SOFTWARE ENERGY CONSUMPTION PREDICTION USING SOFTWARE CODE METRICS

by

Sedef Akinli Koçak

Master of Science in Chemical Engineering, University of Maine, USA, 2001

Master of Business Administration, Ankara University, Turkey, 2001

Bachelor of Science in Chemical Engineering, Ankara University, Turkey, 1997

A dissertation

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Program of

Environmental Applied Science and Management

Toronto, Ontario, Canada, 2018

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A DISSERTATION

SOFTWARE ENERGY CONSUMPTION PREDICTION USING SOFTWARE CODE METRICS

Doctor of Philosophy 2018

Sedef Akinli Koçak

Environmental Applied Science and Management

Ryerson University

Abstract

In recent years, a significant amount of energy consumption of ICT products has resulted in environmental concerns. Growing demand for mobile devices, personal computers, and the widespread adaptation of cloud computing and data centers are the main drivers for the energy consumption of the ICT systems. Finding solutions for improving the energy efficiency of the systems has become an important objective for both industry and academia.

In order to address the increase in ICT energy consumption, hardware technology, such as production of energy efficient processors, has been substantially improved. However, demand for energy is growing faster than improvements are being made on these energy-aware technologies. Therefore, in addition to hardware, software technologies must also be a focus of research attention. Although software does not consume energy by itself, its characteristics determine which hardware resources are made available and how much electrical energy is used.

Current literature on the energy efficiency of software, highlights, in particular, a lack of measurements and models. In this dissertation, first, the relationship between software code properties and energy consumption is explored. Second, using static code metrics regression-based energy consumption prediction models are investigated. Finally, the models performance are assessed using within product and cross-product energy consumption prediction approaches. For this purpose, a quantitative based retrospective cohort study was employed. As research methods, observational data collection, mining software repositories, and regression analysis were utilized. This research results show inconsistent relationships between energy consumption and code size and complexity attributes considering different types of software products. Such results provide a foundation of knowledge that static code attributes may give some insights but would not be the sole predictors of energy consumption of software products.

Acknowledgements

The process of developing this dissertation represents countless hours of contemplation, reflection, and intellectual and personal discovery. I appreciate the constant kindness, patience, and endless support from my supervisor throughout this journey. I would like to express my deepest appreciation to Dr.Ayşe Başar Bener who inspired me, challenged me, kept me going during tough times, and helped me find my way. This work would not have been possible and involved such an interdisciplinary body of research without her guidance and insight.

I also extend my gratitude to my co-supervisor, Dr.Andriy Miranskyy, for all his support, valuable advice, constructive discussions, and technical help throughout this study. I am tremendously fortunate to have had him through this journey. I would also like to thank my thesis committee, for their precious feedback and support.

I would like to thank, for their technical contributions, IBM Canada, who made this research possible. A special thanks goes to Dr.Gülfem Işıklar Alptekin for her guidance as well as her friendship. I would also like to thank all of the graduate students, and post-docs in the Data Science Laboratory at Ryerson University for their friendship and professional help during this adventure.

I thank my parents for their love. They have always been there for me. I am deeply indebted to my husband, Dr.Akın Koçak, for his continuing support and patience. It would not have been possible to complete this research without his love, respect and extensive trust. Finally, I would like to dedicate this dissertation to my daughter, Ipek Koçak, who is my best friend. I am so proud of myself for completing this endeavour but nothing compares with being a mother.

.

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Acronyms

CC	Cyclomatic Complexity. 13
CP	Cross-Product. 104
CPU	Central Processing Unit. 7
DBGEN	Data Base Generator Utility. 32
EC	Energy Consumption. 28
EU	European Union. 2
GB	Gigbayte. 32
ICT	Information Communication Technology. 1
IoT	Internet of Things. 6
LOC	Lines of Code. 13
LOCC	Lines of Code Change. 14
MAE	Mean Absolute Error. 25
MCC	Modified Cyclomatic Complexity. 30
MSR	Mining Software Repositories. 11
OECD	Organization for Economic Cooperation and Development. 1
OLAP	Online Analytical Processing. 32
OO	object oriented. 4

OS Operating System. 35

QGEN Query Generator Utility. 32

RMSE Root Mean Square Error. 25

SD Standard Deviation. 41

TPC-H Transaction Processing Unit H. 32

TWh Terawatt Hour. 3

USB Universal Serial Bus. 32

WEEE Waste Electrical and Electronic Equipment. 3

WP Within-Product. 104

Chapter 1

Introduction

The importance of understanding the relationship between Information Communication Technology (ICT) and environmental issues is widely acknowledged in areas such as energy conservation, climate change, and sustainable resources management [125]. ICTs and their applications may result in positive and/or negative effects on the environment. Green ICT is the study and practice of using computing resources efficiently [106].

