFID - Personal safety device - Tag reader screen - Tag reader gateway

In the recorder software, the hard disks are persistent storage that are within the software boundary.

2.5. Software Requirements

This section describes the requirements of the software to be measured including the software functionalities and the interaction with its functional users. In the following, the context diagram of the software is displayed: the context diagram defines the boundary of software and its relationship between the functional users (COSMIC, 2017).

2.5.1. Recorder Unit Requirements

This is the set of requirements at the software level. Figure 2.1 depicts the context diagram of the software and its interaction with the functional users.

Requirement 1. Prepare the configuration file.

The software needs to receive and store the configuration file on the hard disk in BOT.

Requirement 2. Record data.

The software receives unformatted streams of data from the sensors through Bus cable and records the required data according to the configuration instructions.

Requirement 3. Store the data.

The recorded raw data need to transmit to the TOP and stores on the hard disk.

Requirement 4. Transfer the data.

When the connection between the recorder and communication server is established, the raw data segments transferred to the communication server through the Bus cable for data manipulation.

Requirement 5. Delete the data.

Once the communication server downloaded all the raw data segments, a reset command is sent to delete the data on recorder.

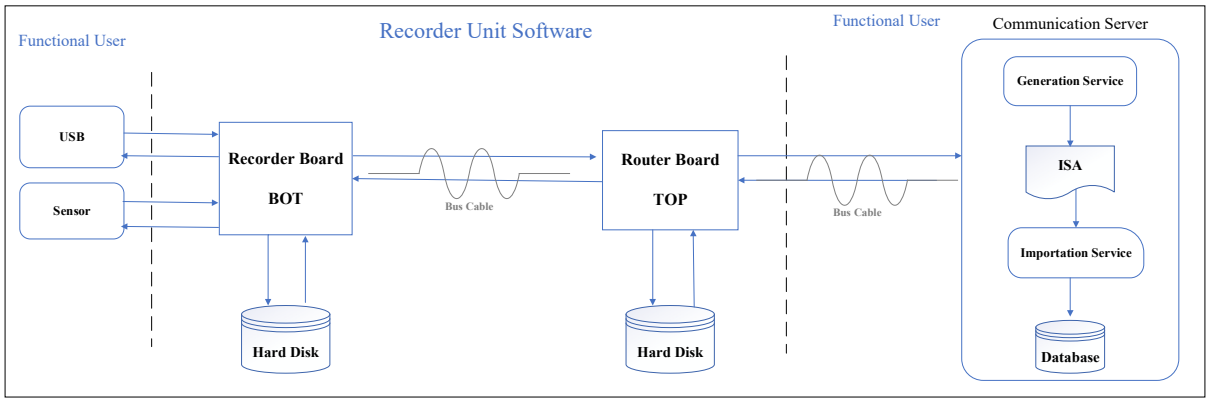

Figure 2.1. Context diagram of recorder unit software

1. The software of recorder has two layers: BOT and TOP;

2. The Bus cable between the BOT and TOP is responsible to transmit data groups from layer one to layer two and the other Bus cable transmits the data from the software to the communication server;
3. Hard disks in BOT and TOP are persistent storages where data groups are stored and can be retrieved when they are requested by the software;
4. Communication server is another software application where the data is manipulated.

2.5.2. Cap Lamp Requirements

This is the set of cap lamp requirements at the software level. There are pre-requisites to set up the cap lamp software before starting to use it in the mines. Figure 2.2 depicts the context diagram of the software and its interaction with the functional users.

Pre-requisite 1. Setting the clock.
The time needs to set according to time zone where the cap lamp is used. To set the time, the person needs to use the lamp and clock/display buttons.
Pre-requisite 2. Setting the serial number of the cap lamp.
The serial number of the cap lamp is a unique number assigned to each cap lamp. It is defined before in the server and stored in database. To enter the serial number, the person needs to use the lamp and clock/display buttons.
Pre-requisite 3. Check the cap lamp LEDs.
The cap lamp has two LEDs: main and auxiliary LED. To use the LED, the person needs to press the lamp button.
Pre-requisite 4. Check the cap lamp information.
The cap lamp holds the information of serial number, current time, and battery status. To check the information, the person needs to press the clock/display button.

Requirement 1. Cap lamp self-test mechanism.
The mechanism is started by a human (miner) taking out the cap lamp from the charger base. All the cap lamps need to be plugged into charger after each shift to get fully charged for the

usage of personnel. The software in the cap lamp starts a self-test automatically and the main LED starts flashing. The lateral buttons stop the mechanism.

Requirement 2. Connect to the available wireless nodes.

The cap lamps must be under the coverage of wireless network in the mine, to be able to send and receive signals. If the cap lamp connects to the node, the green LED blinks, otherwise, the red LED blinks.

Requirement 3. Send distress signal.

The distress signal is sent by pressing both lateral buttons simultaneously. An emergency request for help transfers to the server through the wireless nodes and red LED starts blinking rapidly (twice per second).

Requirement 4. Send man down alarm.

The man down alarm is sent when the accelerometer sensor in the cap lamp does not detect the miner's movements for 90 seconds. After 60 seconds of not detecting any movement, the main LED of the cap lamp starts blinking, and if the cap lamp remains immobile thorough a full 90-second, a man down alert is generated and sent to server while the main LED is still blinking.

Requirement 5. Cancel the man down alert.

Movement alone does not cancel the man down alarm. To cancel the man down alarm, the miner should press the lateral buttons. The main LED stops blinking, and a cancelation signal is sent to the server through the wireless nodes.

Requirement 6. Respond to evacuation notice.

The operator behind the console can send an evacuation notice to all the enabled cap lamps currently online. When the evacuation notice is received by the cap lamp, the red LED turns on and the main LED blinks every 30 seconds for a period of 5 seconds.

Requirement 7. Alarm acknowledgment by console.

When the distress signal is received by the server, the server logs the time, miner's location, and cap lamp serial number. Then:

1. The distress signal appears on console and the distress window pups out;
2. The cap lamp red LED blinks slowly (once every 2 seconds) to indicate that the distress alarm is received at the surface;
3. Once the alarm is acknowledged from the panel in console, the cap lamp red LED turns off.

Requirement 8. Alarm acknowledgment by cap lamp.

When the evacuation notice is received by cap lamp, the miner responds to that by pressing the lateral buttons and cap lamp sends a confirmation message to the console. The main LED keeps blinking for 5 seconds every 30 minutes, and the red LED remains on until the evacuation ends.

Requirement 9. Stop the evacuation notice.

If the emergency clears, the operator stops the evacuation notice and both the main LED and red LED stop blinking.

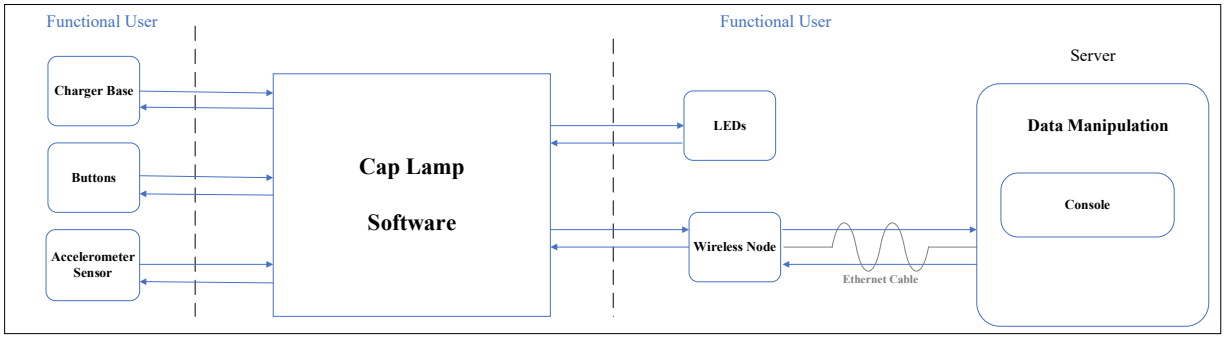

Figure 2.2. Context diagram of cap lamp software

1. The software of cap lamp has one layer;
2. Ethernet cable between the last wireless node and the server is responsible to transmit data to the server.

2.5.3. Tag Reader Requirements

This is the set of tag reader requirements at the software level. There are pre-requisites before starting to use the tag reader. Figure 2.3 shows the context diagram of the software and its interaction with the functional users.

Pre-requisite 1. Turn on the tag reader by pressing the on/off button.

Pre-requisite 2. The tag reader should be connected to the tag reader gateway and the server. The tag reader is connected to the gateway by a wire and the gateway is connected to the server through a TCP/IP network.

Requirement 1. Connect to gateway.

While the tag reader establishes the connection between gateway and server, the following message displays on the tag reader screen: “connecting to server”. Once both connections established, the tag reader display the following message: “tag reader is ready, please scan your personnel tag”.

Requirement 2. Scan the personnel tag ID (RFID).

The software expects the personnel tag to get scanned first. The miner scans his RFID card and waits for the validation from the server.

Requirement 3. Scan the tag ID (serial number) of personal safety device.

The software proceeds to scan the personal safety device after scanned the RFID. Once the serial number is scanned, the software sends it to the server through the gateway for the validation.

Requirement 4. Restart the association process.

If a wrong tag ID is scanned, the tag reader restarts the association process and the miner needs to restart the procedure.

There is a 10-second timeout after scanning the personnel tag ID. If no device is scanned for this period of time, the tag reader restarts the association process.

If the tag reader loses connection with the gateway, the tag reader restarts the association process.

Requirement 5. Confirmation/error message.

Once the server received the serial number, an association is created between the RFID and the serial number and the successful association message is sent to the tag reader through the gateway.

If a personal safety device is scanned instead of RFID, the following error message shows up: “it is not a personnel tag”.

If personnel tag is scanned instead of personal safety device, the following error message shows up: “it is not a personal device”.

If undefined tag is scanned, the server does not recognize the tag and the following error message shows up: “unrecognized tag”.

Note: after the shift ends, the miner places the personal safety device into its charger base. This generates a disassociation message that clears the association between the personal safety device and employee RFID. The disassociation message is sent to the server through the available wireless network.

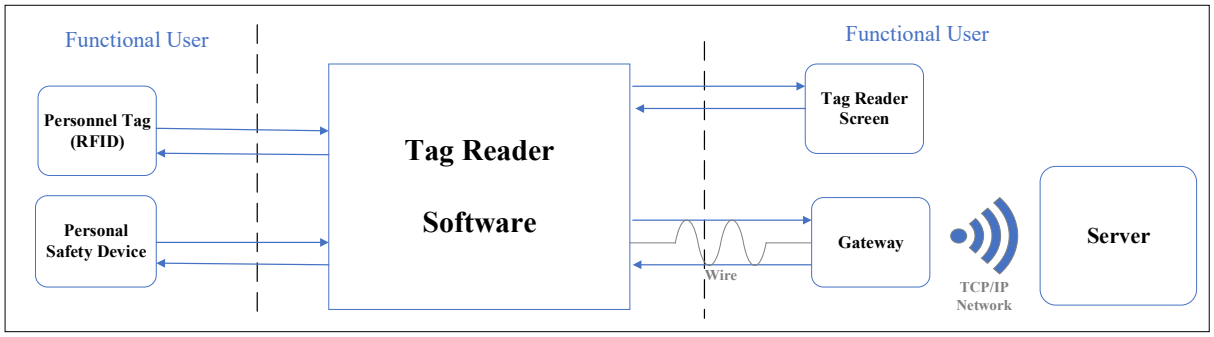

Figure 2.3. Context diagram of tag reader software

1. The software of tag reader has one layer;
2. The gateway is connected to the tag reader by a wire. TCP/IP network is for transmitting data to the server;
3. The server is another software application outside of the boundary of measurement.

2.6. COSMIC Measurement Results

This section provides the measurement results of the three software in Table 2.2 grouped by the type of data movements.

The unit of size measurement is COSMIC Function Point (CFP). According to the COSMIC method, there are four types of data movements (COSMIC, 2017):

Entry: moves a data group into a functional process from a functional user.

Exit: moves a data group out of a functional process to a functional user.

Write: moves a data group from a functional process to persistent storage.

Read: moves a data group from persistent storage to a functional process.

Based on the set of requirements for each software, the functional processes related to each FUR are identified and the size of them are measured using the COSMIC method.

Table 2.2. COSMIC measurement results of software solutions for underground mines

Software	**Data movements**				**COSMIC size**
	Entry	**Exit**	**Write**	**Read**	
Recorder unit	5	2	3	2	12 CFP
Cap lamp	12	17	0	0	29 CFP
Tag reader	7	7	0	0	14 CFP

The COSMIC-ISO 19761 method can be used to measure the functional size of the real-time embedded software deployed in underground mines (answer to RQ 1). The obtained COSMIC sizes can be used as one of the independent variables for the construction of priori estimation models in the companies. At the beginning of the project, once the other estimated variables are known (such as effort), the project leaders can use these size information to build their own estimation models (software effort/cost/timelines) with greater certainty and control. It makes it possible to quantify the functions the software delivered in a standardized, repeatable, and objective way, and independently of development technologies.

CHAPTER 3

MEASUREMENT OF AI SOLUTIONS FOR UNDERGROUND MINES

There is not yet any reported used of COSMIC sizing with AI and ML projects in organizations. The aim of this chapter is to answer the RQ 2 and demonstrates that the FP technique can be used in the ML context. This chapter reports on our usage of COSMIC for sizing two ML algorithms and an AI dashboard developed using Python programming language to provide AI solutions for underground mines problems. The measurement is performed for two different viewpoints:

1. Software viewpoint which is the classical viewpoint of measurement that describes the functionalities the software does and;
2. Data scientist viewpoint which describes the software tasks required to be developed by the data scientist for using this AI solution.

The AI solutions are new in this software development company and there has not yet been any official release for the AI application. In practice, the lack of formal FUR documentation as well as size information, and the unavailability of the experts familiar with previous AI projects in the company necessitate to identify the FUR during the coding process and then fully document them when the coding and testing are accomplished.

The research objectives in this chapter is to size with COSMIC the functional requirements of developed ML algorithms that are already programmed and specific to AI projects and applicable for the company's underground mines problems. The size measurement for two different viewpoints can assist the data scientists to obtain insights of what needs to be coded to develop a specific functionality in the software, how to size them, and how to use this size for effort estimation purposes. The method selected in this research to measure the functional size of the software is the COSMIC ISO 19761 standard.

Section 3.1 provides an explanation of the AI solutions in mining projects and introduces the ML algorithms and features of an AI dashboard. Section 3.2 presents the scope and purpose of measurements along with the context diagram of the software. Section 3.3 presents the measurement results.

3.1. Overview on Mining Problems and AI Solutions

The data obtained by the recorder are stored in the database in the server and the data scientists have access to the stored data for analysis purposes. One of the sources of data is the SQL database: the data scientist can read the data by querying in Python and the other source of data is HDF files stored per product per month. Advanced statistical analysis using ML is used to identify patterns in the data and build AI solutions.

3.1.1. Event Detection

Event detection is used in mines for multivariate time series data to recognize the event triggers. The purpose of events detection is to identify the occurrence of events to monitor an environment. The goal of events detection is to recognize the problems and learn how to detect the occurrence of these events for several purposes:

1. Inspection: it is almost impossible to inspect in real-time the vehicles in the mines to make sure they are working, and that they are not broken or near to get broken down. Detecting events helps to recognize problems and fix them on time before they occur;
2. Safety: from the human resource safety point of view, it is not possible to check on them visually to assure the workers are doing well and not in major danger. In practice, all mine workers wear cap lamps and, the software in the cap lamp detects the worker's movements and sends signals to the operator. If a danger happens, the issue is detectable and rescue procedures can be initiated and carried on;

3. Work productivity: as in other industry sectors, productivity is a critical issue in the mining industry. Depending on what is measured for the productivity, the event detection can help to identify the reasons why the productivity has increased or decreased over a period of time. In the software development company, one of the KPIs for productivity is the Standard Production Time (SPT) for vehicles as defined by AFNOR.

For example, when the vehicle is in standby mode, the periods need to be recognized using ML solutions. The time taken activities in mining are broken down in Figure 3.1 and the standard time slot for SPT is specified.

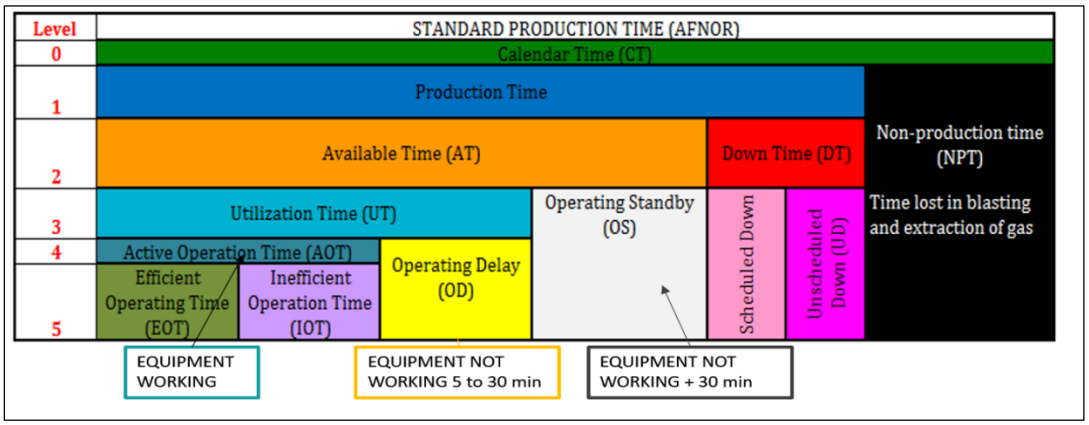

Figure 3.1. Definition of SPT – AFNOR

3.1.2. K-Means Clustering - Detecting Idling Speed in Vehicles

The K-Means algorithm is one of the unsupervised ML algorithms that group similar data points together and discover underlying patterns (Garbade, 2018). The algorithm identifies randomly selected centroids, allocates every data point to the nearest cluster, and then repeats the calculations to optimize the positions of the centroids.

The idling status refers to vehicle's engine functioning while the vehicle is not in motion (e.g., a taxi driver is stopped at a red light):

1. When the vehicle is off (Down Time in Figure 3.1), the engine speed is zero;
2. When the driver starts the vehicle on, the engine speed goes up.

Depending on the type of work that the vehicle is doing (e.g. loading, unloading, and dumping) the engine speed is variable (usually up to 2400 r/min):

1. When the vehicle is not in motion and the wheels are stopped (the gear is neutral), the engine speed indicator shows a specific engine speed: this is the idling state where:
2. The engine speed is positive, and the wheel speed is zero (Engine speed > 0 (r/min) & Wheel speed = 0 km/h).

As defined in Figure 3.1, operating delay (OD) and operating standby (OS) activities represent an idling state of vehicle: they both indicate that the equipment is on but not working and the only difference in their definitions is how long the equipment has not been working.

There are different reasons why a vehicle does not work for a long time such as: malfunction, unexpected safety issues happened to the driver and sensors deterioration. Identifying these reasons will help to find out what occur underground when the vehicle is idle, in particular since real-time inspection underground has many challenges.

Furthermore, for productivity analysis, these idling periods have to be excluded from the production time formula and one way to exclude those periods is to identify such idling periods and events (here, the engine speed) and removing such data from SPT calculation.

Before running the algorithm through the data, a series of conditions have to be set based on initial assumptions (here: Engine speed > 0 (r/min) & Wheel speed = 0 km/h) in order to select the potential data points. The algorithm identifies the centroid(s) which is (are) the idling engine speed in vehicles. The obtained values can be used as an input value for monitoring and optimizing purposes in other coding calculations such as:

1. Calculate OD and OS in each shift;
2. Monitor the driver’s status while doing job underground.

The K-Means algorithm has been chosen for the following two reasons:

1. Type of dataset: unlabeled data (i.e., data without defined categories or groups);
2. K-Means is computationally faster and simpler compared to other clustering algorithms.

It is a hard task for the data scientist. The data scientist knowledge and expertise are required to find out the best fitted algorithm to implement.

3.1.3. Predictive Maintenance

The purpose of predictive maintenance is to determine a condition and a point in a future time to estimate when the maintenance should be performed. Most of the vehicles in the mine are only installed once and they stay there for years until the work finishes in that specific area. It is almost impossible to transfer the vehicles in and out every time for maintenance. In addition, every day or random checking inside the mine is not a wise, utilized and cost-benefit way due to safety and performance issues.

So, robust predictive and data analytics are needed to anticipate the future points based on past trends. Common ML algorithms are largely available to make prediction based on the past trend of data. Here, one of the elements that needs to be predicted is the air filter restriction in the vehicles: an air filter restriction is a device to measure the air induction and it is installed on vehicles (trucks and loaders) in the mine.

3.1.4. Linear Regression - Predicting Remaining Air Filter Restriction

Linear regression is a form of regression analysis that shows the relationship between the independent variable and dependent variable. In statistics, it is used when an independent variable and a dependent variable have a linear relation and as the amount of data grows, the model learns to make more accurate predictions.

The air filter restriction indicator measures the air induction. As the restriction progressively decreases, the indicator moves up to the maximum point: when it reaches the maximum, the air filter should be changed, and the indicator reset to zero.

In the mines, the air filter device is installed on vehicles. The recorder collects the air filter data and sends them to the server. Prediction is needed to forecast when is the time to change the air filter restriction device: the air filter restriction data and the total engine hours data are used to build the regression model for estimating the remaining hours of the air filter.

There are two reasons why the linear regression is chosen for this case:

1. In each air filter cycle, the air filter restriction variable and total engine hours have linear positive relationship as in Figure 3.2 The red line in each cycle shows such relationships. Since the number of data points are huge, they overlap on each other.
2. It is computationally faster and simpler compared to other prediction models for this case.

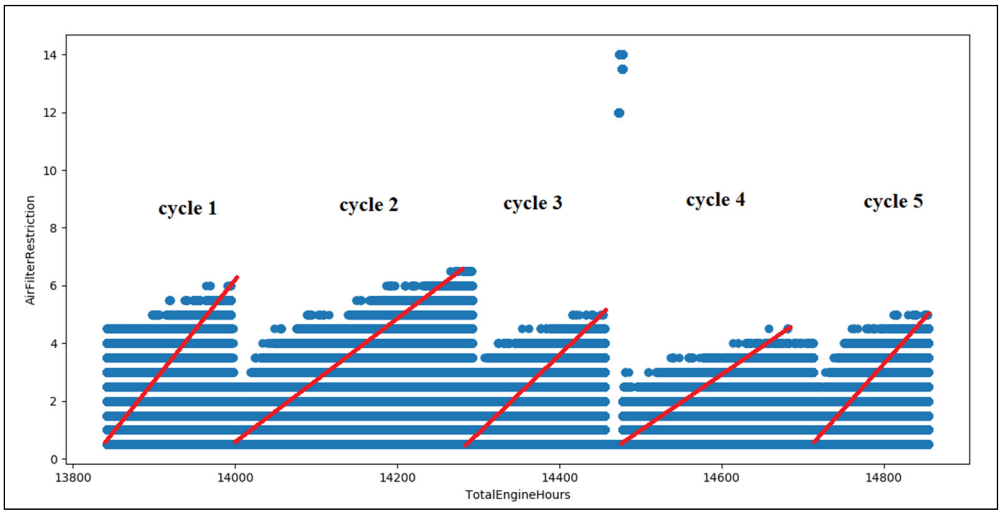

Figure 3.2. Behavior of air filter and total engine hours in each cycle

3.1.5. AI Dashboard with Streamlit

Streamlit is a framework for ML projects which allows to create a data app in Python and run and browse data in real time on the web browser. It is a dynamic, highly interactive, and quickly built and deployable app with various widgets. Less code is needed in Python compared to other programming languages like HTML.

According to the needs of AI teams, different features can be coded with the purpose of inspecting data and build ML models on them. The main advantages of the AI dashboard are:

1. It gives the data science workers an interface, particularly in data visualization;
2. The developed app allows to upload data and provides quick overview of statistical analytics and visualization on a web browser environment for the data scientists in real time;
3. It avoids re-working and re-coding each time when the data scientists want to observe the data;
4. It is a user-friendly tool for someone without programming knowledge whose wants to see a display of data, such as senior managers and stakeholders.

The user interface allows to select, choose, upload, and execute data without touching the script and the staff using the same domain in a company can connect to it by the URL on their Google Chrome web browser. Here, the data loader feature, statistics summary, equation, filter selection and data visualization feature were developed.

3.2. Purpose and Scope of Measurement

Two different viewpoints are discussed here, and the measurements have been carried on according to two purposes and scopes of measurement.

Software viewpoint: The purpose of the measurement is to determine the amount of the functionality of the software.
The scope of the measurement is the software requirements to implement an ML algorithm. The requirements specify the functionalities that the software delivers to its users for that specific algorithm whether or not some of them are done automatically by Python or come from reused functions.

Data scientist viewpoint: The purpose of measurement is to provide size data for data scientists for their own AI development work.
The scope of the measurement is the data scientist requirements to implement an ML algorithm. It states the data scientist tasks in Python to implement an algorithm.

According to the main five stages of AI life cycle in Figure 3.3, which indicates the processes data scientists follow in AI projects, the aim of data scientist viewpoint in this thesis only positions the software development tasks to implement the ML algorithm and not the full development of AI software.

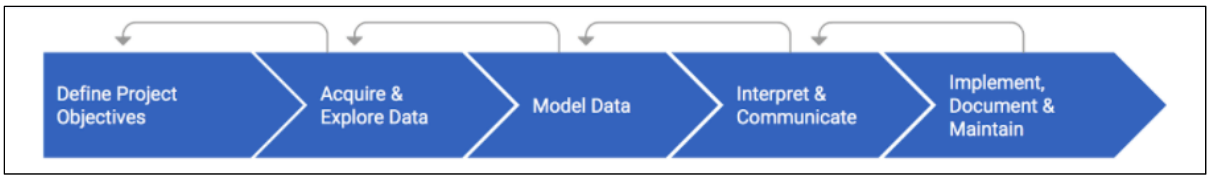

Figure 3.3. Machine Learning life cycle – The DataRobot website (https://www.datarobot.com)

It is to be noted here that these two measurement processes were done after the software development has been completed: here, the FUR used were available after the coding process and there was no uncertainty and ambiguity in these implemented requirements.

The context diagram of the AI software is displayed in Figure 3.4 where:

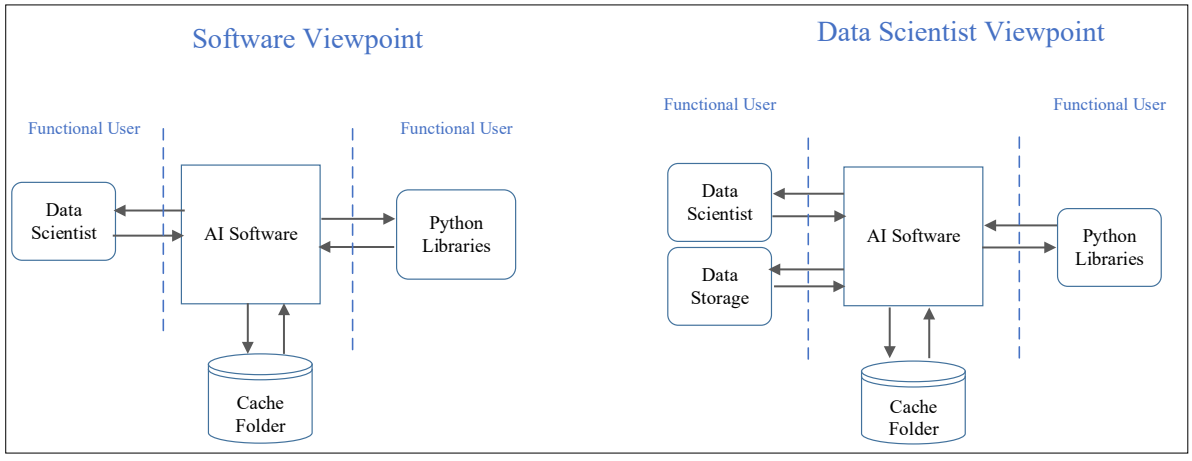

Figure 3.4. Context diagram of the AI software – two viewpoints

1. The dashed lines show the software boundary;
2. The software to be measured is the AI software (here, two ML algorithms and an AI dashboard) that is going to be written in the code platform;

3. The data scientist who does the coding and implementation part in the software is the human functional user (he sends and receives data from the software);
4. The Python libraries which are pre-programmed as another software application in external libraries are also considered as functional users;
5. The data storage is where the data is maintained. It can be stored as file in computer, server, and a SQL database and it is the functional user in data scientist viewpoint when (s)he reads the data from storage into Python;
6. The cache folder is the persistent storage stored under a user directory. It is a temporary storage which saves the current outputs as long as the coding program is running.

3.3. Software Requirements

This section describes the FUR of the AI software to be measured from the two viewpoints:

1. The software viewpoint includes the functionalities the software does and;
2. The data scientist viewpoint includes the data scientist requirements to develop the AI software.

3.3.1. K-Means Clustering Requirements

The set of requirements at the software level for the K-Means clustering algorithm from software viewpoint and data scientist viewpoint are presented in Table 3.1 and Table 3.2.

Table 3.1. Software requirements - K-Means clustering algorithm

Requirement 1. Receive the inputs.
The number of clusters (K) are given to the software.
Requirement 2. Apply the K-Means function.
The K-Means function is called from the Sklearn.cluster library with a defined K (here: K=2) to generate the model. Fit the dataset to model and obtain the centroids. The model generalizes to the given datasets using the Fit command.

Requirement 3. Obtain the centroid(s).
The list of centroids is obtained which is the engine speed values when the vehicle is in the idling state.
Requirement 4. Plot the scatter graph and specify the centroids in different colors.
The idling speeds are displayed on graph.

Table 3.2. Data scientist requirements - K-Means clustering algorithm

Requirement 1. Import Python libraries.
The libraries are a collection of functions and methods that allow to perform many actions without writing the code. The required libraries are: - Numpy: for numerical operations, - Pandas: for data manipulation and analysis, - Matplotlib: to design the graphs and charts, - Sklearn.cluster and K-means: for clustering algorithm.
Requirement 2. Upload the raw data and create a data frame using the required columns.
The data scientist uploads the raw data from data storage using pd.read function. Depending on the size of the data, the data scientist decides how many data files to upload. The data scientist enters the title of the columns as input to the pd.read function. Here the columns are: Date/time, engine speed, wheel-based speed, and dump positioning.
Requirement 3. Create two empty columns in the existing data frame for further calculation.
Two empty columns are created to put the results in the further calculation: state and idle.
Requirement 4. Process the raw data.
The unneeded data should be selected and removed from the data frame. - Engine speed = 0 (the engine is in offline/down mode) and wheel-based ≠ 0 (the vehicle is not in motion), - Dump positions: dump refers to a placement of vehicle when unloading the material. When the vehicle is in dump position, the wheel-based speed is equal to zero, but the angle of dumper bed positioning is more than 40^0. - Higher values of engine speed: make a set of comparison and label the data with 1 when the vehicle does not work and label the data with 0 when the vehicle works. The results will be put in the state column. When the vehicle is working (e.g. loading, unloading, and dumping), the engine works with maximum power and when the vehicle is not 'working', the engine speed should be lower than the engine speed at which the vehicle is working. - Apply the constraint for idling period is defined (e.g. Idling > 5 minutes) To determine the immobility time, the data scientist writes a sum function to sum the values labeled with 1 in state column, with this difference that when it is reached to a value labeled with 0 in the state column, resets the summation to zero and start counting again. The results are added in idle column.

Requirement 5. Apply the K-Means function with defined K and fit the dataset to the K-Means model to obtain the centroids.
The data scientist enters the number of K clusters (here: K=2). A cluster is a collection of data points aggregated together because of certain similarities. Fitting a dataset means that the ML model generalizes the given datasets to data similar to those on which it was trained.
Requirement 6. Plot the scatter graph and specify the centroids in different colors.
The data scientist plots the graph and specifies the centroids. The centroids are the outputs which are the engine speed values.

Note for requirement 2: data cleaning and index changing need to be done before by the data scientist. The procedure is not specific to this context and not described in these software requirements: therefore, its measurement is not included.

The scatter plot in Figure 3.5 displays the idling speeds in vehicles in different color. As mentioned in requirement 5, the idling speeds are the obtained centroids which are the outcome of K-Means algorithm. In this example the idling speeds are [646 , 699 r/min] and the wheel based speed is equal to zero which means the vehicle is not in motion.

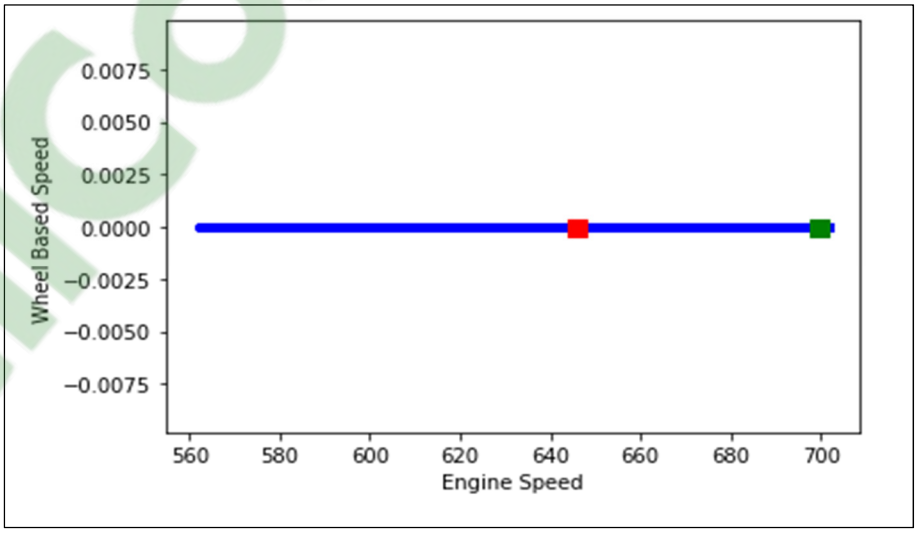

Figure 3.5. Example of identified clusters using K-Means algorithm

3.3.2. Linear Regression Requirements

The set of requirements at the software level for the linear regression algorithm from the software viewpoint and data scientist viewpoint are presented in Table 3.3 and Table 3.4. The details of functional process and measurements in the two mentioned scopes along with data groups are provided in Appendix II.

Table 3.3. Software requirements - Linear regression algorithm

Requirement 1. Receive the regression equation.
The regression equation is given to the software. $Y = A * T + B$ (A is the regression co-efficient and B is the intercept)
Requirement 2. Receive the threshold.
The threshold is the maximum air filter among all cycle ($T = MAX$ *(Air Filter Restriction)*). Any other required thresholds can be given here.
Requirement 3. Receive the regression variables and training sets.
The dependent variable and independent variable and training sets are defined. Use the train split method or feed the whole historical dataset as training set.
Requirement 4. Apply the linear regression to the training set.
Call the regression algorithm from the scikit-learn library. Then, fit the linear model to the training datasets using Fit command.
Requirement 5. Obtain the regression equation.
Request the software to produce the calculated remaining hours.

Table 3.4. Data scientist requirements - Linear regression algorithm

Requirement 1. Import Python libraries.
The libraries are a collection of functions and methods that allow to perform many actions without writing the code. The required libraries are: - Numpy: for numerical operations, - Pandas: for data manipulation and analysis, - Stats: for the outlier identification, - Scipy: for using the mathematics and scientific computing, - Sklearn: for linear regression algorithm.

Requirement 2. Upload the raw data and create a data frame using the required columns.
The data scientist uploads the raw data from data storage using pd.read function. Depending on the size of the data, the data scientist decides how many data files to upload. The data scientist gives the title of the columns as input to the pd.read function. Here the columns are: date/time, air filter restriction, total engine hours and engine speed.
Requirement 3. Create one empty columns in the existing data frame for further calculation.
An empty column (Hours) is created to put the results in the further calculation.
Requirement 4. Process the raw data.
The unneeded data should be selected and removed from the data frame. - Total engine hours = 0 (the engine is in offline mode), air filter < 0 (the air filter does not work) and engine speed more than quantile (0.25) or average, depends on data scientist judgment (e.g. engine works with power, so the air filter works), - Outliers are removed using z-score method, The start and end of each cycle using 'find_peaks" function in Python along with the duration of each cycle and put the results in Hours column.
Requirement 5. Define the regression variables, inputs and training sets.
The dependent variable (y, hours) and independent variable (x, air filter) and training set are defined. The threshold is the maximum air filter among all cycle (*T = MAX (Air Filter Restriction)*).
Requirement 6. Apply the linear regression function.
The linear model is called from scikit-learn library and then fit the model to the training set.
Requirement 7. Obtain the calculated equation.
Put the obtained intercept and co-efficient from the linear regression into the regression equation and derive the estimated remaining hours by subtracting Y and the hours at prediction date in the dataset.

3.3.3. AI Dashboard Requirements

This sub section presents the set of requirements to develop an AI dashboard with Streamlit. The developed AI dashboard is to upload the data and see it visually on the web browser instead of writing code each time in the code platform. This AI dashboard is coded based on the requirements for the software development company to browse the data for underground mines. The details of functional processes and measurement results for the two mentioned

scopes along with data groups are provided in Appendix II. The five main features are developed in this AI dashboard:

Feature 1. Data loader.