Climate change is a major environmental concern. At the recent Paris Climate Conference (COP21) in December 2015, it was stated that it is necessary to "undertake rapid reductions of carbon emission" [145] and to move towards renewable energy sources and low- and non-carbon-dependent energy sources. Meaning that energy consumption in all sectors needs to be minimised. While researchers are often concerned with the carbon footprint of different sectors and activities, waste management and energy consumption of the overall ICT sector, they rarely consider the role of software in the environmental impact of computer technologies. Until recently, most of the work done within the industry regarding environmental concerns focused on improving the energy efficiency of hardware. The idea of considering energy efficiency in the software context alone was not fully acknowledged until the "Greensoft" model was introduced by Naumann et al. [141]. Since then, the trend has changed and a software related context has emerged.

Green ICT is a broad field that includes the entire system life cycle from hardware manufacturing, to minimizing computer-related waste, to software development. Being green in ICT simply means being efficient in using natural resources and on the impacts of ICT on CO_2 emissions. This is either related to the power consumption of systems or to the role of system as an enabling technology for conserving energy in various fields [43]. Thus, resource usage is an important issue in green ICT [179]. The Organization for Economic Cooperation and Development (OECD) defines "Green ICT" as "an umbrella term for technologies with better environmental performance than previous generations (direct impacts) and ICTs that can be used to improve environmental performance throughout the economy and society (enabling and systemic impacts)" [125]. Direct impacts refer to positive and negative impacts due to the physical existence of products (goods and services) and related processes. Enabling impacts refers to positive and negative impacts due to applications that reduce environmental impacts across economic and social activities [52, 78]. Systemic impacts on the environment are rooted in behaviour and behavioural change. Systemic outcomes of green applications largely depend on end-user acceptance, lifestyle adjustments, and changes in collective social behaviour. Systemic impacts are mostly related to rebound effects [78]. For example, increased technological efficiency may lead people to use even more of the technologies which, in total, consume more energy. This dissertation is focusing on direct impact due to the physical existence and usage of software products.

Energy consumption is now becoming a major concern in society. Although, software engineering tends to focus on the technical elements and artificial systems with clear boundaries, identifiable parts and connections, modules and dependencies, there is little understanding of how it is perceived by software engineering professionals and how energy concerns can become an embedded part of the software engineering process [38]. Existing or new software systems need to be redesigned to address environmental issues and support environmentally sustainable business models and processes.

Predictive models become crucial to guide software practitioners in their decision-making processes. Prediction models can be applied during different development processes to estimate the various properties of software product [22]. For example, to predict defective parts of the software [56, 124, 139, 186], to assess the reliability of the system [56, 130], and cost/effort estimation [187] of the system. Energy consumption detection of the software product is mostly relies on direct measurements, but this is time consuming [83, 155] and needs investment. Similar to many prediction research in software engineering, using software metrics allow software companies to estimate the impact of code features on energy consumption.

1.1 Motivation

There has been a focus in both practice and research to reduce the energy impact of computational systems. Many earlier studies focused on the hardware energy efficiency ([184, 59, 25, 207]). However, hardware is driven by software. Therefore, software energy impact is critical to the system's overall energy consumption.

Many countries have passed laws on clean and efficient energy and environmental protection. For example, the Environmental Protection Legislation in the European Union (EU)

2012/19/EU [54]-(Waste Electrical and Electronic Equipment (WEEE)) focuses not only on the physical waste of computing equipment, but also on the production, collection, storage, processing, presentation, and communication of information by electronic products and equipment. It also focuses on the "producer responsibility", meaning that producers accept responsibility when they design their products to minimize environmental impacts. This responsibility clearly indicates the link between producer responsibility and design and production changes for the industry. In order to have an energy efficient end product, energy consumption must be taken into consideration when designing the software. It is commonly accepted that software characteristics ultimately determine, to a large degree, the energy efficiency of a software system [83, 103]. Therefore, the focus energy concerns should also include software optimization. An effective way of determining the software energy impact at the design stage is to predict its energy consumption before completing its coding. The major difficulty of energy consumption prediction is the lack of data, since the energy consumption measurement of software products are relatively new and there is a lack of such product data available in software repositories. For this reason, the motivations of this dissertation are:

- Investigating relationships between software code metrics and energy consumption by using data from various open source products and from a closed source product, and
- Building a prediction model using regression by applying within-product and cross-product data.