The Data loader provides three options (SQL server, local computer, and excel/csv) to upload the data directly from the web browser. The cache function is used to avoid uploading data each time.

Feature 2. Statistics summary.

The statistics summary provides information about the shape of the datasets, data types and a table of summary statistics including min, max, average, standard deviation, and quantile in the order of columns.

Feature 3. Add equations.

This is used to perform the mathematical operation on columns. The equation is given to the software and then the software applies it on the columns.

Feature 4. Filter selection.

This is used to perform the mathematical operation throughout the data frame or grab a piece of dataset. The command is given by the user to the software.

Feature 5. Data visualization.

Data visualization is developed to plot the dataset with variables on x and y axes. Three plots are developed for this AI dashboard:

1. Seaborn heatmap: is a two-dimensional graphical representation in a matrix shape which demonstrates the correlation between each individual value;
2. Scatter chart: is a two-dimensional plot that represents data as a collection of points;
3. Line chart: is a graph that displays the trend of the data along a number line.

Table 3.5 and Table 3.6 describe the software functions from two different viewpoints for this developed AI dashboard.

Pre-requisite: The Streamlit app needs to be installed on the Python environment using the instruction in Streamlit document on the website and by the local URL obtained in the last step of installation, the web app can be opened on browser.

Table 3.5. Software requirements - AI dashboard

Requirement 1. Upload the data.
There are three options to choose how to upload the data: SQL server, local computer, and excel/csv. Depending on the chosen way, the software asks the required input and proceeds to upload the data.
Requirement 2. Create data frame.
The software provides a list of required columns to choose and based on that creates the data frame.
Requirement 3. Exert statistic summary.
The software displays the statistics summary of the created data frame. In order to show the statistics summary, a checkbox is created to allow to check when the user wants to use this feature.
Requirement 4. Apply the equation.
The software receives the equation entered by user and then applies the required operation on the column when receives the checkbox command from user.
Requirement 5. Apply filter selection.
The software receives the filter selection command entered by user and then applies the required operation throughout the data frame when receives the checkbox command from user.
Requirement 6. Plot the seaborn chart.
The seaborn chart is displayed when the software receives the checkbox command from the user.
Requirement 7. Plot the scatter graph.
The software receives the X and Y variables and then plots the scatter graph.
Requirement 8. Plot the line chart.
The software receives the X and Y (can be more than one Y variable) and plots the multi-line chart.

Table 3.6. Data scientist requirements - AI dashboard

Requirement 1. Import Python libraries.
The main required libraries are: Streamlit, numpy, pandas, mysql.connector, seaborn, plotly_express and plotly_subplot.
Requirement 2. Create checkbox for the required steps.
The data scientist needs to define checkbox for the required functionalities. The checkbox acts as trigger and allows the software to apply the functions.
Requirement 3. Create a widget list for uploading data.
The data scientist needs to create a list of options and based on chosen option the next step appears to the user.

Requirement 4. Create a widget list of equipment.
The data scientist creates a list of all available equipment for the user selection.
Requirement 5. Connect Python to the SQL database.
The data scientist needs to link Python to the SQL database to be able to query the data directly from SQL database.
Requirement 6. Create date picker sidebar widgets.
The date picker widget enables to choose a start and end date for uploading data.
Requirement 7. Create a path for uploading data from local computer.
A path (file location) needs to be created for uploading data files from local computer.
Requirement 8. Create Excel/csv file picker widgets to drag the data file.
The file picker allows to drag the file from the computer into a box dragger in web browser.
Requirement 9. Upload data and create data frame.
In this step, the data scientist adds the cache function for the performance. The cache function avoids the software to upload data every time for the operation and browsing. When the data is uploaded and columns are chosen, the data frame is created. A checkbox is created to allow to create the data frame.
Requirement 10. Exert statistics summary.
The “describe” function provides the statistics summary of the created data frame. A checkbox is created to allow to present the summary.
Requirement 11. Create box for equations.
A textbox is created, and the calculation is applied to the data frame using “eval” command.
Requirement 12. Create box for filter selection.
A textbox is created, and the calculation is applied to the data frame using “eval” command. The coding command for the equations and filter selection is different, so the data scientist needs to create both separately.
Requirement 13. Create seaborn heatmap plot.
The seaborn plot can be created using seaborn function in the seaborn library.
Requirement 14. Create scatter plot.
The scatter plot can be created using px.scatter function from plotly_express library. The X and Y variables are needed to define by selectbox function.
Requirement 15. Create multi-line chart.
The multi-line chart can be created using plotly_express library and subplot function. The X and Y variables are needed to be defined by selectbox and multiselect functions.

3.4. COSMIC Measurement Results of AI Software

This section provides the measurement results of the three AI software based on their FUR from two different viewpoints. The functional processes and details of measurement are provided in the Appendix II.

As shown in Figure 3.4, the software to be measured is the AI application while the software interacts with its functional users (data scientist, python libraries and ML algorithms).

Table 3.7 and 3.8 present the total size of the developed ML algorithms and AI dashboard while the COSMIC size of the software in two different viewpoints is relatively close to each other.

The software viewpoint determines the amount of functionality of the software. Simply put, what the software shall do in terms of requirements for the ML algorithms and AI dashboard.

Table 3.7. COSMIC measurement results of AI software – software viewpoint

AI software	**Data movements**				**COSMIC size**
	Entry	**Exit**	**Write**	**Read**	
K-Means clustering	5	2	2	1	10 CFP
Linear regression	7	1	5	1	14 CFP
AI dashboard	12	13	2	0	27 CFP

The data scientist viewpoint measures the amount of work that the data scientist needs to perform to implement an ML algorithm. In other words, it determines size data for data scientists for their own AI development work.

Table 3.8. COSMIC measurement results of AI software – data scientist viewpoint

AI software	Data movements				COSMIC size
	Entry	Exit	Write	Read	
K-Means clustering	12	0	0	0	12 CFP
Linear regression	14	0	0	0	14 CFP
AI dashboard	25	0	0	0	25 CFP

The results obtained from these three cases showed that the function point techniques are applicable for the ML algorithms and AI applications (answer to RQ 2). The FUR and size information of developed ML algorithms can be used as one of the variables for estimating the development effort of an AI application as well as construction of priori estimation models for AI software. Although there are some common functionalities in the developed AI software in this project, the applicability of them is different and making each application distinct from any other (Lesterhuis & Abran, 2019). The documented FUR allow the data scientists to focus on other ML challenges rather than the coding aspects. The measurement in two different viewpoints assists them to recognize the functionalities (allocated to software) they need to develop as well as the amount of work they should do in practice to develop an ML algorithm in similar projects. Developing estimation models based on requirements and using standard units facilitate the planning for future projects as well as monitoring for projects in progress.

CHAPTER 4

EFFORT ESTIMATION MODEL

This chapter presents an exploratory research on the software projects that have been implemented in a software release using three months of data to answer the RQ 3 and RQ 4 which how descriptive statistics can provide relevant posteriori estimation models that could be applicable in priori context. In practice, there has never been in this company a priori estimation model using the historical data based on software size and effort: estimation was mainly done based on expert opinion and the experience from the past releases with related weaknesses. This raises a number of questions: “How good are expert’s estimates based only on opinions? Can the managers plan the projects based on the expert’s judgments?”. Having formal estimation models in the organization can improve estimation quality and can be a success story (Jørgensen, Boehm, & Rifkin, 2009) and more reliable than the judgment-based effort estimation methods.

The objective of this chapter is to measure the size of the projects developed in a software release for three months, followed by the construction of a posteriori estimation models for these projects. Effort is the other variables in our model that the relevant data on effort is collected for each project. The proposed model is an effort estimation model in the specific context of embedded communications software development underground mines. Functional size is measured using COSMIC method.

Section 4.1 presents an introduction on software release in the company and the related terminology. In section 4.2, the statistical analysis is presented and a discussion on outliers’ removal. Section 4.3 reports on first research phase and the applied method. In section 4.4, the second phase of research and the corresponding results are presented. In section 4.5 the estimation for outliers and risk probabilities of the proposed model are discussed and in section 4.6 the proposed model is aggregated for its usage of industry.

4.1. Software Release

Software release refers to the sum of developments and improvements of a piece of software that is going to officially launch for the customers. In each software release, there are a set of changes to the existing software for instance new features are added to the software to improve the functionality of the software solutions and the identified defects are solved to improve the software selling to customers.

For each development/improvement, a ticket is created in an agile project management software which can be tracked in the system.

The COSMIC method – ISO 19761 has been applied to measure the size in CFP of each project. The corresponding effort in hours was recorded in the time sheets stored in the system. The projects can be classified into three groups:

1. Defects: bugs and errors in the software;
2. Features: new functionalities added to the software;
3. Incidents: customer tickets for problems that sometimes lead to a bug and customer requests to add new functionality in the software.

4.2. Statistical Analysis

Table 4.1 presents the descriptive statistics of 100 raw data points in a software release (includes three months of historical data) in this organization and the functional size in CFP and effort in hours.

Table 4.1. Descriptive statistics of functional size and effort – N=100

Development Projects	SUM	Min	Max	Average	Median	Standard Deviation
Functional Size (CFP)	226	1	11	2.26	2	1.63
Effort (hrs)	2194	1	200.5	21.94	11.2	30.27

The sum of size of the development projects for three months of historical data is 226 CFP and sum of effort is 2194 hours. However, there is a large variation in effort (Std = 30.27 hrs) for a number of projects that have either lower or higher productivity in terms of effort for an equivalent size.

From Table 4.1, a two-dimensional scatter graph is presented in Figure 4.1 with the functional size in CFP on the horizontal axis and with their corresponding effort in hours on the vertical axis.

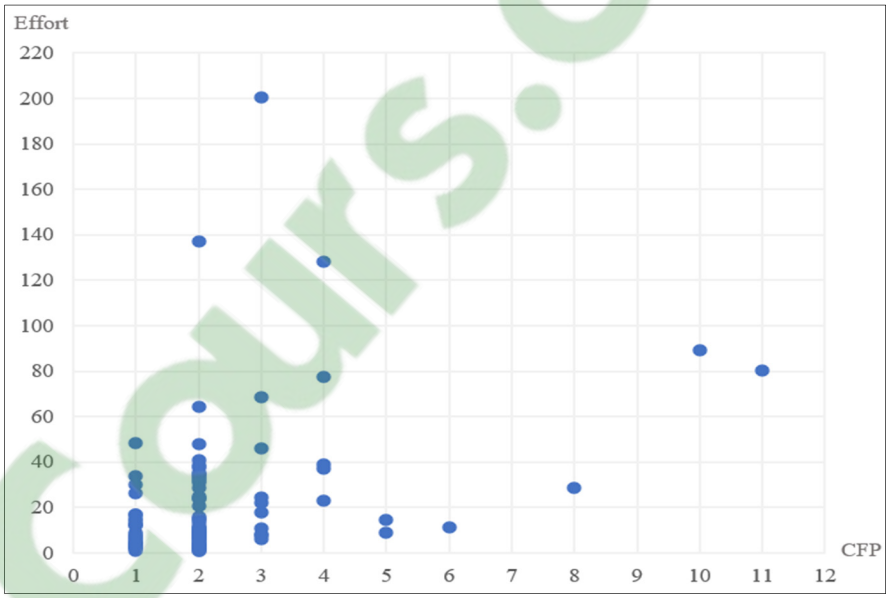

Figure 4.1. Development projects: functional size and effort–N=100

From Figure 4.1, it can be observed that on the x-axis:

1. Most of the data points are within the 1 to 3 CFP size range;
2. There is a sparsely populated interval between 4 and 5 CFP;
3. There is only one data point each with 6, 8, 10 and 11 CFP.

And on the y axis:

1. The effort for most projects varies from 1 to 50 hours;
2. 5% of the projects have an effort between 50 and 100 hours;
3. Only 3 projects have more than 100 hours of effort.

For estimation purposes, these development projects can be classified into two different groups: defects and features.

4.2.1. Defects and Features Groups

This sub section provides descriptive statistics of the raw data classified into two groups: defects in Table 4.2 and features in Table 4.3.

It is to be noted that the Incident group has only 6 data points with a total size of 9 CFP and 98 hours of effort therefore, this Incident group is too small for a meaningful statistical analysis and no statistical results are provided in this research.

Table 4.2. Functional size and effort - defects group– N=49

Defect group	SUM	Min	Max	Average	Median	Standard Deviation
Functional size	81	1	3	1.65	2	0.59
Effort	633.4	1	137	12.93	8	20.29

Table 4.3. Functional size and effort - features group– N=45

Feature group	SUM	Min	Max	Average	Median	Standard Deviation
Functional size	136	1	11	3.02	2	2.12
Effort	1462.4	1	200.5	32.5	23	37.1

From Table 4.2 and 4.3 it can be observed that:

1. The numbers of data points in the defects and features groups are almost equal (49 and 45, respectively);
2. Most of the feature projects have a functional size between 2 CFP to 4 CFP in each development project;
3. Most of the data points in the defects group have a 1 CFP and 2 CFP size;
4. The minimum effort for each group of features and defect is 1 hour;

5. The effort for most of the defect projects varies from 1 to 20 hours;
6. The effort for most of the feature projects varies from 1 to 40 hours;
7. The size and effort of the feature group with 45 data points are larger than the defect group of 49 data points: feature projects take almost 2.3 times more effort than defect projects while the number of the groups are almost equal:
   a. Average time for a feature = 32.5 hours while,
   b. Average time for a defect = 12.93 hours.

4.2.2. Distribution of Effort and Outliers Identification

This section presents the data points of effort variable in a box plot to display the distribution of effort and identify the statistical outliers on effort in both groups. The box plot is an EDA tool for determining outliers and summarizing large quantities of information. The data point that is located outside the whiskers of the box plot is an outlier and needs to be excluded for further analysis. Figures 4.2 and 4.3 depict the box plot of defects and features.

For the outlier identification on the size variable, since there is a very small range of values, from 1 to 11 CFP, no data point has been removed based on the size variable.

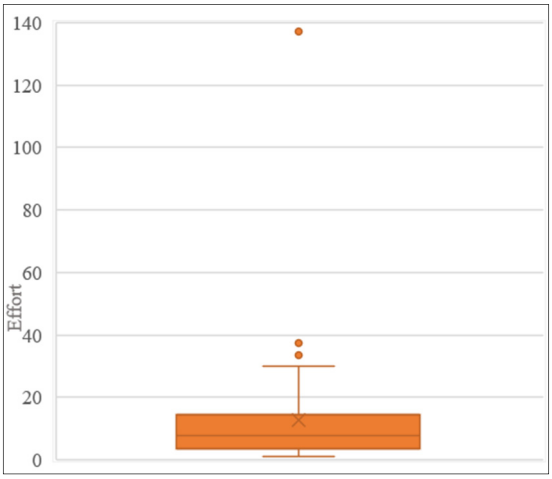

Figure 4.2. Box plot of defects - N=49

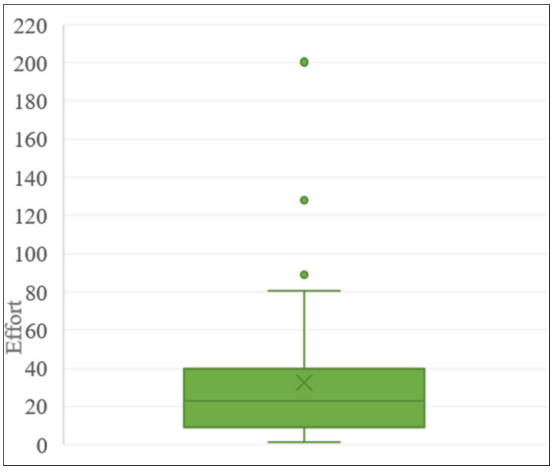

Figure 4.3. Box plot of features - N=45

Seven data points of effort variable in total have been identified as statistical outliers, and they need be removed for the statistical analysis that will follow. The outlier analysis is discussed in sub section 4.2.3. Overall:
In defects: 4 data points with a sum of 8 CFP and 242.75 hours effort are excluded.
In features: 3 data points with a sum of 17 CFP and 417.5 hours effort are excluded.

The two-dimensional scatter graph of development projects in the defects and features groups are displayed in Figures 4.4 and 4.5.

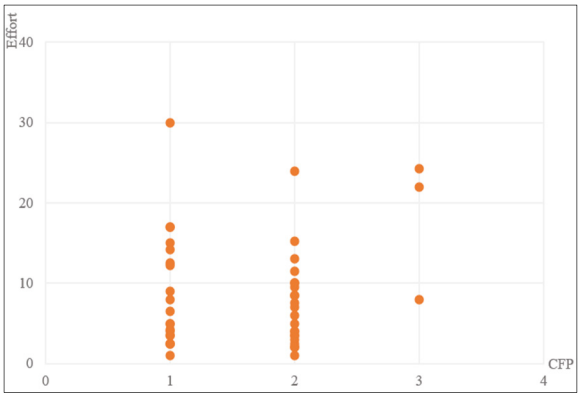

Figure 4.4. Defects group - N=45
(excluding 4 outliers)

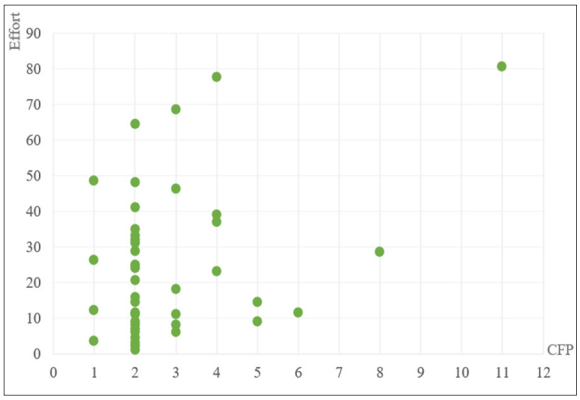

Figure 4.5. Features group - N=42
(excluding 3 outliers)

4.2.3. Statistical Outliers Analysis

As mentioned in the previous section, seven data points of effort variable have been identified as statistical outliers using the box plot method. Those seven data points are significantly larger than all the others in their group. The related Grubbs test for the identification of outliers on effort variables and derived p-values are indicated in Table 4.4 while:

1. The outlier number 1 with 137 hours effort is significantly larger; It is at more than three standard deviations from the average of 12.93 hours effort. It is a significant outlier candidate in the defect group;

2. The outlier number 2, 3, and 4 are far from the average of the dataset. As the p-value and the critical z measures show the amount of standard deviation is far away from the mean;
3. The outlier number 5 with 200.5 hours effort is more than three standard deviations away from the mean. It is a significant outlier candidate in the features group;
4. The outlier number 6 with 128 hours effort is a significant outlier candidate. The p-value and critical z value indicate that;
5. The outlier number 7 is two standard deviations away from the average of the dataset which affects the results of the dataset in the feature group.

Table 4.4. Outlier analysis of effort dataset– significance level = 0.05

NO	Group	Size	Effort	P-value	Critical z	Result
1	Defect	2	137	0.0001	6.1136	Significant outlier
2	Defect	2	37.5	0.0035	2.9239	Furthest from the rest of dataset
3	Defect	2	34.55	0.0034	2.9257	Furthest from the rest of dataset
4	Defect	2	33.7	0.0015	3.1752	Furthest from the rest of dataset
5	Feature	3	200.5	0.0001	4.5281	Significant outlier
6	Feature	4	128	0.0003	3.6579	Significant outlier
7	Feature	10	89	0.0058	2.7612	Furthest from the rest of dataset

Without those seven outliers on effort variables, the distribution of effort in both groups will be much closer to the normal distribution. Figure 4.6 displays the normal distribution curve for both groups after the removal of the outliers.

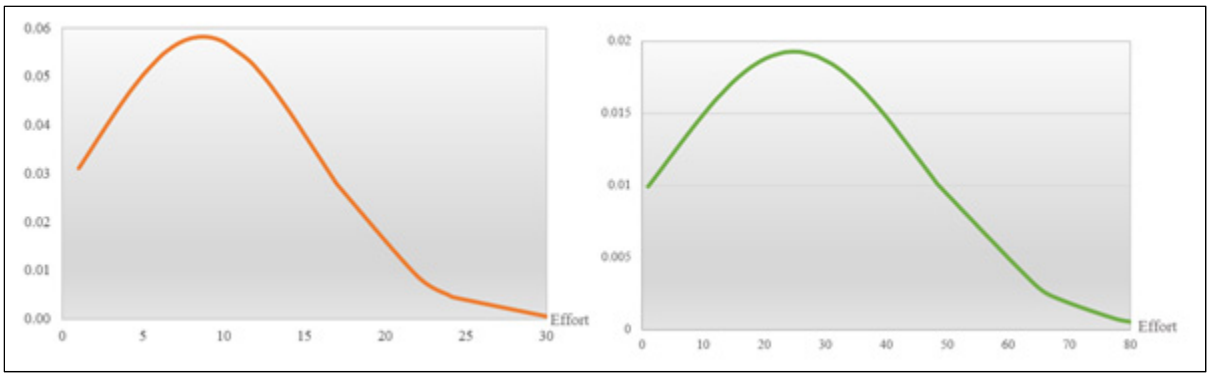

Figure 4.6. Normal distribution curve in defect group (on left) and feature group (on right) excluding seven outliers on effort variables

The average of effort is reduced when those outliers are excluded. The analysis of the impact of outliers on the dataset in the defects group and features group is presented in Table 4.5 while:

In defects: the total effort is decreased by 243 hours (excluding 4 outliers) and,

In features: the total effort is decreased by 417 hours (excluding 3 outliers).

Table 4.5. Analysis of the impact of outliers on effort dataset

Defect	**Effort**	**Size**	**Feature**	**Effort**	**Size**
Total (N=49)	633.4	81	Total (N=45)	1462.4	136
Average (N=49)	12.93	1.65	Average (N=45)	32.5	3.02
Standard Deviation (N=49)	20.29	0.59	Standard Deviation (N=45)	37.1	2.12
Total (N=45) excluding 4 outliers	390.65	73	Total (N=45) excluding 3 outliers	1044.9	119
Average (N=45)	8.68	1.62	Average (N=42)	24.87	2.8
Standard Deviation (N=45)	6.84	0.61	Standard Deviation (N=42)	20.74	1.89

4.3. Research Phase 1: Estimation Model with Linear Regression

This section presents the linear regression models of the relationship between an independent variable - which is size (in CFP), and the dependent variable which is effort (in hours).

The linear regression approach is first applied to build the estimation model and fit a linear equation to the observed data. The estimation model derived from linear equation can be used to predict the amount of effort based on the project COSMIC size. The analysis of linear regression approach is presented in Figures 4.7 and 4.8 that plot the scatter graph with the fitted line on the data.

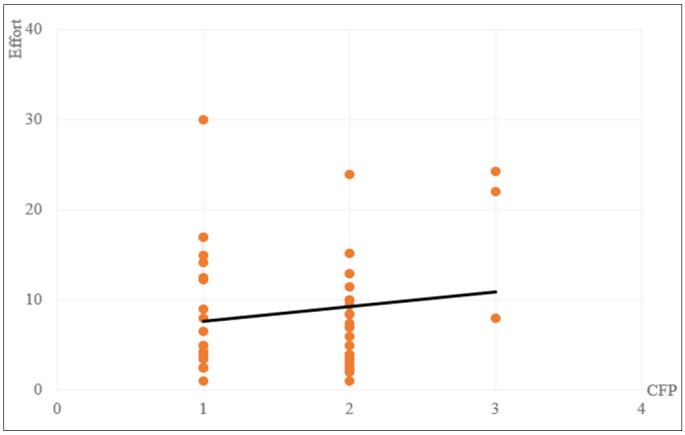

Figure 4.7. Linear regression of defects – N=45

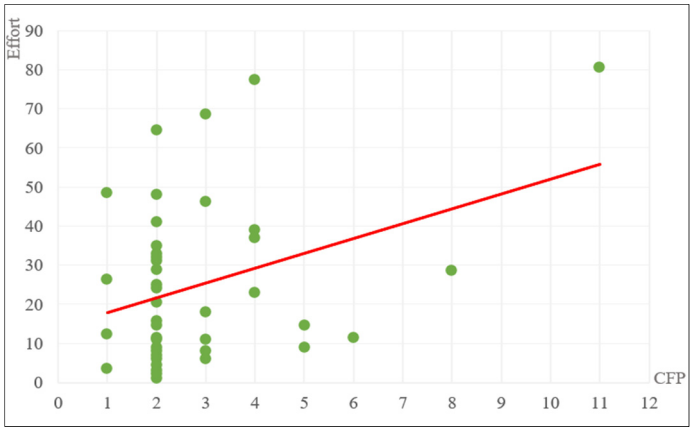

Figure 4.8. Linear regression of features – N=42

Based on the obtained results of linear regressions, the summary of regression analysis is presented in Table 4.6 while:

Despite of the positive regression coefficient, the correlation between independent variable and dependent variable is very weak with a large variation.

Table 4.6. Regression analysis summary for defect and feature group

	Defect	**Feature**
Intercept	6.0704	14.117
Co-efficient	1.6093	3.7982
R^2	0.0208	0.1212
Regression equation	***Effort = 1.6*CFP + 6 hrs***	***Effort = 3.8*CFP + 14 hrs***

The R^2 value in both the defects and features group is less than 0.3 so, the relation between size and effort is very weak.

Table 4.7. The R^2 values

$R^2 > 0.7$	Strong relationship
$0.5 < R^2 < 0.7$	Moderate relationship
$0.3 < R^2 < 0.5$	Weak relationship
$R^2 < 0.3$	Very weak relationship

In other words, either the relationship between variables might not be linear or that another variable might have a significant influence on the relationship between size and effort. Therefore, other significant variables along with size can impact on effort and our estimation model.

We can conclude that, for this small range of dataset, estimation models based on functional size and linear regression analysis are poor estimators and better estimation models must be developed (answer to RQ 3).

4.4. Research Phase 2: Estimation Model with Descriptive Statistics

Descriptive statistics and EDA approach are carried out to analyze the data points and develop a posteriori estimation model that could be used in an a priori context. Software projects have various productivity ratios and some of them may have significantly different productivity behavior and require much more effort compared to the other data points. They need to be identified to take the required precautions in the project life cycle (Abran, 2015). Through descriptive statistic the basic characteristics of data are described. Some extreme projects that may have similar characteristics will be compared to other projects of similar size and effort and those characteristics may lead to very high unit effort and such large productivity variations.

In this section, the dataset is analyzed based on their main characteristics. Some of the data points have significantly different effort compared to the other data points and to obtain improved estimation models, these data points need to be identified. To identify them, the specifications of each project have been investigated and their common specifications were extracted. The next step is to summarize the main characteristics of development projects and be able to gain maximum insights into the data set and its underlying structure.

It was observed in both groups that those significant data points have the same characteristics, and those characteristics lead to much more effort in the projects:

1. 90% of the dataset are within the range close to average and less than two standard deviations away from the average. This dataset is named category 1;
2. The rest of 10% have the characteristics that need effort more than two standard deviation away from the average. This dataset is named category 2.

By classifying the dataset based on their main characteristics, two estimation models can be built using a summary of hypotheses and using a simpler statistical technique, here a simple average of effort. This will help to know how to estimate the average effort of a development project with its initial characteristics that exist in the beginning of the project.

Tables 4.8 and 4.9 present the characteristics that were observed to be similar in each group.

Table 4.8. Similar characteristics of data points - feature group – N=42

Category 1 (90%) – N=38	**Category 2 (10%) – N=4**
Configuration: - in console, software, database, translation	Design software: - redesign software for a full implementation, deign new feature from scratch
Code changes and code mergers	Server update

Table 4.9. Similar characteristics of data points - defect group – N=45

Category 1 (90%) – N=41	Category 2 (10%) – N=4
Connection error: - mainly between console and its components and database. - fixing connection in management panel.	Connection between two individual software or server: - when two individual software do not have connection and one of them might be broken.
Software error: - when the software supposed to display something, and it does not. - when the software needs to update the information and it does not perform it at proper time.	Unexpected software reboot
Properties and limit: - change the frequency, increase/decrease a character limit and fields limit in database	Missed to consider information in the console.

According to the derived effort and the comparison between those classified 90% and 10% of the development projects. It can be noted that those 10% of development projects affect the effort dramatically depending on the type of development projects and increase the average effort of the projects. Table 4.10 presents the summary of results:

Table 4.10. Average effort and standard deviations in each group

	Defect	Feature
Average effort for 90%	7 hrs	20 hrs
Std (90%)	4.6	14
Average effort for 10%	25 hrs	73 hrs
Std (10%)	3.44	7.5

From Table 4.10 and Figure 4.9 it can be observed for the defects group that:

1. The average effort for 90% of the defects with similar characteristics is 7 hours (e.g., almost 1 business day) while;
2. The average effort spending on the rest of 10% is 3.5 times larger (e.g., 25 hours and = 3 business days).

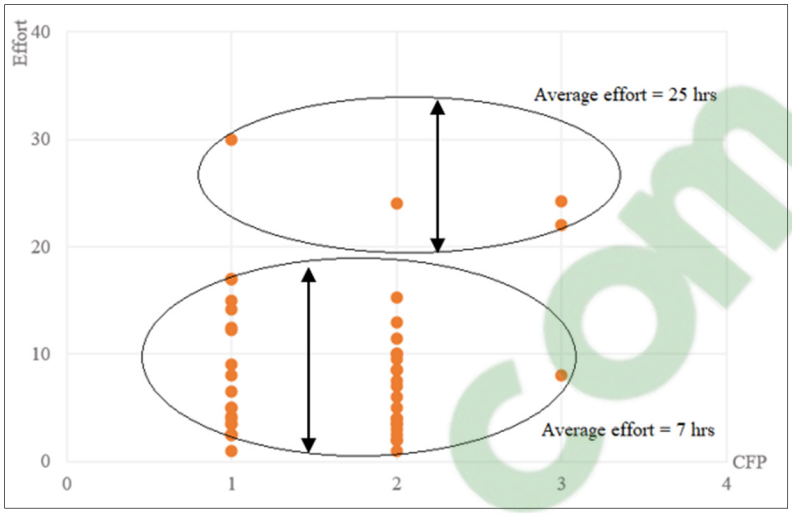

Figure 4.9. Average effort based on characteristics – defects

From Table 4.10 and Figure 4.10 it can be observed for the features group that:

1. The average effort for 90% of the features with similar characteristics is 20 hours (e.g., 2.5 business days) while;
2. The average effort spending on the rest of 10% is 3.5 times larger: it takes 73 hours and 9 business days (almost 2 weeks).

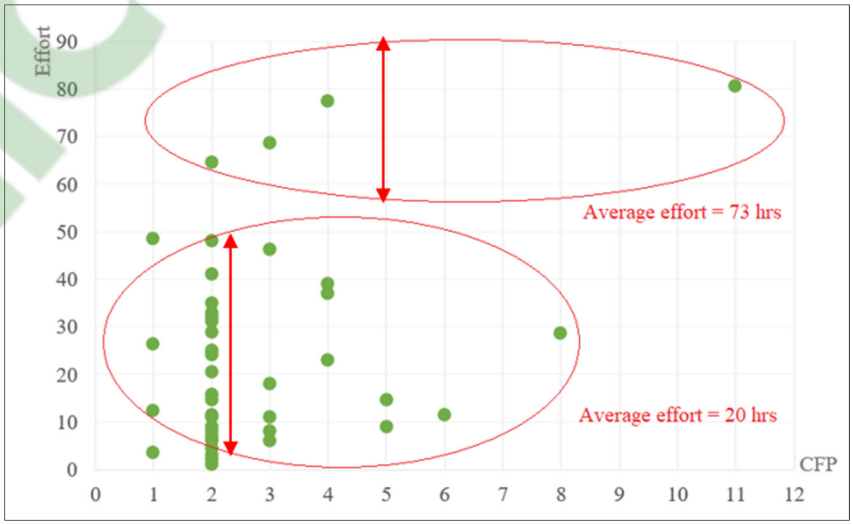

Figure 4.10. Average effort based on characteristics – features

Overall, the effort spent on the features is almost 3 times more than the effort spent on the defects. Tables 4.11 and 4.12 present the descriptive statistics of the data points in functional size and effort according to the classified percentage in each group.

Table 4.11. Functional size and effort according to classified percentage – defects

Classification	SUM	Min	Max	Average	Median	Standard Deviation
Category 1 (N=41)	64 CFP	1 CFP	3 CFP	1.5 CFP	2 CFP	0.54
	290 hrs	1 hrs	17 hrs	7 hrs	6 hrs	4.6
Category 2 (N=4)	9 CFP	1 CFP	3 CFP	2.25 CFP	2.5 CFP	0.95
	100 hrs	22 hrs	30 hrs	25 hrs	24.12	3.44

Table 4.12. Functional size and effort according to classified percentage – features

Classification	SUM	Min	Max	Average	Median	Standard Deviation
Category 1 (N=38)	99 CFP	1 CFP	8 CFP	2.6 CFP	2 CFP	1.44
	754 hrs	1 hrs	48.5 hrs	20 hrs	15.12 hrs	14
Category 2 (N=4)	20 CFP	2 CFP	11 CFP	5 CFP	3.5 CFP	4
	291 hrs	64.5 hrs	80.5 hrs	73 hrs	73 hrs	7.5

From Table 4.11 and through a comparison in Table 4.2, it can be concluded that in defects group:

1. The standard deviation of effort is decreased from 20.29 (in Table 4.2) to 4.6 for the category 1 and to 3.44 for the category 2. There are less values of dataset farther away from the mean;
2. The standard deviation of size is still below 1 which means there was not much more variation in the size from the beginning;
3. The average size in Table 4.2 is still the same as the average size for the category 1 however, the average size has a minor larger size for the category 2.

From Table 4.12 and through a comparison in Table 4.3, it can be concluded that in the features group:

1. The standard deviation of effort is decreased from 37.10 (in Table 4.3) to 14 hours for the category 1 and to 7.5 hours for the category 2. There are less values of dataset farther away from the mean;
2. The standard deviation of size is also declined from 2.12 (in Table 4.3) to 1.44 hours for the category 1 while, the variation of size is increased to 4 hours for the category 2;
3. The average size in Table 4.3 is around 3 CFP while the average size for the category 1 is declined to 2.6 CFP and the average size is larger at 5 CFP for the category 2.

With this EDA approach, a number of distinct estimation models can be proposed based on similarity of characteristics and the approximated effort is derived from the average effort in the corresponding groups, based on historical data. The EDA based on descriptive statistics provides insight to obtain better understanding of the characteristics of the datasets in both groups and discover expected behavior. We can conclude that, in our small range of dataset, the descriptive statistics and using EDA approach provide better posteriori estimation models that the organizations can use in a priori context for a new development projects (answer to RQ 4).

4.5. Estimation Steps for Outliers and Risk Probabilities

This section presents the information of risk probability in the proposed estimation model. As it is shown in the outlier's analysis sections (4.2.2 and 4.2.3) projects with same size have much more effort than another of comparable size and these variations of effort sometimes cause a very high unit of effort in the software projects even their characteristics are in one of the classified categories. It is important to know that these projects are inevitable that we have such variations in real business environment and there is no guarantee the effort for a new project be equal to the estimated effort. By investigating among the datasets of defect and feature groups and using the information of outliers in box plots 4.2 and 4.3, the following risk probability in our proposed estimation model exist.

From 94 data points in total (considering the statistical outliers) there is:

1. A probability of 93 % that a new development project is among one of those categories and;
2. A probability of 7% that a new development project is an outlier and we can not directly estimate the effort and the size at the beginning of the project.

Therefore, the risk factors in each group are as follows:

In the defects group there is a risk probability of:

1. 92% that the new project has one of the characteristics either in category 1 or 2 and the probability of 8% to be an outlier;
2. 6% that the new project which seems to be an outlier has the estimated effort between 25 to 40 hours;
3. 2% that the project benefited from very high unit effort compared to others and the effort goes over 130 hours.

In the features group there is a probability of:

1. 93% that the new project has one of the characteristics either in category 1 or 2 and the probability of 7% to be an outlier;
2. 4.5% that the new project has one of the characteristics but benefited from very low unit effort;
3. 4.5% that the new project which seems to be an outlier has the estimated effort between 73 to 140 hours;
4. 2.5% that the benefited from very high unit effort compared to others and the effort goes over 200 hours.

By looking at estimation results objectively, it may have a probability that the new project takes more effort than the estimated effort. Since the statistical outliers have the same characteristics of the rest of the projects – see Table 4.13-:

Table 4.13. Outlier characteristics in each group, defects and features

NO	Group	Size	Effort	Category 1	Category 2
1	Defect	2	137		✓
2	Defect	2	37.5	✓	
3	Defect	2	34.55	✓	
4	Defect	2	33.7	✓	
5	Feature	3	200.5		✓
6	Feature	4	128	✓	
7	Feature	10	89		✓

The project managers need to calculate an effort contingency in their estimation process meaning that a new project may go off track for any reason and take much more effort compared to past projects. Therefore, to calculate the effort contingency, we use the Expected Value method with respect to the project's category and their probabilities in each group.