Although static code attributes may not be the sole predictors of energy consumption of software products, such a model would provide some insights to software development managers before the code is executed so that any code changes may be made before the production and maintenance stages of the product development life cycle.

1.2 Problem Definition

The relative share of worldwide electricity consumption of communication networks, data centers and personal computer products and services was about 4.7% in 2012 with 920 Terawatt Hour (TWh) [1] of the 19,000 TWh [191]. This consumption proportion is predicted to increase to 14.5% of worldwide power consumption in 2020 [193]. As a consequence, average monthly CO_2 concentration exceeds 400 parts per million (2016) [143].

Rapid expansion of demand on hardware and software systems, such as the use of mobile devices and smartphones, web-services, and the widespread adaptation of cloud systems have

[1]1TWh is 10E+12 watt-hours

aspects of energy consumption growth. It is projected that the energy costs (the financial impact of energy consumption) of operating a typical system will exceed building/producing such system [90]. Rapid transition in the information era, has signaled a vast increase in resource demand and energy consumption that is not expected to stop anytime soon. Simultaneously, software product size and complexity are increasing, including the overall complexity of decisions surrounding software development. Additionally, there has been a significant increase in awareness of resource consumption of software systems. For example, companies with large data centers are starting to inquire as to whether or not the energy consumption problem should be tackled from a software perspective, with the goal of controlling the growing trend of computing capacity requirements [33]. Regarding the software products aspect, companies are now facing challenges with how to build innovative products that not only meet the customers needs, but are also environmentally friendly [148]. As a result, the energy efficiency of software is becoming increasingly important [155, 90, 83].

Although, there have been many researchers who work on power consumption and monitoring of embedded systems, the energy consumption of software has recently been gaining the attention of research community [183, 173, 144]. Software engineering does not currently provide consolidated knowledge on the relationship between software products and its energy consumption [82, 128, 129, 5, 154].

Regarding software properties, size, object oriented (OO) and churn metrics have been analyzed in prediction studies, especially defect prediction in software engineering. These metrics are easy to use and widely used [123]. Size, complexity and churn attributes can be automatically and easily collected, even for very large systems. In software energy consumption studies, an evidence of a potential relationship between size and churn software metrics and energy consumption has been found [80, 82, 83, 67, 5, 129]. In addition to size and churn metrics, Miranskyy et al. [129] validated that there is a perfect correlation between execution time and energy consumption.

Execution time may be determined by static program analysis based on path information for the program, such as identification of instructions, and the control-flow graph of the code [156]. Cyclomatic complexity measure approach is to measure the number of independent paths of a program. Program complexity can be controlled by a control graph which includes branching into/ out of a loops and decision paths [116]. Theoretically, control flow of a program depends on problem settings. In the problem setting values of variables used in conditions (i.e., loop conditions or conditions of alternatives) and the values of pointers to functions determine the control flow. As a consequence, the execution time may be effected. The actual time behaviour of loops may differ from their behaviour within the program[156]. In practice, non-deterministic polynomial completeness of the problem may affect the execution time of such loops [147]. Complexity, therefore, could be a good indicator of energy consumption when considering control flow and the execution time approach and also considering strong relationship between execution time and energy consumption as a measure of complexity. On the other hand, complex code does not necessarily mean slow. Complex code may be very efficient but it is also hard for a human to comprehend. That is why the defects often occur in complex parts of the code. Hence, complexity attribute enriches the information content of defect prediction models.

In this study, the aim was to better understand the relationship between code characteristics and energy consumption at an early stage of the software development life cycle. As it was explained earlier, static code attributes can easily be extracted from the source code in an automated manner, and therefore they may be used to gain insights for the energy consumption of software product.

1.3 Organization of the Dissertation

The dissertation is structured into the following chapters.