The calculation of the effort contingency using the Expected Value method starts with the identification of risks probabilities and the average effort impact in each category per group. Then, the expected effort contingency is calculated by multiplying the probability of risk occurring in each category per group by the average effort if it happens and then adding up the results – see the formula below, how the expected effort contingency is calculated per category-group-:

Expected Effort Contingency = (Estimated effort Probability of risk occurring) + (Average outlier effort* Probability of risk occurring)*

Table 4.14 and 4.15 presents the expected effort contingency of the projects using Expected Value method:

Table 4.14. Expected effort contingency of projects – defect group (N=49)

	Category 1	**Category 2**
Number of data points	41+3 (outlier) = 44	4+1 (outlier) = 5
Estimated effort	7 hrs	25 hrs
Probability of risk occurring	0.93	0.8
Average outlier effort	35 hrs	137 hrs
Probability of risk occurring	0.07	0.2
Expected Effort Contingency	**9 hrs**	**47 hrs**

Table 4.15. Expected effort contingency of projects – feature group (N=45)

	Category 1	**Category 2**
Number of data points	38+1 (outlier) = 39	4+2 (outlier) = 6
Estimated effort	20 hrs	73 hrs
Probability of risk occurring	0.97	0.66
Average outlier effort	128 hrs	145 hrs
Probability of risk occurring	0.03	0.34
Expected Effort Contingency	**23 hrs**	**97 hrs**

In the case above, it may need to add those extra hours to the base average effort as a contingency to cover risks of finding outliers in the projects.

4.6. Steps for Industry to Use the Proposed Estimation Models

This section aggregates the results of the exploratory research and classifies them for the usage within an estimation process tailored to this organization. Figure 4.11 represents a process workflow for estimating in this organization the development projects. Depending on the type of development project, the following step by step estimation approach can be taken:

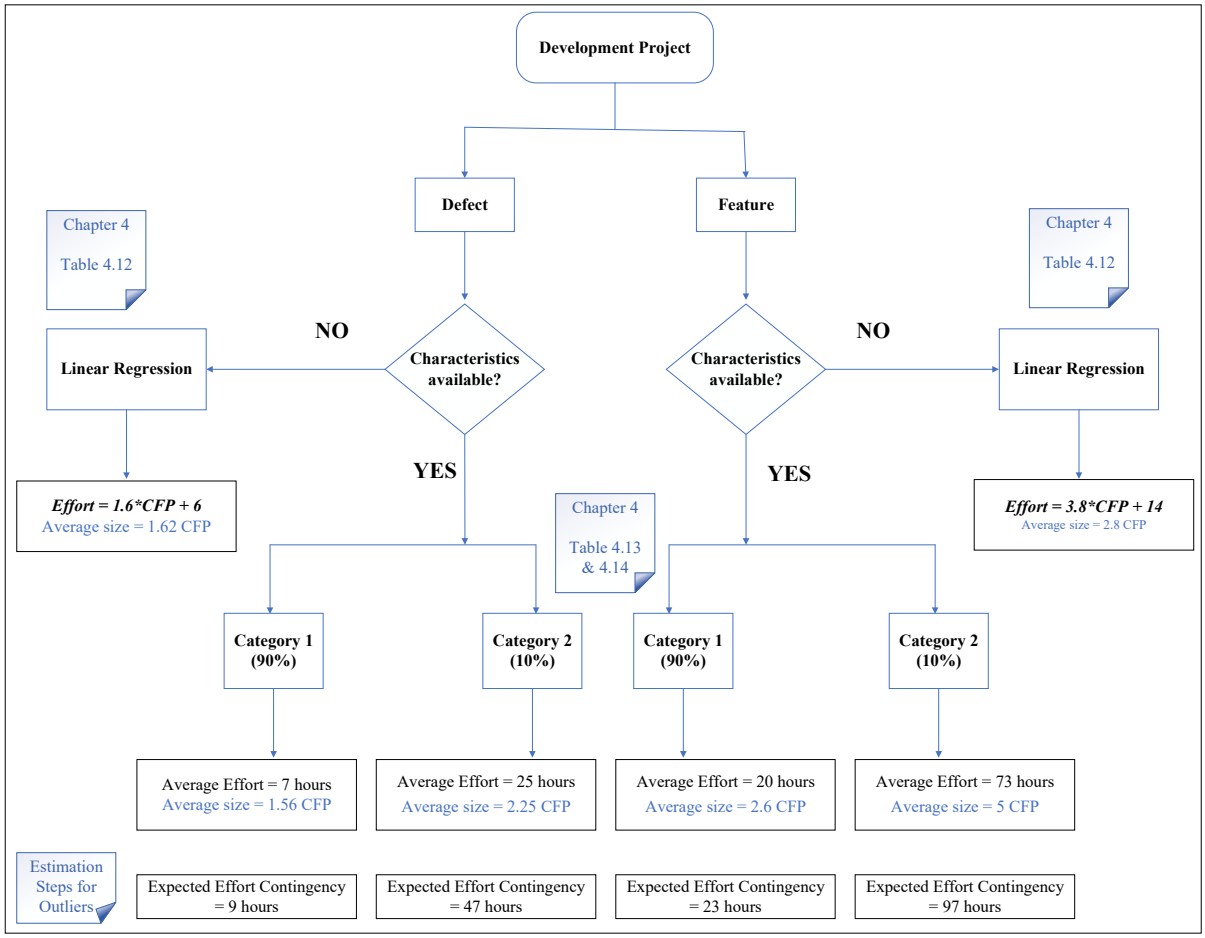

Figure 4.11. Workflow of process for effort estimation for industry

Step 1. Specify the type of the development project: defect or feature.

Step 2. Are the characteristics of the project identifiable at the beginning of the project? (if "NO" follow step 2.1 & 2.1.1, and if "yes" follow step 2.2 and continue).

Step 2.1. If "NO" (the characteristics of the defect/feature are unknown):

Use the linear regression model based on functional size only.

Note: (effort: 'dependent variable' & CFP size: 'independent variable')

Step 2.1.1. Is the size information available?

If "Yes": the CFP size should be put in the regression equation.

IF "NO": the average size is used for effort estimation using the linear regression model.

Table 4.16. The linear regression model for effort estimation

Type of the project	Regression equation	Size information
Defect	*Effort = 1.6*CFP + 6 hrs*	Average size = 1.62 CFP
Feature	*Effort = 3.8*CFP + 14 hrs*	Average size = 2.8 CFP

Step 2.2. If "YES" (the characteristics of the defect/feature are known):

Choose the most relevant characteristics from one the categories below:

Table 4.17. The category of characteristics of the development projects

Group	Category 1 (90%)	Category 2 (10%)
Defect	**Connection error** - mainly between console and its components and database. -fixing connection in management panel. **Software error** - when the software supposed to display something, and it does not. - when the software needs to update the information and it does not perform it at proper time. **Properties and limit** - change the frequency, increase/decrease a character limit and fields limit in database	**Connection between two individual software or server:** - when two individual software do not have connection and one of them might be broken. **Unexpected software reboot** **Missed to consider information in the console.**
Feature	**Configuration:** - in console, software, database, translation **Code changes and code merges**	**Design software:** - redesign software for a full implementation, deign new feature from scratch **Server update**

Step 2.2.1. The project leader needs to decide in which category the project best matches and grab the average estimated effort and the average estimated size for the new project.

Step 3. To calculate the contingency of the projects, use the Expected value method. Identify the risks probabilities and the average effort impact, then calculate by multiplying the risk's probability by the average effort and add up the results.

Expected Effort Contingency = (Estimated effort Probability of risk occurring) + (Average outlier effort* Probability of risk occurring)*

Step 4. Do not assign the contingency to a specific estimation, but accumulate it into the portfolio of projects; this contingency fund can be used in the future when, a posteriori, a project is observed to have happened to be an outlier that needs a posteriori funding.

Table 4.18. The estimated values of effort and size based on project's category

Group	**Category 1 (90%)**	**Category 2 (10%)**
Defect	Estimated effort = 7 hrs Expected effort contingency = 9 hrs Average size = 1.56 CFP	Estimated effort = 25 hrs Expected effort contingency = 47 hrs Average size = 2.25 CFP
Feature	Estimated effort = 20 hrs Expected effort contingency = 23 hrs Average size = 2.6 CFP	Estimated effort = 73 hrs Expected effort contingency = 97 hrs Average size = 5 CFP

CHAPTER 5

SOFTWARE ICEBERG APPROXIMATION

This chapter proposes a new technique for estimating and approximating function points early in the lifecycle of a software project. This research addressed to RQ 5 and provides answers for if the size scaling factors applicable early in a software development life cycle. In this chapter an analytical study using ISO-IEEE 21948 standard on requirements engineering (ISO/IEC/IEEE 29148 Standard, 2011) is presented. It reports on empirical research carried out with two COSMIC case studies. Since there were no documentation of FUR in a priori or a posteriori context available in the company, we had to use the alternative set of data in the COSMIC case studies. Where the requirements are documented at various levels of detail, identified, and classified using a number of concepts from ISO-IEEE 21948 standard and the corresponding COSMIC function points measured on the basis of the most detailed requirements. From these observations, and comparison of the information available at various points in time, again based on ISO-IEEE 21948 standard concepts, size scaling factors specific to these case studies can be developed.

Section 5.1 reports on the requirement engineering lifecycle documented in ISO-IEEE 21948 standard and on the iceberg analogy. Section 5.2 introduces the relevant terminology in requirements engineering and the documentation levels for the proposed iceberg approximation technique. In section 5.3 and 5.4, the COSMIC case studies are analyzed using the proposed size approximation technique and the statistical results and corresponding scaling factors are presented.

5.1. Requirements Engineering and Iceberg Analogy

The ISO-IEEE 29148 standard on requirements engineering presents a number of concepts related to the sources, types and levels of detail of the requirements throughout the system and

software life cycle. The initial set of requirements originates from two sets of sources, the business stakeholders and other stakeholders, which leads to the 'systems' requirements.
From the 'system' functional requirements, some will be allocated to 'software' requirements (as well as to hardware requirements and at times to manual operational procedures). These sources provide the system contextual requirements, including the system purpose, system scope and system overview. From this contextual information, the following are identified next– Figure 5.1 (ISO/IEC/IEEE 29148 Standard, 2011):

1. System functional requirements,
2. System non-functional and quality requirements.

ISO-IEEE 29148 also notes that in addition to software functions explicitly identified, there may be interfaces identified, but not yet specified, as well as quality requirements, still at a high level.

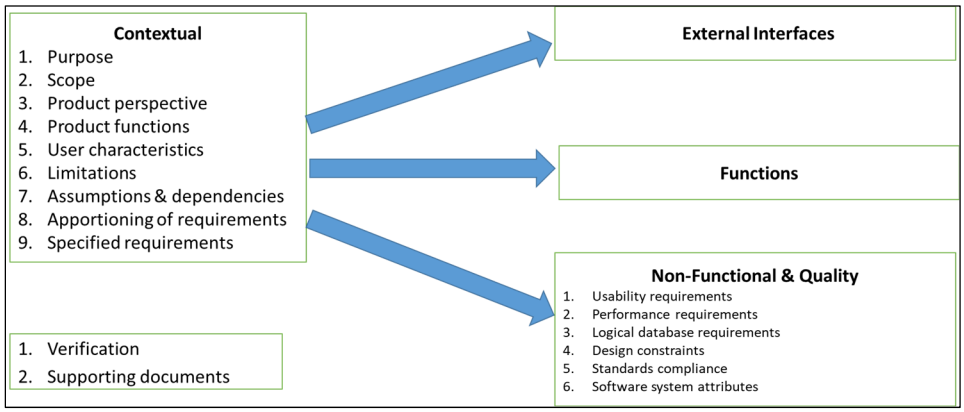

Figure 5.1. Software requirements derived from systems requirements (ISO/IEC/IEEE 29148 Standard, 2011)

It is to be noted that a large number of these system non-functional quality requirements may be derived from stakeholders not identified upfront in the feasibility studies but who must be involved at some point in the operationalization and ongoing operation of the software developed.

At the end of development, all these functional details are known to the developers, even if not formally documented and the software functionality implemented can be measured precisely in COSMIC function points. An outside measurer without access to all the documentation, or the undocumented functionality (such as that derived from system non-functional requirement (NFR) and implemented late in the testing phase), would miss a number of software functions. Similarly, when measurement is done earlier in the lifecycle, a number of software functions which have been neither identified nor specified in detail would be 'invisible' to the outside early measurer.

To illustrate the above, the iceberg analogy is useful, illustrated in Figure 5.2, where on the left, only the easily visible part of the iceberg (e.g., above water line) is evident, to the view on the right where the total iceberg is revealed.

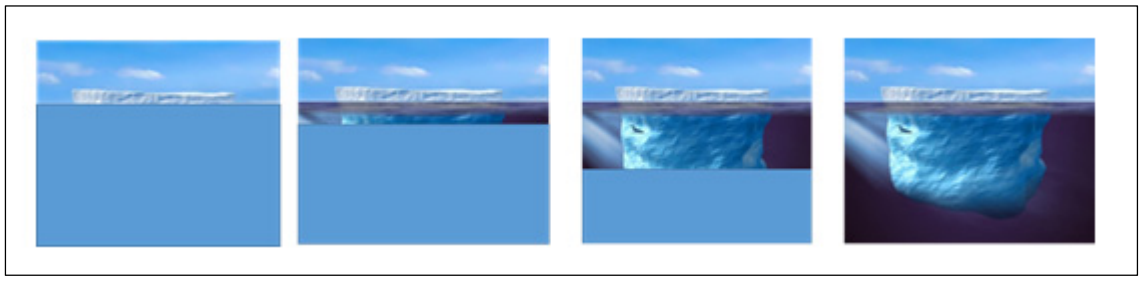

Figure 5.2. The Iceberg analogy: initially visible functions (left) to full view (right)

The concepts from ISO-IEEE 29148 were used, as per the iceberg analogy, to:

1. Identify the types of requirements when they become visible in the life cycle and;
2. Develop (by working backwards) ratios to extrapolate in previous phases.

5.2. Relevant Terminology and Documentation Level

The terminology from ISO-IEEE 29148 to describe the functionality of software is detailed in three levels:

Level 1: Business functions (list of 'system functions'): the information at this level is available at vision time or feasibility study phase;

Level 2: Business functions allocated to software functional processes (list of 'software functions'): often this is available early in the specifications phase;
Level 3: Detailed functionality allocated to each software functional process (functional details allocated to software): this is ideally completed and verified at the end of the specifications phase as well as at the end of a project, provided that the functional documentation has been kept up to date across the lifecycle.

The sources of these functional details were analyzed using the concepts from ISO-IEEE 29148. Each of the functional details within each functional process of the case study was classified into the following five categories with the following color-coding scheme:

1. Functionality from business requirements– allocated to software functions - level 2;
2. Functionality with more details from business requirements - level 3;
3. Operational functionality for implementing in practice the business requirements functionality - level 3;
4. Functionality derived from system requirements & allocated to software - level 3;
5. Functionality related to an interface to other software applications - level 1 or 2.

To be noted, the functions in yellow refer to the 'system requirements' related to data and operational quality requirements allocated as software functions. For ease of readability and traceability, this is re-labelled as 'quality functionality' in the subsequent tables.

This approach was explored with two COSMIC case studies: course registration system (CRS) and restaurant management system (Resto-Sys) and using the statistical information from these measurements and classifications, scaling factors can derived next.

5.3. CRS Case Study and its Scaling Factors in COSMIC Function Points

The classification and color-coding scheme for the functional processes were applied on the COSMIC CRS case study.

The list of business functions and functional processes along with an example of classification for the CRS case study is available in Appendix III.

5.3.1. Results of Functional Size Distribution at Level 3 - CRS Case Study

The results from the classification by origin of the functional details for all the functional processes of the course registration system in five different types of functionality (the types are color-coded) together with their corresponding COSMIC size in CFP units and percentages are presented in Table 5.1, while the two last lines present:
The percentage of the functionality comes from the five different types of functionality.
The average size comes from the five types of functionality.

Table 5.1. CRS – functional classification and size at functional process level 3 (N=21)

ID	Functional process name	Direct business	Business details	Operational business	Quality functions	Interface	Total size in CFP	%	%	%	%
1	Add a professor	1	0	2	2	0	5	20	0	40	40
2	Modify a professor	1	0	1	1	0	3	33.3	0	33.3	33.3
3	Delete a professor	1	0	1	1	0	3	33.3	0	33.3	33.3
4	Enquire on a professor	1	0	2	1	0	4	25	0	50	25
							15				
……							…				
Sub-total in CFP		**21**	**9**	**42**	**30**	**1**	**102**	-	-	-	-
Percentage over TOTAL size		**20%**	**9%**	**41%**	**30%**	**-**	**100%**	-	-	-	-
Average size		**1**	**0.4**	**2**	**1.4**	-	-	-	-	-	-

5.3.2. Functional Size Distribution at Level 2 - CRS Case Study

Using the iceberg analogy -as shown in Figure 5.3- and the above information sized at the functional process level it can be noticed that:

1. 20 % of the functionality comes from the size of the functions listed from the systems software requirements;
2. 9 % comes from the functional details added later to the software requirements;
3. 41% comes from the operational functionality that must be added to implement such functional requirements in an operational context (business, embedded software, etc.);
4. 30% came from the implementation of quality derived functionality allocated to the software – here more specifically 'data integrity'.

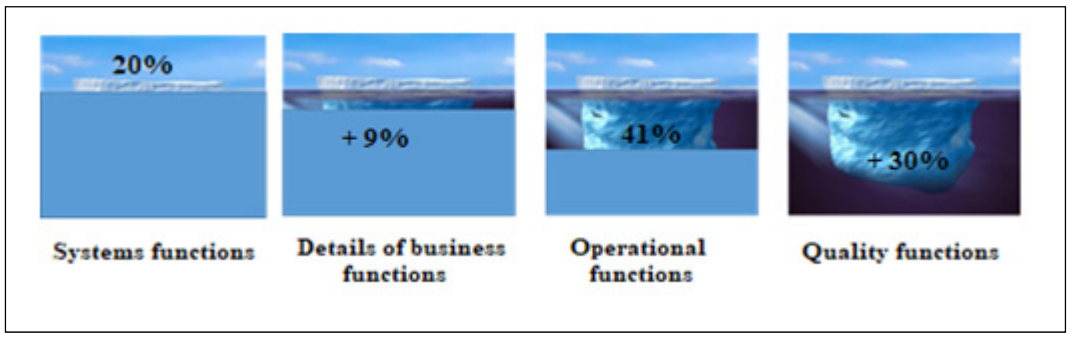

Figure 5.3. CRS case study – functional size distribution

It is to be observed that the above requirements over the lifecycle were progressively identified, from earliest to latest. Such information (i.e. the percentage per classification of requirements) can then be used as scaling factors to estimate the final size of the fully developed software, taking into account the functionality-type to be added across the project life. Of course, the usual caveat applies for similar types of applications, similar organizational contexts, etc.