- *Chapter 2* discusses related work in the literature and gives background information and some related definitions. The research question is provided.
- *Chapter 3* provides research design details accompanied by an explanation of each method used. The Chapter ends with a proposed model.
- *Chapter 4* presents the data collection approach for both observational and mining software repositories followed by descriptive statistics of datasets.
- *Chapter 5* demonstrates an empirical evaluation of energy consumption using software static code metrics. This chapter also introduces proposed energy consumption prediction models and cross-product energy consumption prediction approach.
- *Chapter 6* concludes the dissertation with an evaluation of the research question, theoretical and practical contributions, threats to validity, and future work.

Chapter 2

Related Work and Research Question

Software engineering research has, for the most part, focused on increasing the reliability, efficiency and cost-benefit relationship of software products for their owners, by focusing on processes, methods, models and techniques to create, verify and validate software systems and keep them operational [148]. But, there is more to it than just operation and cost-benefits: it has been suggested that every line of code has not only financial and technical implications [24], but also environmental implications [148].

As ICT use continues to grow, so does the demand for energy. For example, laptop and desktop machines for personal and business use, mobile systems, smart devices in Internet of Things (IoT), data centers of different sizes, and large computing clusters for different purposes. Thus, energy management of systems has become important in almost all computational domains. Optimization of hardware to save energy has been studied by better understanding software-hardware interactions. For mobile systems, energy analysis and optimization techniques have been studied. Since battery life has long been an important issue with customers, energy optimization inside a hardware component through processor, material, mechanical, device, circuit, architecture design [59, 207, 60, 192, 190] has been improved. Additionally, optimization hardware behavior for energy savings through software[1]-hardware interactions has been studied.

[1]especially the operating systems

2.1 Software and Energy Consumption

Although, energy management of hardware systems is the predominant driver for resulting energy consumption, software itself has a considerable influence on energy consumption [34]. Many researchers have emphasized that reducing the power consumption of software is becoming more important in many environments, such as mobile systems [211], embedded systems [183] and data centers [17]. Although industrial interest in software power consumption is limited, mobile device and mobile software development companies have an increasing interest in energy efficient tools and applications, such as Power Tutor [48] an Android power monitor supported by Google. Another example is power consumption tooling for applications running on a Windows Phone 7 platform, provided by Microsoft Research [67, 68].

The importance of software, regarding the energy consumption of computational systems, was first stressed in Kaushik and Johnson [161]. They indicated that software directs much of the activity of the hardware. Consequently, the software has a substantial impact on the power dissipation of computational systems. In the literature, most of work on energy management has been on *measurements* and *optimization* of the hardware/software system.

Regarding *measurement*, much of the previous work has attempted to reduce power consumption by improving the power-efficiency of individual hardware components. Tiwari et al. [183, 184] conducted one of the earliest studies on power measurements. They modelled the energy of a Central Processing Unit (CPU) instruction using a base energy, as well as a transition energy for each pair of instructions. Lorch and Smith [110] discussed software techniques to utilize power saving provisions provided by various hardware components, such as CPUs, disks, displays, wireless communication devices, and main memory. As power consumption is not just CPU related, several studies have focused on the energy efficiency of software by addressing memory related power consumption [25]. Akinli Kocak et al. [102, 128] concentrated not only on CPU and memory, but also I/O activities and storage, while others have examined the energy efficiency of the operating system at a routine or system-call level [107] by transcoding the received content [177]. All of these studies focused on energy consumption related to the system resources within specific hardware architecture [86, 206].

Gurumurthi et al. [69] introduced a power simulator, SoftWatt, to simulate power consumption on mainly hardware components such as CPU or memory systems. Amsel et al. [8] designed a tool, GreenTracker, that simulates a real system's power usage based on CPU measurements, and benchmarks individual application power. They measured the system in real time. Gupta et al. [67] described a method for measuring the power consumption of applications running on a Windows Phone 7.

Regarding *optimization*, a primary goal of energy consumption measurements is to optimize the system for reduced energy consumption. Li et al. [108] applied the idea of load balancing to server-room heating and cooling and Fei et al. [55] used context-aware source code transformations and achieved power consumption reductions of up to 23% for their software. Selby [167] investigated methods of power reduction by code transformations and compiler optimizations.

To date, the research community is particularly interested in measuring systems' resource usage performance by taking into account CPU, memory and hard disk, input/output operations and network. In the last few years, besides measurements, research on mining software repositories to investigate the changes in power consumption or performance over revisions is gaining attraction: Shang et al. [168] investigated performance changes over versions of software; and Gupta [67] worked on mining Windows Phone 7 power consumption traces. A set of case studies were also conducted by Hindle [80, 82, 83, 209]. He first introduced a "green mining" methodology of relating software change and configuration to power consumption in Firefox and RTorrent. Then, Hindle and Zhang [209] observed that power consumption can vary between versions; however, CPU usage is not enough to model power consumption. Then they opened their dataset to the research community.