The statistical information and scaling factors can be used to provide an estimate of the final size of the corresponding software -see Figure 5.4-:

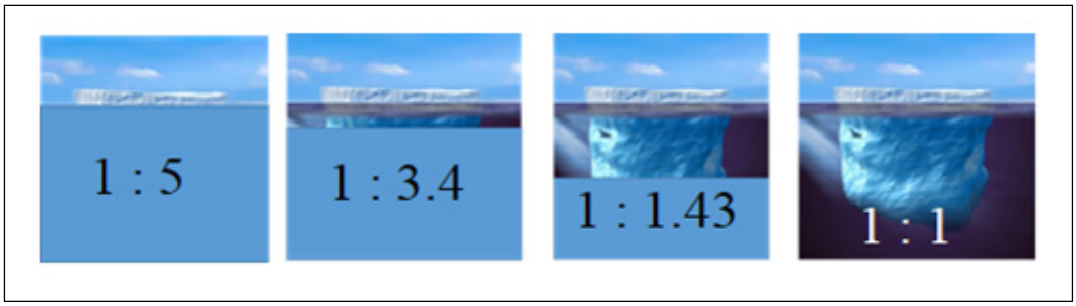

Figure 5.4. CRS case study - transformation into scaling factors of requirements

1. 20% system functions lead to a 1:5 scaling factor, for example: a size of 10 FP would lead to 10*5= 50 CFP when fully specified, including operational functions and data integrity functions;
2. Details of business functions: 20%+9% (= 29%) leads to a 1:3.4 scaling factor, for example: a size of 20 CFP would lead to 20*3.4 = 68 CFP.

The above have all been classified and calculated on the basis of the functional processes allocated to software.

5.3.3. Functional Size Distribution at Level 1 - CRS Case Study

A similar approach can be developed for earlier usage by using the list of 'system' requirements (e.g. level 1) instead of the list of functional processes (e.g. level 2) that become available later.

To develop a scaling factor for level 1, the size information available at levels 2 and 3 was rolled-up at level 1 (e.g., system requirements level in ISO-IEEE 29148) with the following system level functions from business functions – see results in Table 5.2.

As an example, if for a subsequent project, 10 additional system functions are identified, this statistical information and scaling factors can be used to estimate the final size of the corresponding software:

10 business functions with a size of 3 CFP at the feasibility study phase would represent 20% of the total expected total functionality (or 30 CFP a functional process), and would then scale up to an estimated final added functional size of 150 CFP.

Table 5.2. CRS – functional classification and size at system function level 1 (N=7)

ID	Functional process name	Business functions	Business details	Operational functions	Quality functions	Interface	Total size	%	%	%	%
1	Maintain professor	4	0	6	5	0	15	27	0	40	33
2	Maintain student info	4	0	6	5	0	15	27	0	40	33
3	Maintain course	5	4	8	8	0	25	20	16	32	32
4	Maintain student schedule	5	3	14	7	1	29	18	11	49	25
Sub-total: Maintain functions		**18**	**7**	**34**	**25**		**84**	**22**	**9**	**50**	**30**
Average: Per maintain functions		**4.5**					**21**				
5	Close registration	1	1	2	3		7	15	15	29	43
6	Submit grades	1	1	4	1		7	15	15	55	15
7	Enquire report card	1	0	2	1		4	25	0	50	25
Sub-total: Other functions		**3**	**2**	**6**	**5**	**0**	**18**	**17 %**	**12 %**	**36 %**	**28 %**
Average: Per other functions		**1**					**6**				
TOTAL %		21	9	41	30	1	**102 CFP**	**20 %**	**9 %**	**41 %**	**30 %**
Average from TOTAL		**3**	**1.3**	**6**	**4.3**		**14.6**				

5.4. Results of Resto-Sys Case Study

The same approach was applied on the COSMIC Resto-Sys case study which is composed of two parts: a mobile app and a web application. For more details see Appendix III section 2.

5.4.1. Results of Functional Size Distribution at Level 3 – Resto-Sys Case Study

The functional processes of the Resto-Sys case study were classified into the five different types of functionality (including security-NFR allocated to software functions) together with their corresponding COSMIC size in CFP units and percentages - see Table 5.3, while the two last lines present:
The percentage of the functionality comes from the five different types of functionality.
The average size comes from the five types of functionality.

The list of business functions and functional processes for Resto-Sys case study is available in Appendix III.

Table 5.3. Resto-Sys case study - list of functional processes and their sizes (N=33)

ID	Functional process name	Direct business	Business details	Operational functions	Quality functions	Security functions	Total size CFP	%	%	%	%	%
1	Log on to mobile app.	0	0	0	0	5	5	0	0	0	0	100
2	Add an order	4	0	7	5	0	16	25	0	44	31	0
3	Modify an order	3	0	6	3	0	12	25	0	50	25	0
Total functional size of mobile application							**33 CFP**					
4	Create new order	4	1	7	2	0	14	29	7	50	14	0

ID	Functional process name	Direct business	Business details	Operational functions	Quality functions	Security functions	Total size CFP	%	%	%	%	%
5	Modify existing order	3	2	3	1	0	9	33.3	22	33.3	11.1	0
6	Log on as administrator	0	0	0	0	4	4	0	0	0	0	100
7	Add user in web app.	1	0	2	1	0	4	25	0	50	25	0
8	View the user list	1	0	2	1	0	4	25	0	50	25	0
..	…….											
Total functional size of web application							**118 CFP**					
Total in CFP		**40**	**3**	**63**	**36**	**9**	**151 CFP**	-	-	-	-	-
Percentage over TOTAL size		**26 %**	**2 %**	**42 %**	**24 %**	**6 %**	**100 %**	-	-	-	-	-
Average size		**1.21**	**0.09**	**1.9**	**1.09**	**0.27**	**-**	**-**	**-**	**-**	**-**	**-**

5.4.2. Functional Size Distribution at Level 2 & 1 – Resto-Sys Case Study

Using the iceberg analogy, the usage of the information and sizes at level 3 can be rollup-up at level 2 of the software functions where - see Table 5.3:

1. 26 % of the functionality comes from the size of the functions listed from the systems software requirements;
2. Only 2 % of the functionality comes from the functional details added later to the software requirements;
3. 42% comes from the operational functionality that must be added to implement such functional requirements in an operational context (here, a business application);

4. 30% comes from the implementation of quality derived functionality allocated to the software – here more specifically - see Table 5.4:
   a. 24 % as data integrity,
   b. 6% as security through the 2 login simple functions.

Table 5.4. Resto-Sys case study – list of the use cases and their sizes – level 2 (N=10)

ID	Use case name	Direct business	Business details	Operational functions	Quality functions	Security	Total size in CFP	%	%	%	%	%
1	Log on to mobile app.					5	5					100 %
2	Maintain order	7	0	13	8	0	28	25	0	46	29	
Total functional size of mobile application							**33 CFP**					
2	Maintain order	7	3	10	3	0	23					
	Sub-total FUR 2 + NFR	14	3	23	11	5	56	25	5	41	20	9
3	Log on as administrator					4	4	0	0		0	100 %
4	Maintain user	5	0	8	4	0	17	29	0	47	23	
5	Maintain item	5	0	7	5	0	17	29	0	47	29	
6	Add an item family	4	0	8	5	0	17	23	0	47	29	
7	Maintain table	5	0	7	7	0	19	26	0	37	37	
8	Maintain menu	4	0	6	4	0	14	29	0	43	29	
9	View list of orders	1	0	2	1	0	4	25	0	50	25	
10	Delete an order	1	0	1	0	0	2	50	0	50	0	
Total functional size of web application							**118 CFP**					

ID	Use case name	Direct business	Business details	Operational functions	Quality functions	Security	Total size in CFP	%	%	%	%	%
Total in CFP		**39**	**3**	**62**	**37**	**9**	**151 CFP**	-	-	-	-	
Percentage over total size		**26 %**	**2 %**	**42 %**	**24 %**	**6 %**	**100%**	-	-	-	-	
Average size in CFP for 8 FUR		**5**	**0.4**	**8**	**4.6**	**4.5**	**151 CFP (or 4.5 & 19)**	-	-	-	-	

To develop a scaling factor for level 1, the size information available at levels 2 and 3 was rolled-up at level 1 (e.g., system requirements level in ISO-IEEE 29148) with the following system level functions from business functions. The scaling factors are shown on the iceberg Figure 5.5.

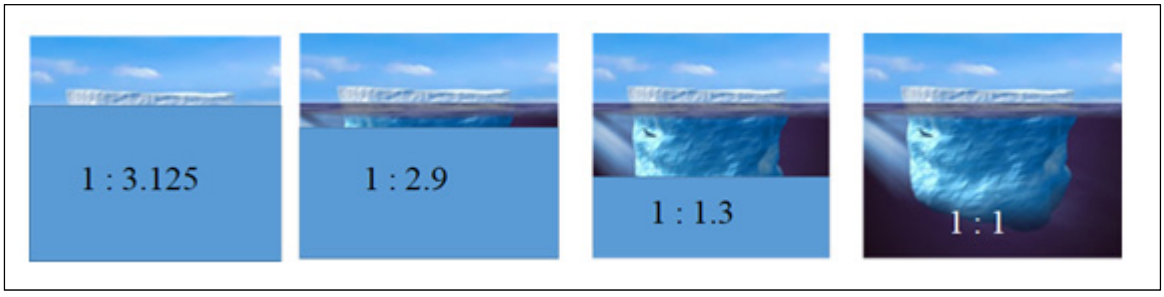

Figure 5.5. Resto-Sys case study - Transformation into scaling factors of requirements

In addition, ISO-IEEE 29148 standard presents a number of concepts related to the sources, types and levels of detail of the requirements throughout the system and software life cycle. ISO-IEEE 29148 standard also notes that in addition to software functions explicitly identified, there may be interfaces identified, but not yet specified, as well as quality requirements, still at a high level.

From these observations and comparison of the information available and described using concepts from ISO-IEEE 29148 standard, at various points in time size scaling factors specific to these case studies and levels of documentation and sizing were developed.

The challenge of designing functionality scaling factors was worked out from known detailed documented requirements measured in a context of full visibility, full documentation, and no uncertainty. More specifically, requirements were positioned at three levels of documentation, from the initial high-level system level down to the most detailed functional levels where all the requirements allocated to the software from the business functions to the operational functions as well as quality functions allocated to software were known.

Such scaling ratios, with successive levels of documentation, can be used in future projects as a project progresses through the lifecycle, and documentation of levels of completeness.

This approximation technique was not operated in this software development company due to some constraints. The functional size measurement using FP technique was used for the first time in this company and before that, no documentation of the FUR were available even specifically at these three levels of documentation. In fact, this functionality-based approximation technique can be used when the provided data on past projects can be collected and relevant classification of functionalities identified. With this in mind, COSMIC measurement of FUR was done for the first time in the company and the majority of concepts need discussion from the initial high-level system level requirements down to the most detailed functional levels. The documentation of FUR specific to each software need to carry out in every software release and then the comparison of the acquired information should be updated at various points in time. Therefore, such scaling ratios, with successive levels of documentation, can be used in the future projects in the company such as project progress through the lifecycle, and documentation of levels of completeness.

5.5. Threats to Validity

The research in this thesis is based on actual measurements and the measurement expert has provided clarifications and assumptions to interpret the functionalities in this specific context. So, there is no significant threat to internal validity.

An external validity threats is associated to the validity of findings. The obtained measurement results on the size of real-time embedded and AI software are specific to this context and the results can not be generalized to other contexts, situations, and settings.

In addition, the scaling ratios derived from the two case studies are specific to these case studies and only the technique can be used in the organizations, that provided data on past projects with relevant classifications.

CONCLUSION

Estimating the development effort of software projects has always been a challenge for software development organizations. The literature on software development effort proves that functional size is one of notable factors that the organizations need to consider when they want to build their own posteriori estimation models.

The objectives of this research were to operationalize and experiment a posteriori measurement of functional size of software using the FSM technique and COSMIC-ISO 19761 method in the specific software development context of AI and real-time embedded software in underground mines. This research is conducted in a software development company and the functional size method is used for the first time in this company where there was no documentation of software functional requirements as well as standard methods to quantify the functional size of the requirements.

In this research project, we answered five research questions. In RQ 1, we performed a posteriori measurement of the software developed in that organization. The measurement results of FUR of three software deployed in underground mine are presented. The results show that the COSMIC method can be used to measure the functional size of the real-time embedded software and these size information can be used as one of the independent variables for the construction of priori estimation models at the beginning of the software projects.

To address the RQ 2, we investigated the size measurement with COSMIC standard in the AI domain and explored whether or not the FP technique could be used to measure the size of AI software and ML algorithms. We reported the size measurement of three AI solutions programmed in Python for two different viewpoints. The obtained measurement results demonstrate that the COSMIC method can be used in AI domain and this size information is one of the variables for estimating the development effort as well as construction of priori estimation models for AI software.

To address to the RQ 3 and RQ 4, estimation models based on size and effort using three months of historical data were built. The linear regression model showed that the relation between size and effort is very weak and might not be linear and other unknown variables have impact on the size-effort relationship. Thereby, a posteriori estimation model based on descriptive statistics and EDA approach was developed. In this model, the dataset is classified into two categories based on their main characteristics and for each category average estimated effort and size are obtained. The model demonstrates that within a small set of data, the estimation with average effort can provide good a posteriori estimation models that could be used in priori context.

To answer to RQ 5 and explore the development of size scaling factors for the usage in a priori context, two detailed COSMIC case studies as an alternative dataset were chosen to explore this issue. An analytical study using ISO-IEEE 21948 standard on requirements engineering and iceberg analogy is developed and the results of scaling factors show that such an approximation technique can be used for estimating function points early in the life cycle of a software project.

Research Contributions and Industry Outcomes

The research paper "Development of COSMIC Scaling Factors Using Classification of Functional Requirements" has been published in "29th International Workshop on Software Measurement (IWSM) & 14th International Conference on Software Process and Product Measurement (MENSURA)" in 2019 in Haarlem, The Netherlands.

The software-iceberg approach has been adopted by the COSMIC Group as an additional sizing approximation technique and included in the 2nd release (February 27, 2020) of "Early Software Sizing with COSMIC" both the Practitioners Guide and Experts Guide in chapter 7.

Part of the measurement of developed ML algorithms has been accepted by the COSMIC Group maintaining the COSMIC Software Sizing Standard.

In addition, I have contributed to the editing process as a co-editor of the case study "Data points clustering with Machine Learning": it is in draft status and is currently being reviewed by the COSMIC 'measurement practices committee – MPC'. It is expected to be approved in the coming weeks for publication on the COSMIC website.

Future Work

The research presented in this thesis can lead to further work in the context of AI size measurement in other industries. It can be pursued based on the methodologies used in this thesis.

Other future work may include considering other possible independent variables in the proposed models that could have had impact on the size and effort variables and improve the accuracy of estimations in the organization.

In this thesis two detailed COSMIC case studies are investigated in order to develop the scaling factors. Additional case studies from other domains may provide additional types and sources of functionality that could be considered for scaling purposes.

APPENDIX I

SOFTWARE ELEMENTS

1. Elements of Recorder Unit

The software comes with the following elements:

1. Configuration tool: to generate the configuration file for the recorder software. All information concerning the external sensors and the data to be recorded and how to process the raw information is included in this file such as: time settings, sensor's channels, engine types, parameters and calculations and all of them are set manually by engineers since it varies for each vehicle, engine type, device and customer. This file is required by the recorder to operate.

2. Backend: is the server-side development and responsible for storing and manipulating data. It includes communication server, generation service, importation service and database.

3. Client application: is responsible to check the recorder in real time and receive alarms and events from recorder.

The recorder unit is connected to the vehicle and server by Bus cables. The cables from the vehicle are joined together by a hardware piece called main harness and the main harness is plugged into one of the recorder's port.

2. Communication Server

The communication server provides an environment by a set of specialized services to handle the data. Once the data which is called raw data is received in the communication server, it is stored in raw folder.
The raw data in the raw folder need to be converted to ISA files using generation service. The generation service converts the raw data to the meaningful data parameters with isa extension and stores them in ISA folder. To do that, the data needs to be filtered. There are equations that is written and configured before by engineers:

1. Filter equation: to filter the data that is not needed to be converted to ISA;
2. Conversion equation: to convert the data from one unit to another (e.g. Fahrenheit to Celsius);
3. Timer equation: to measure time period of a specific parameter;
4. Alarm state equation: to specify the alarm state (e.g. temperature over 50 C is alarm state).

Then, the importation service imports the ISA files into the database. The ISA files that go to the importation service create events. The events are created by a set of conditions that are written and configured before by engineers. If a condition is met, an event is created, and the events are stored in the database. For example, if the temperature parameter is over 50 C for a

period of 10 minutes, create high temperature event. If a condition is not met, the ISA files are transferred to ISA processed folder and archived there.

The order of processing data files is: IBC – Raw – ISA – Database. The ISA files are readable using MET Analyzer application installed on windows.

The client application is an API[2] console and the data is interpreted on the console. There is backend software to create reports and display alarms in the console for user's follow-up. There are standard reports and customized reports that are written engineers using Power BI application. The reports are written based on recognized events stored in database. The notification and alarm warnings are configurable in the console for user's observation, such configurations are:

1. Alarm notification in console: a message is displayed in the alarm list;
2. Email: an email is sent to the configured email address;
3. Fax: a report is sent to the fax machine;
4. Printer: a report is sent to the printer;
5. PDF: a pdf report is created and saved in a folder that configured before.

The communication server parts, and data conversions are not specific to the recorder unit measurement context; therefore, its measurement is not included.

3. Cap Lamp Hardware Elements

Cap lamp safety device has several buttons on the device itself that makes the software inside the cap lamp start the process:

1. There are two buttons on the top of the cap lamp named lamp button and clock/display button which are mostly used for displaying information;
2. The lateral buttons are applicable for the emergencies and safety purposes;
3. There are two LEDs: main LED and auxiliary LED;
4. There are two network LEDs:
   a. Green: blinks once every 5 seconds which means the cap lamp is under wireless network coverage and;
   b. Red: blinks once every 60 seconds which means the cap lamp is not under wireless network coverage.

To be noted that, the number of the nodes under the ground depends on size of the mine and the distance to the server. The nodes should be installed at suitable distances and the miner should work within a specified area to stay connected. The last node is connected to the server by an ethernet cable.

---

[2] Application Program Interface

4. Tag Reader Hardware Elements

In order to be able to use the tag reader, the RFIDs and the device's serial numbers must be configured manually in the server. The scanning is done by a tag reader and then the tag reader must be connected to a tag reader gateway and the server to perform the association in the server.

5. COSMIC Size of the Software

Based on the given FUR in the main text in chapter 2, the measurement of functional size of the software is presented next.

5.1. Recorder Unit Measurement Results

The measurement with COSMIC Function Points of each corresponding functional process is presented in Tables A I-1 to A I-5.

Functional process 1: Receive the configuration file.

Table-A I-1. FP 1 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The Software receives the configuration file on BOT	Configuration file	Entry	1
The software stores the configuration file on hard disk on BOT	Configuration file	Write	1

Functional process 2: Record data on BOT.

Table-A I-2. FP 2 – size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives unformatted streams of data	Unformatted data	Entry	1
The software reads the configuration from configuration file	Configuration file	Read	1
The software records the formatted data as raw data	Data manipulation	-	-
The software sends the raw data from BOT to TOP	Raw data	Exit	1

Functional process 3: Store data.

Table-A I-3. FP 3 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives raw data on TOP	Raw data	Entry	1
The software stores the raw data on hard disk on TOP	Raw data	Write	1
The raw data stores as raw data segments	Data manipulation	-	-

Functional process 4: Transfer data to the communication server.

Table-A I-4. FP 4 – size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives the connection establishment through Bus cable	Connection signal	Entry	1
The software reads the raw data segment from hard disk on TOP	Raw data segments	Read	1
The software sends the raw data segments to the communication server	Raw data segments	Exit	1

Functional process 5: Delete the data.

Table-A I-5. FP 5 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives the reset command	Reset command	Entry	1
The software deletes the raw data on TOP's disk	Reset command	Write	1

5.2. Cap lamp Measurement Results

The measurement with COSMIC Function Points of each corresponding functional process is presented in Tables A I-6 to A I-14.

Functional process 1: Start cap lamp's self-test mechanism.

Table-A I-6. FP 1 – size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives a self-test command after taking out the cap lamp from charger base	Self-test command	Entry	1
The software sends a blinking command to the main LED	Blinking command	Exit	1
The software receives stop command from lateral buttons pressed by miner	Stop command	Entry	1
The software sends stop command to the main LED to turn it off	Stop command	Exit	1

Functional process 2: Connect to the available wireless nodes.

Table-A I-7. FP 2 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives connection signal from available wireless nodes	Connection signal	Entry	1
Confirmation/error message – green/red LED blinks	Successful/unsuccessful message	Exit	1

Functional process 3: Send distress signal.

Table-A I-8. FP 3 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives distress command from lateral buttons pressed by miner	Distress command	Entry	1
The software sends distress signal to the available wireless node	Distress signal	Exit	1
The software turns on the red LED	Blinking command	Exit	1

Functional process 4: Send man down alarm.

Table-A I-9. FP 4 – size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives the 60 seconds immobility message from the accelerometer sensor in the cap lamp	Immobility message	Entry	1
The software turns on the main LED	Blinking command	Exit	1
The software receives the 90 seconds immobility message from the accelerometer sensor in the cap lamp	Immobility message	Entry	1
The software sends man down alarm to the available wireless node	Man down alarm	Exit	1

Functional process 5: Cancel the man down alert.

Table-A I-10. FP 5 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives a stop command from lateral buttons pressed by miner to cancel the man down alarm	Stop command	Entry	1
The software stops the main LED blinking	Stop command	Exit	1
The software sends the cancelation signal to the available wireless node	Cancelation signal	Exit	1

Functional process 6: Respond to evacuation notice.

Table-A I-11. FP 6 – size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives the evacuation notice	Evacuation notice	Entry	1
The software turns on the red LED	Blinking command	Exit	1
The software turns on the main LED	Blinking command	Exit	1

Functional process 7: Alarm acknowledgment by console.

Table-A I-12. FP 7 - size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives a confirmation that the distress signal is received to the server	Confirmation message	Entry	1
The software sends a command to red LED blinking slowly	Blinking command	Exit	1
The software receives the acknowledgment message	Acknowledgment message	Entry	1
The software turns off the red LED	Stop command	Exit	1

Functional process 8: Alarm acknowledgment by cap lamp.

Table-A I-13. FP 8 – size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives acknowledgment from lateral buttons	Acknowledgment command	Entry	1
The software sends the message to the server	Acknowledgment message	Exit	1
The software switches the red LED blinking in slow mode	Blinking command	Exit	1

Functional process 9: Stop the evacuation notice.

Table-A I-14. FP 9 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives stop evacuation notice	Stop command	Entry	1
The software sends stop command to main LED	Stop command	Exit	1
The software sends stop command to red LED	Stop command	Exit	1

5.3. Tag reader Measurement Results

The measurement with COSMIC Function Points of each corresponding functional process is presented in Tables A I-15 to A I-18.

Functional process 1: Connect to the gateway.

Table-A I-15. FP 1 – size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives the successful connection from gateway	Successful connection	Entry	1
The software displays the waiting message on tag reader screen (waiting the gateway connects to the server)	Waiting message: "connecting to server"	Exit	1
The software receives both connection establishment	Successful connection	Entry	1
The software displays the connection message on screen	Connection message: "tag reader is ready"	Exit	1

Functional process 2: Scan the personnel tag (RFID).

Table-A I-16. FP 2 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives tag ID from the miner	Tag ID	Entry	1
The software sends the tag ID to the server through the gateway	Tag ID	Exit	1
Error/Confirmation message may appear	Error/confirmation message	Exit	1

Functional process 3: Scan the serial number of personal safety device.

Table-A I-17. FP 3 – size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives the tag ID of personal safety device	Tag ID	Entry	1
The software sends the tag ID to the server through the gateway	Tag ID	Exit	1
The server receives the serial number and creates an association	Data manipulation	-	-
Error/Confirmation message may appear	Error/confirmation message	Exit	1

Functional process 4: Restart the association process

Table-A I-18. FP 4 - size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives the wrong tag ID	Tag ID	Entry	1
The software receives the 10-second time out signal	Time out signal	Entry	1
The software loses the connection signal with gateway	Connection signal	Entry	1
The software sends restart the association process	Restart association	Exit	1

APPENDIX II

AI SOFTWARE MEASUREMENT RESULTS

The list of functional process and the details of COSMIC measurement for three AI software are presented next.

K-Means Clustering – Software Viewpoint

The list of functional process is presented in Table A II-1 to A II-4.

Functional process 1: Receive the inputs.

Table-A II-1. FP 1 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives the inputs (K =2)	Inputs	Entry	1
The software stores the inputs in cache folder	Inputs	Write	1

Functional process 2: Apply the K-Means function.

Table-A II-2. FP 2 - size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives K-Means function	K-Means function	Entry	1
The software creates the model	Model	Write	1
The software receives the fit command	Fit command	Entry	1
The software retrieves the model from cache folder	Model	Read	1

Functional process 3: Obtain the centroids.

Table-A II-3. FP 3 – size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives the requests to produce the centroids	Request	Entry	1
The software outputs the centroids	Centroids	Exit	1

Functional process 4: Plot the scatter graph.

Table-A II-4. FP 4 – size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives plot function and specified inputs (x, y, colors)	Plot function	Entry	1
The software displays the scatter graph and centroids positions	Scatter plot	Exit	1

K-Means Clustering – Data Scientist Viewpoint

The list of functional process is presented in Table A II-5 to A II-12.

Functional process 1: Import Python libraries.

Table-A II-5. FP 1 – size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist specifies the required libraries	Libraries	Entry	1

Functional process 2: Upload the raw data and create a data frame using the required columns.

Table-A II-6. FP 2 – size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist asks the software to upload the data using "pd.read" function and specifies the file location (path) and column's title to create data frame	"pd.read" function	Entry	1

Functional process 3: Create two empty columns in the existing data frame for further calculation.

Table-A II-7. FP 3 – size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist generates two new empty columns	columns	Entry	1

Functional process 4: Process the raw data.

Table-A II-8. FP 4 – size = 4 CFP

Sub-process	Data group	Data movement	CFP
The data scientist removes the unneeded values of engine speed=0, wheel speed≠ 0	Parameters	Entry	1
The data scientist excludes dump positioning from the data frame	Parameters	Entry	1
The data scientist enters the desired quantile values to label the data points so as to remove the vehicle's actual working data points	Quantile	Entry	1
The data scientist removes the data points with immobility less than constraint	Constraint	Entry	1

Functional process 5: Apply the K-Means function with defined K and fit the dataset to the K-Means model.

Table-A II-9. FP 5 – size = 3 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters the number of clusters K with which the K-Means algorithm determines the K centroids	K cluster	Entry	1
The data scientist enters the fit command to fit Engine speed datasets to the K-Means model	Fit command	Entry	1
The data scientist requests to print the centroids.	Centroids	Entry	1

Functional process 6: Plot the scatter graph.

Table-A II-10. FP 6 – size = 2 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters the plot function with cluster dataset and specifies the corresponding x and y axes	Plot function	Entry	1
The data scientist enters the set of centroids and specifies the colors	Centroids	Entry	1

Linear Regression – Software Viewpoint

The list of functional process is presented in Table A II-11 to A II-15.

Functional process 1: Receive the regression equation.

Table-A II-11. FP 1 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives the equation	Equation	Entry	1
The software stores the equation in cache folder	Equation	Write	1

Functional process 2: Receive the thresholds.

Table-A II-12. FP 2 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives the inputs	Inputs	Entry	1
The software stores the inputs in cache folder	Inputs	Write	1

Functional process 3: Receive the regression variables and training sets.

Table-A II-13. FP 3 - size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives regression variables	Variables	Entry	1
The software stores the variables in cache folder	Variables	Write	1
The software receives training sets	Training sets	Entry	1
The software stores the training sets in cache folder	Training sets	Write	1

Functional process 4: Apply the linear regression to the training set.

Table-A II-14. FP 4 – size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives linear function	Linear function	Entry	1
The software creates the linear model	Model	Write	1
The software receives the fit command	Fit command	Entry	1
The software retrieves the model from cache folder	Model	Read	1

Functional process 5: Obtain the regression equation.

Table-A II-15. FP 5 – size = 2 CFP

Sub-process	Data group	Data movement	CFP
The software receives the requests to produce the results	Request	Entry	1
The software outputs the remaining hours	Predicted remaining hours	Exit	1

Linear Regression – Data Scientist Viewpoint

The list of functional process is presented in Table A II-16 to A II-22.

Functional process 1: Import Python libraries.

Table-A II-16. FP 1 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist specifies the required libraries	Libraries	Entry	1

Functional process 2: Upload the raw data and create a data frame using the required columns.

Table-A II-17. FP 2 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist asks the software to upload the data using "pd.read" function and specifies the file location (path) and column's title to create data frame	"pd.read" function	Entry	1

Functional process 3: Create an empty column in the existing data frame for further calculation.

Table-A II-18. FP 3 – size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist generates a new empty column	column	Entry	1

Functional process 4: Process the raw data.

Table-A II-19. FP 4 - size = 4 CFP

Sub-process	Data group	Data movement	CFP
The data scientist removes the unneeded values of total engine hours=0 and air filter < 0 and engine speed > quantile (x) from the data frame	Parameters	Entry	1
The data scientist removes the noises using z-score function	z-score function	Entry	1
The data scientist uses "find_peak" function to find the start and end of each cycle	"find_peak" function	Entry	1
The data scientist calculates the duration of each cycle in hours	Mathematical formula	Entry	1

Functional process 5: Define the regression variables, inputs, and training sets.

Table-A II-20. FP 5 - size = 4 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters the inputs	Inputs	Entry	1
The data scientist defines the equation	Equation	Entry	1
The data scientist defines the variables	Variables	Entry	1
The data scientist defines the training sets	Training sets	Entry	1

Functional process 6: Apply the linear regression function.

Table-A II-21. FP 6 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters the linear function	Linear function	Entry	1
The data scientist enters the fit command to fit the training set to the linear model	Fit command	Entry	1

Functional process 7: Obtain the calculated equation.

Table-A II-22. FP 7 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist requests the calculated results	Request	Entry	1

AI Dashboard – Software Viewpoint

The list of functional process is presented in Table A II-23 to A II-30.

The “Error message” may appear from the Streamlit app if something wrong is chosen or the app disconnects/stops before displaying data.

Functional process 1: Upload the data.

Table-A II-23. FP 1 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives the inputs depending on the chosen way to upload the data	Inputs	Entry	1
The software uploads the raw data	Data manipulation	-	-
The software stores the data in cache	Raw data	Write	1
Error messages may appear	Error message	Exit	1

Functional process 2: Create data frame.

Table-A II-24. FP 2 – size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives the name of the chosen columns	Columns	Entry	1
The software creates the data frame with the chosen columns	Data manipulation	-	-
The software displays the data frame	Data frame	Exit	1
The software stores the data frame in cache	Data frame	Write	1
Error messages may appear	Error message	Exit	1

Functional process 3: Exert statistic summary.

Table-A II-25. FP 3 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives checkbox command	Checkbox command	Entry	1
The software displays the summary	Statistics summary	Exit	1
Error messages may appear	Error message	Exit	1

Functional process 4: Apply the equation.

Table-A II-26. FP 4 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives the equation	Equation	Entry	1
The software receives checkbox command	Checkbox command	Entry	1
Error messages may appear	Error message	Exit	1

Functional process 5: Apply filter selection.

Table-A II-27. FP 5 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives the filter selection	Filter selection	Entry	1
The software receives checkbox command	Checkbox command	Entry	1
Error messages may appear	Error message	Exit	1

Functional process 6: Plot the seaborn chart.

Table-A II-28. FP 6 - size = 3 CFP

Sub-process	Data group	Data movement	CFP
The software receives checkbox command	Checkbox command	Entry	1
The software displays the chart	Seaborn chart	Exit	1
Error messages may appear	Error message	Exit	1

Functional process 7: Plot the scatter graph.

Table-A II-29. FP 7 – size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives the X and Y variables input	Input variables	Entry	1
The software receives checkbox command	Checkbox command	Entry	1
The software displays the scatter chart	Scatter chart	Exit	1
Error messages may appear	Error message	Exit	1

Functional process 8: Plot the line chart.

Table-A II-30. FP 8 - size = 4 CFP

Sub-process	Data group	Data movement	CFP
The software receives the X and Y variables input	Input variables	Entry	1
The software receives checkbox command	Checkbox command	Entry	1
The software displays the line chart	Line chart	Exit	1
Error messages may appear	Error message	Exit	1

AI Dashboard – Data Scientist Viewpoint

The list of functional process is presented in Table A II-31 to A II-45.

Functional process 1: Import Python libraries.

Table-A II-31. FP 1 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist specifies the required libraries	Libraries	Entry	1

Functional process 2: Create checkbox for the required steps.

Table-A II-32. FP 2 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters checkbox command	Checkbox command	Entry	1

Functional process 3: Create a widget list of options for uploading data.

Table-A II-33. FP 3 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist defines a list of options	Options	Entry	1

Functional process 4: Create a widget list of equipment.

Table-A II-34. FP 4 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist defines a list of input equipment	equipment	Entry	1

Functional process 5: Connect Python to the SQL database.

Table-A II-35. FP 5 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The data scientist defines the connection function	SQL connection	Entry	1
The data scientist enters the server credentials	Server credentials	Entry	1

Functional process 6: Create a date picker sidebar widget.

Table-A II-36. FP 6 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist defines the sidebar function	Sidebar function	Entry	1

Functional process 7: Create a path for uploading data from local computer.

Table-A II-37. FP 7 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist defines a path	Path	Entry	1

Functional process 8: Create Excel/csv file picker widgets to drag the data file.

Table-A II-38. FP 8 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist defines the file picker widget	File picker widget	Entry	1

Functional process 9: Upload the data and create data frame.

Table-A II-39. FP 9 - size = 4 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters the cache function	"@st.cache" function	Entry	1
The data scientist asks the software to upload the data using "pd.read" function	"pd.read" function	Entry	1
The data scientist defines the "multiselect" function for choosing the columns	"multiselect" function	Entry	1
The data scientist enters the pd.dataframe to create data frame	"pd.dataframe" function	Entry	1

Functional process 10: Exert statistics summary.

Table-A II-40. FP 10 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters "describe" function	"describe" function	Entry	1

Functional process 11: Create equations.

Table-A II-41. FP 11 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The data scientist defines textbox function	Textbox function	Entry	1
The data scientist enters "eval" command	"eval" command	Entry	1

Functional process 12: Create filter selection.

Table-A II-42. FP 12 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The data scientist defines textbox function	Textbox function	Entry	1
The data scientist enters "eval" command	"eval" command	Entry	1

Functional process 13: Create seaborn heatmap plot.

Table-A II-43. FP 13 - size = 1 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters "seaborn.heatmap" function	"seaborn.heatmap" function	Entry	1

Functional process 14: Create Scatter plot.

Table-A II-44. FP 14 - size = 2 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters "px.scatter" function	"px.scatter" function	Entry	1
The data scientist defines corresponding inputs (x and y) using selectbox function	selectbox function	Entry	1

Functional process 15: Create multi-line chart.

Table-A II-45. FP 15 – size = 4 CFP

Sub-process	Data group	Data movement	CFP
The data scientist enters "make_subplots" function	"make_subplots" function	Entry	1
The data scientist defines corresponding inputs X using selectbox function	selectbox function	Entry	1
The data scientist defines corresponding inputs Y using multiselect function	multiselect function	Entry	1
The software receives "go.line" function which allows to plot the y variables in the order of choosing	"go.line" function	Entry	1

APPENDIX III

SIZE APPROXIMATION OF COSMIC CASE STUDIES

1. CRS Case Study

The list of business functions and functional process derived from the COSMIC case study is presented in Table A III-1 and A III-2, respectively.

Table-A III-1. CRS case study: list of business functions (N=7)

NO	Business function
1	Maintain professor information (by the registrar)
2	Maintain student information (by the registrar)
3	Maintain courses to teach (by professor)
4	Maintain student schedule (by students)
5	Close registration (by the registrar)
6	Submit grades (by professor)
7	Enquire report card (by students)

Table-A III-2. CRS case study: list of functional processes (N=21)

NO	Functional process
1	Add a professor
2	Modify a professor
3	Delete professor
4	Enquire on a professor
5	Add a student
6	Modify a student
7	Delete a student
8	Enquire on student
9	Add courses to teach
10	Modify a course to teach
11	Delete courses to teach
12	Enquire courses to teach
13	Enquire on course to teach details
14	Add courses
15	Modify a course
16	Delete a course
17	Enquire on courses
18	Enquire on course details
19	Close registration
20	Submit grades
21	Enquire on a report card

An example of color-coding classification for the first business function is presented below:

Table-A III-3. CRS case study: example of the classification of types of functionality

FP No/ Req.	Process descriptions	Functional user/ object of interest	Sub-process description	Data group	DM type	CFP	∑
1/ 1.2.1	Add a professor	Registrar/ professor	Registrar enters information for the professor	Professor data	E	1	
			The system validates the entered data and checks if a professor of the same name exists already	Professor data	R	1	
			The system creates a new professor	Professor data	W	1	
		Registrar/ professor	Display the system generated professor ID number	Professor ID	X	1	
		Registrar/ messages	Display error message	Messages	X	1	
							5 CFP

2. Resto-Sys Case Study

Resto-Sys requirements are as follows:

1. Resto-Sys ensures communication between the smartphone client (the waiter) and the web client (the administrator);
2. Web client maintains the database (which is included in the DB server).