2.2 Static Code Metrics and Predictive Models in Software Engineering

In this section, related work in the area of predictive models in software engineering are presented. Predictive models in software engineering is a mature research area. There are many studies in defect prediction [121, 124, 188, 28, 131] where software code metrics were used.

Static code metrics, as predictors in software engineering, have been widely adopted [120, 28, 101, 121, 142] A software failure occurs when a service delivered by software deviates from fulfilling its intended functionality due to problems in the code, process, or people; error is the discrepancy between the delivered and intended functionality [194]; and the cause of an error is a defect [188]. Therefore metrics that measure product, or process, or people characteristics are good indicators of defects. Static code attributes are the characteristics of the source code and that is why they have been extensively used in defect prediction studies. Various software metrics have been investigated, including size metrics (e.g., lines of code), complexity metrics (e.g., McCabe‘s cyclomatic complexity [122, 131]), process/churn metrics [72], and social network metrics [18].

Different metric categories show different performances in these prediction studies. For example, D'Ambros et al. [50] systematically compared the predictive power of different metric categories, and found that process metrics are superior in predicting the defect proneness.

Misirli et al. [131] found that code and network metrics vary with defect category. Network metrics have higher effect than code metrics for defects reported during functional testing and in the field, and vice versa for defects reported during system testing. They also reported churn metrics are the best for predicting all defects. Most defect prediction studies combine well-known methodologies such as statistical techniques [214, 188, 186] and learning algorithms [189, 122, 28]. While some researchers prefer regression models, others use more complicated models like Bayesian Networks that include causal relationship between project and process metrics [130].

Energy consumption of software may not be so tightly related to code characteristics as seen in defect prediction. However, code characteristics such as size, and complexity of the code may result in higher resource consumption. Size and complexity are successful predictors of productivity [93]. A significant relationship was found between software size and both productivity and defect rate [111], implying that larger projects are more productive and have lower levels of defects [93]. Productivity is also related to program performance. Program performance or how quickly a program solves a specific problem is influenced by a number of factors, some of which are developer controlled and others that are dependant on the hardware and software environment of the programs execution. Program execution time is related to energy consumption of system, thus energy consumption may be considered as a performance measure. Although performance are increased with more powerful systems without software productivity gains. However, software development companies are constantly looking for ways to increase both productivity and code quality. Although it cannot be sure about the exact energy consumption of the code unless the code is executed and runtime metrics are collected. However, static analysis may give us useful insights before the code is executed, especially if there are such data readily available for the purposes of building various predictive models and/ or performing software analytics.

Regarding energy consumption prediction, “green mining” is the first attempt to leverage historical information extracted from the publicly available software to model software power consumption [83]. Hindle [83] explored the feasibility of power consumption prediction based on OO metrics and churn metrics using three open source projects. He demonstrated that software change can affect power consumption. He also found evidence of a potential relationship between some software metrics and power consumption. In another study, Gupta et al. [67] investigated power consumption of one mobile software product by mining software repositories and quantified power consumption in different modules within the same software. Miranskyy et al. [129] studied the effect of software metrics on energy consumption and execution time of one open source database product.

All of these energy consumption related studies are complementary to this research. The main focus of this research is to investigate to what degree static code metrics and energy consumption are related, but also proposing models for energy consumption prediction using a regression on various open- and closed-source software products.

Within-Project and Cross-Project Predictions: Cross-project prediction research that has used metrics from one project to predict characteristics of metrics in a different project. Most of the research in this area mainly focused on cost estimation and defect prediction. For example, cost estimation models such as SLIM by Putnam and Fitzsimmons [157], Function Points by Albrecht and Gaffney [6], and COCOMO by Boehm et al. [21] have provided general purpose models that can be applied to arbitrary projects. As a consequence, studies have been undertaken to address both cross-projects and within-projects effort estimation. Most research has addressed effort estimation in cross-project models [199, 95, 118]. Kitchenham et al. [96] reveals that, although some software development companies would benefit from cross-project benchmarks, there is no clear indicator of when it works most effectively. After Kitchenham et al.[96], Turhan et al. [189] and Zimmerman et al. [213] analyzed different open source products to address defect prediction across projects based on static code measures such as code churn (code modified, added, deleted) and code complexity, metrics. Turhan et al. [189] found that cross-project data dramatically increases the probability of detecting defective modules (i.e. 22% increase); Zimmerman et al. [213] found a low ratio of cross-project prediction performance (i.e., 3.4 %). They also addressed how the domain and the process influence cross-project predictions.