The Resto-Sys application includes the following functionality:

1. Smartphone client receives the waiter's username and password;
2. Web server retrieves data from the DB and provides the required data to the smartphone client;
3. Smartphone client maintains the customer order (by adding or modifying an order);
4. Web client receives the administrator's username and password;
5. Web server retrieves data from the DB and provides the required data to the web client;
6. Web client maintains the required data, the FUR in section 1.4 A in the case study document.

The Resto-Sys includes the following tasks:

1. Order management allows the waiter to add, and/or modify an order via his smartphone. It also allows the administrator to delete an order. During working hours, the waiters (smartphone) and the administrator (web client) are continuously connected;
2. Account management, involves user management, and enables access to the application with a username and password;
3. Restaurant menu management allows the management of item[3] families and the classification of items into item families.

Note that the users of Resto-Sys (waiter and administrator) must be logged on before executing one of the previous tasks (order management, account management, and restaurant menu management).

The list of system functions and functional process derived from the COSMIC case study is presented in Table A III-4 and A III-5, respectively.

Table-A III-4. Resto-Sys case study - use case identification (N=10)

Actor	Global use cases	Detailed use cases
Waiter	Logon	FUR1: Logon
	Maintain order	FUR2: Maintain order
Administrator	Logon	FUR3: Logon
	Maintain data	FUR4: Maintain user FUR5: Maintain item FUR6: Maintain item family FUR7: Maintain table FUR8: Maintain restaurant menu
	Maintain order	FUR9: View the list of orders FUR10: Delete customer order

Table-A III-5 Functional processes – Resto-Sys case study (N=33)

FUR	Functional Processes
FUR 1: Logon	FP 1: Logon
FUR 2: Maintain Order	FP 2: Add an Order
	FP 3: Modify an Order
	FP 4: Create a new order
	FP 5: Modify an existing order
FUR 3: Logon	FP 6: Logon
FUR 4: Maintain User	FP 7: Add a User

[3] Item is used to describe a dish and beverage

FUR	Functional Processes
	FP 8: View Users List
	FP 9: View a User data
	FP 10: Modify User Data
	FP 11: Delete a User
FUR 5: Maintain Item	FP 12: Add an Item
	FP 13: View Items List
	FP 14: View an Item Data
	FP 15: Modify an Item
	FP 16: Delete an Item
FUR 6: Maintain Item Family	FP 17: Add an Item Family
	FP 18: View Item Families List
	FP 19: View Item Family Data
	FP 20: Modify an Item Family
	FP 21: Delete an Item Family
FUR 7: Maintain Table	FP 22: Add a Table
	FP 23: View Tables List
	FP 24: View a Table Data
	FP 25: Modify Table Data
	FP 26: Delete a Table
FUR 8: Maintain Restaurant Menu	FP 27: Add a Restaurant Menu
	FP 28: View Restaurant Menu List
	FP 29: View a Restaurant Menu Data
	FP 30: Modify a Restaurant Menu
	FP 31: Delete a Restaurant Menu
FUR 9: View the List of Orders	FP 32: View the List of Orders
FUR 10: Delete a Customer's Order	FP 33: Delete an Order

APPENDIX IV

PAPER PUBLISHED IN IWSM-MENSURA 2019

Development of COSMIC Scaling Factors Using Classification of Functional Requirements

Alain Abran[1], Shaghayegh Vedadi[1]

[1] École de Technologie Supérieure – ETS, University of Quebec (Montréal, Canada)
alain.abran@etsmtl.ca
shaghayegh.vedadi-moghaddam.1@ens.etsmtl.ca

Abstract. The focus of this paper is estimating COSMIC function points early in the software development lifecycle. The main input to function point sizing is the set of functional requirements for a piece of software. However, very early in the lifecycle it is unrealistic to expect this set of requirements to describe the full scope of functionality, including all the necessary functional details. The application of the size scaling factors developed within this context is illustrated with two COSMIC case studies. While the scaling factors are specific to the case studies used, the approximation technique presented can be used in most organizations provided that data on past projects can be collected and relevant classifications of functionalities identified. For the purpose of this paper, the ISO 19761 COSMIC function points standard is taken as reference for discussion, while the majority of concepts presented are generic to other similar ISO standards.

Keywords: Functional size measurement, COSMIC, size estimation, functional requirements quality, software estimation, ISO 19761

1 Introduction

Function point sizing (FPS) quantifies the functional size of software and is used for various purposes in software project management, including effort estimation, project planning, project monitoring, productivity studies and benchmarking [1, 2]. Compared to lines-of-code based measures, FPS stands out among software size metrics as it is based on the requirements themselves, which as soon as they become available, prior to any coding, can provide size information in the earlier stages of the software development life cycle (SDLC). This makes FPS a tool of choice for planning techniques that require an early view of the software to be developed. Despite being available earlier than other sizing methods, a precise application of FPS requires that the functional requirements of the software be detailed, and the architecture defined [3]. More often than not, this point in the lifecycle comes relatively late for project estimation needs. Therefore, several size estimation techniques have been proposed that can be used for these early management activities.

The main input of FPS is the set of functional requirements for a piece of software. However, very early in the SDLC it is unrealistic to expect this set of requirements to describe the full scope of functionality of the software as a whole, including all the

LIST OF BIBLIOGRAPHICAL REFERENCES

Abran, A. (2010). *Software Metrics and Software Metrology.* Hoboken, N.J: John Wiley & Sons. doi:https://doi.org/10.1002/9780470606834

Abran, A. (2015). *Software Project Estimation: The Fundamentals for Providing High Quality Information to Decision Makers.* Hoboken, N.J: Wiley-IEEE Computer Society Press. doi:https://doi.org/10.1002/9781118959312

Abran, A., & Robillard, P. N. (1996). Function Points Analysis: An Empirical Study of its Measurement Processes. *IEEE Transactions on Software Engineering, 22*(12), 895-910. doi:https://doi.org/10.1109/32.553638

Abran, A., & Vedadi, S. (2019). Development of COSMIC Scaling Factors Using Classification of Functional Requirements. *29th International Workshop on Software Measurement (IWSM) & 14th International Conference on Software Process and Product Measurement (Mensura)*, *2476*, pp. (31-46). Haarlem, The Netherlands. Retrieved from https://espace2.etsmtl.ca/id/eprint/19820

Abran, A., Al-Sarayreh, K., & Cuadrado-Gallego, J. J. (2013). A Standard based Reference Framework for System Portability Requirements. *Computer Standards & Interfaces Journal, 35*(4), 380 - 395. doi:https://doi.org/10.1016/j.csi.2012.11.003

Abran, A., Silva, I., & Primera, L. (2002). Field Studies Using Functional Size Measurement in Building Estimation Models for Software Maintenance. *Journal of Software Maintenance and Evolution: Research and Practice, 14*(1), 31–64. doi:https://doi.org/10.1002/smr.245

Almakadmeh, K. (2013). *Development of a Scaling Factors Framework to Improve the Approximation of Software Functional Size with COSMIC – ISO 19761.* Doctoral Thesis, École de technologie superieure, University of Québec, Montreal, QC. Retrieved from https://espace.etsmtl.ca/id/eprint/1190

Almakadmeh, K., & Abran, A. (2013). Experimental Evaluation of an Industrial Technique for the Approximation of Software Functional Size. *International Journal of Computers and Technology, 10*(3), 1459-1474. doi:https://doi.org/10.24297/ijct.v10i3.3276

Al-Sarayreh, K. T. (2011). *Identification, Specification and Measurement using International Standards of the System Non-functional Requirements Allocated to Real-time Embedded Software.* Doctoral Thesis, Ecole de technologie supérieure, University of Québec, Montreal, QC. Retrieved from https://espace.etsmtl.ca/id/eprint/923

Al-Sarayreh, K. T., & Abran, A. (2010). Software Specifications Framework for System Operations Requirements. *International Journal of Computer and Information Sciences, 10*(3). Retrieved from https://espace2.etsmtl.ca/id/eprint/10278

Al-Sarayreh, K. T., Abran, A., & Cuadrado Gallego, J. (2012). A Standard based Model of System Maintainability Requirements. *Journal of Software Evolution and Process, 25*(5), 459-505. doi:https://doi.org/10.1002/smr.1553

Berg, K., Dekkers, T., & Oudshoorn, R. (2005). Functional Size Measurement Applied to UML-based User Requirements. *Software Measurement European Forum (SMEF)*, (pp. 70-81). Rome, Italy. Retrieved from https://www.researchgate.net/publication/254068874_Functional_Size_Measurement_applied_to_UML-based_user_requirements

Boehm, B., & Abst, C. (2000). *Software Cost Estimation with COCOMO II.* Upper Saddle River, N.J: Prentice Hall Press.

Cheikhi, C., & Abran, A. (2014). An Analysis of the PROMISE and ISBSG Software Engineering Data Repositories. *International Journal of Computers and Technology, 13*(5), 4456-4474. doi:https://doi.org/10.24297/ijct.v13i5.2535

COSMIC. (2015). *Course Registration System Case Study.* COSMIC Organization. Retrieved from https://cosmic-sizing.org

COSMIC. (2015). *Guideline for Early or Rapid COSMIC Functional Size Measurement by Using Approximation Approaches.* COSMIC Organization. Retrieved from https://cosmic-sizing.org

COSMIC. (2017). *The COSMIC Functional Size Measurement Method Version 4.0.2.* COSMIC Organization. Retrieved from https://cosmic-sizing.org

COSMIC. (2019). *Case Study Sizing Natural Language/UML Use Cases for Web and Mobile Applications using COSMIC FSM.* COSMIC Organization. Retrieved from https://cosmic-sizing.org

COSMIC. (2020). *Data Points Clustering with Machine Learning: A case study from the Data Scientist as both the developer and user of that software.* COSMIC Organization. Retrieved from https://cosmic-sizing.org

COSMIC. (2020). *Early Software Sizing with COSMIC: Experts Guide.* COSMIC Organization. Retrieved from https://cosmic-sizing.org

COSMIC. (2020). *Early Software Sizing with COSMIC: Practitioners Guide.* COSMIC Organization. Retrieved from https://cosmic-sizing.org

Desharnais , J. M., & Abran, A. (2003). Approximation Techniques for Measuring Function Points. *13th International Workshop on Software Measurement (IWSM)* (pp. 272-286). Montreal, QC: Springer Verlag.
Retrieved from
https://www.researchgate.net/publication/228738008_Approximation_techniques_for_measuring_function_points

Dumke, R., & Abran, A. (2011). *COSMIC Function Points: Theory and Advanced Practices.* Boca Raton, Fla.: CRC Press-Taylor & Francis Group.

Fetcke, T. (2001). A Generalized Representation for Selected Functional Size Measurement Method. *11st International Workshop on Software Measurement (IWSM)*, (p. 12). Montreal, QC. Retrieved from
https://pdfs.semanticscholar.org/bb3e/a6e8e51758a0e241ed34a2b87b2399650465.pdf

Garbade, M. J. (2018, 09 12). *Machine Learning*. Retrieved from Towards Data Science: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

García-Floriano, A., López-Martín, C., Yáñez-Márqu, C., & Abran, A. (2018). Support Vector Regression for Predicting Software Enhancement Effort. *Information and Software Technology, 97*, 99-109. doi:https://doi.org/10.1016/j.infsof.2018.01.003

Hosni, M., Idri, A., & Abran, A. (2018). Improved Effort Estimation of Heterogeneous Ensembles using Filter Feature Selection. *13th International Conference on Software Technologies (ICSOFT)*, (pp. 405-412). Porto, Portugal.
doi:http://dx.doi.org/10.5220/0006929104050412

Huijgens, H., Bruntink, M., Deursen, A., Storm, T., & Vogelezang, F. (2016). An Exploratory Study on Functional Size Measurement Based on Code. *IEEE/ACM International Conference on Software and System Processes (ICSSP)* (pp. 56-65). Austin, TX: IEEE. doi:https://doi.org/10.1109/ICSSP.2016.016

Hussain, I., Kosseim, L., & Ormandjieva, O. (2013). Approximation of COSMIC Functional Size to Support Early Effort Estimation in Agile. *Data & Knowledge Engineering, 85*, 2-14. doi:https://doi.org/10.1016/j.datak.2012.06.005

Idri, A., & Abran, A. (2018). Software Development Effort Estimation with Feature Selection: A Systematic Mapping Study. In Fujita, & Herrera-Viedma (Eds.), *New Trends in Intelligent Software Methodologies, Tools and Techniques* (Vol. 303, pp. 439-452). Amsterdam, The Netherlands: IOS Press Ebooks. doi:https://doi.org/10.3233/978-1-61499-900-3-439

ISO/IEC 14143-1 Standard. (2007). *Functional Size Measurement* (2 ed.). Geneva, Switzerland: International Organization for Standardization & International Electrotechnical Commission.

ISO/IEC 19761 Standard. (2011). *Software Engineering: COSMIC: A Functional Size Measurement Method* (2 ed.). Geneva, Switzerland: International Organization for Standardization & International Electrotechnical Commission.

ISO/IEC/IEEE 29148 Standard. (2011). *Systems and Software Engineering - Life cycle processes- Requirements Engineering* (1 ed.). Geneva, Switzerland: International Organization for Standardization & International Electrotechnical Commission.

Jayakumar, K., & Abran, A. (2017). Estimation Models for Software Functional Test Effort. *Journal of Software Engineering and Applications, 10*(4), 338-353. doi:https://doi.org/10.4236/jsea.2017.104020

Jørgensen, M., Boehm, B., & Rifkin, S. (2009). Software Development Effort Estimation: Formal Models or Expert Judgment. *IEEE Software, 26*(2), 14 -19. doi:https://doi.org/10.1109/MS.2009.47

Lavazza, L., & Morasca, S. (2019). Empirical Evaluation and Proposals for Bands-based COSMIC Early Estimation Methods. *Journal of Information and Software Technology, 109*, 108-125. doi:https://doi.org/10.1016/j.infsof.2019.02.002

Lesterhuis, A., & Abran, A. (2019). COSMIC Sizing of Machine Learning Image Classifier Software Using Neural Networks. *29th International Workshop on Software Measurement (IWSM) & 14th International Conference on Software Process and Product Measurement (MENSURA). 2476*, pp. 121-129. Haarlem, The Netherlands: CEUR Workshop Proceedings. Retrieved from http://ceur-ws.org/Vol-2476/short4.pdf

Lokan, C. J. (2005). Function Points. In M. Zelkowitz (Ed.), *Advances in Computers* (Vol. 65, pp. 297-347). Elsevier Academic Press.

Lopez-Martin, C., & Abran, A. (2012). Applying Expert Judgment to Improve an Individual's Ability to Predict Software Development Effort. *International Journal for Software Engineering and Knowledge Engineering, 22*(4), 467-483. doi:https://doi.org/10.1142/S0218194012500118

Meridji, K., Al-Sarayreh, K. T., Abran, A., & Trudel, S. (2019). System Security Requirements: A Framework for Early Identification, Specification and Measurement of Software Requirements. *Computer Standards & Interfaces, 66*. doi:https://doi.org/10.1016/j.csi.2019.04.005

Santillo, L. (2000). Early FP Estimation and the Analytic Hierarchy Process. *11th European Software Control and Metrics (ESCOM) & 3rd Software Certification Programme in Europe (SCOPE) Conference*, (pp. 249-258). Munich, Germany. Retrieved from https://www.researchgate.net/publication/2531383_Early_FP_Estimation_and_the_Analytic_Hierarchy_Process

Sellami, A., Haoues, M., Borchani, N., & Bouassida, N. (2018). Guiding the Functional Change Decisions in Agile Project: An Empirical Evaluation. *International Conference on Software Technologies* (pp. 327-348). Porto, Portugal: Springer, Cham. doi:https://doi.org/10.1007/978-3-030-29157-0_15

Singh, J. (2017). *Functional Software Size Measurement Methodology with Effort Estimation and Performance Indication.* Hoboken, N.J: Wiley-IEEE Computer Society Press.

Soubra, H., Abran, A., Stern, S., & Ramdan-Cherif, A. (2011). Design of a Functional Size Measurement Procedure for Real-Time Embedded Software Requirements Expressed using the Simulink Model. *21st International Workshop on Software Measurement (IWSM) & 6th International Conference on Software Process and Product Measurement (MENSURA)* (pp. 76-85). Nara, Japan: IEEE. doi:https://doi.org/10.1109/IWSM-MENSURA.2011.52

Trudel, S. (2012). *Using the COSMIC Functional Size Measurement Method (ISO 19761) as a Software Requirements Improvement Mechanism.* Doctoral Thesis, École de technologie superieure, University of Québec, Montreal, QC. Retrieved from https://espace.etsmtl.ca/id/eprint/1003

Trudel, S., Desharnais, J.-M., & Clout, J. (2016). Functional Size Measurement Patterns: A Proposed Approach. *26th International Workshop on Software Measurement (IWSM) & 11st International Conference on Software Process and Product Measurement (MENSURA)* (pp. 23-34). Berlin, Germany: IEEE. doi:https://doi.org/10.1109/IWSM-Mensura.2016.016

Ungan, E., Trudel, S., & Abran, A. (2018). Analysis of the Gap Between Initial Estimated Size and Final (True) Size of Implemented Software. *28th International Workshop on Software Measurement (IWSM) & 13rd International Conference on Software Process and Product Measurement (MENSURA)*, *2207*, pp. 123-137. Beijing, China. Retrieved from http://ceur-ws.org/Vol-2207/IWSM_Mensura_2018_paper_10.pdf

Ungan, E., Trudel, S., & Poulin, L. (2017). Using FSM Patterns to Size Security Non-functional Requirements with COSMIC. *27th International Workshop on Software Measurement (IWSM) & 12th International Conference on Software Process and Product Measurement (MENSURA)*, (pp. 64-76). New York, N.Y. doi:https://doi.org/10.1145/3143434.3143461

Vogelezang, F., & Prins, T. G. (2007). Approximate Size Measurement with the COSMIC Method: Factors of Influence. *Software Measurement European Forum (SMEF).* Rome, Italy.Retrieved from https://www.researchgate.net/publication/260006048_Approximate_size_measurement_with_the_COSMIC_method_Factors_of_influence

Yılmaz, G., Ungan, E., & Demirörs, O. (2011). The Effect of the Quality of Software Requirements Document on the Functional Size Measurement. *United Kingdom Software Metrics Association International Conference on Software Metrics and Estimating.* London, UK.

Zelkowitz, M. (Ed.). (2005). *Advances in Computers* (1 ed., Vol. 65). Cambridge, MA: Elsevier Academic Press.