Canfora et al. [32] explicitly took into account the trade-off between effectiveness and inspection cost proposing a multi-objective approach for cross-project defect prediction. Their approach is based on a multi-objective logistic regression model built using a genetic algorithm. Recently, Minku and Yao [127] investigated the best use of cross-project data, and proposed a framework to learn the relationship between cross-project and within-projects explicitly, allowing cross-project models to be mapped to the within-project context. Their framework achieved similar/better performance to within product results with less within-project data than a corresponding within-project model.

Prior to this dissertation, no study has been conducted with cross-project models in the context of energy consumption prediction. This dissertation is the first research to asses energy consumption prediction across products based on static code metrics. The data originates from within-product data. Here, prediction is used as a “cross-product prediction” if a model learned from one product and the prediction is performed on another product. Most of the cross-project studies have faced challenges in their models related to variations in the distribution of predictors. To overcome this challenge, software metrics were transformed in both within-product and and cross-product to make them more similar in their distribution [140, 210].

2.3 Background and Definitions

This section summarizes the background on the energy impact of ICT and the importance of energy efficiency in a software context. Additionally, there is a discussion on code properties and software metrics.

2.3.1 Energy Efficiency of Software, and Definitions

Energy efficiency is simply the goal of reducing the amount of energy required to provide products and services. Something is more energy efficient if it delivers more services for the same energy input, or the same services for less energy input [2]. Rising energy costs, energy hunger applications, limited battery life, and the high performance demands require optimization in computers and mobile devices. In this regard, there are various approaches aimed at achieving energy saving opportunities. Although software does not consume energy by itself, its characteristics determine which hardware resources are made available and how much power is used. Therefore, the energy efficiency of a software product has recently become a popular area of research in academia [81, 27, 104, 154].

Definition:

Energy and Power: In the literature, energy and power are used interchangeably. However, they represent different values. Therefore, it is important to provide a clear definition at the outset. The simplest definition of *energy* is the "capacity to do work". Energy is defined as the amount of electricity consumed per unit of time [164, 47]. Energy is amount of work, so the total amount of electricity consumed.

$$Energy = Power \times Time, \tag{2.1}$$

In this research, all the energy measurements are written as "energy consumption" with unit of watt-hour, which is an electrical energy equivalent to a power consumption of one watt for one hour.

2.3.2 Energy Consumption and Mining Software Repositories

Data mining is an important methodology used in various research today. Researchers are building prediction models and pattern identification tools and techniques by mining the large repositories of data generated through the use of information technology in various domains. These data are useful in conducting research in many areas of software engineering, such as, software reliability, finding developer expertise, quality of software, resource utilization, effort, cost and time estimation, dependency analysis, defect prediction, and impact analysis [89]. Mining Software Repositories (MSR) field analyses the rich data available in software repositories to uncover interesting and actionable information about software systems and projects. Examples of software repositories are:

- *Historical repositories* such as source control repositories (e.g., Mercurial), bug/defect tracking repositories (e.g., Bugzilla), and historical information about the evolution and progress of a project (e.g., Promise) [71, 73].
- *Run-time repositories* such as deployment logs, contain information about the execution and usage of an application at a single site or multiple deployment sites [71, 73].
- *Code repositories* such as Sourceforge and GitHub contain the source code of various applications developed by multiple developers [71, 73].

By mining these repositories, useful and important patterns and information on software projects can be revealed and used to shape the future projects [65]. Recently, MSR was used to conduct energy related research from software engineering perspective. One of the first studies in this area was conducted by Gupta et al. [67]. They focused on combining MSR techniques with power performance and introduced a method for gathering and analyzing power data on mobile device running Windows Phone 7. With the methodology, they quantified and detected differences in power usage of the mobile device. Aditional studies were conducted by Hindle [81, 81, 83]. He combined the MSR research and energy consumption by studying multiple versions of the three open source projects and their characteristics of energy consumption patterns. He also studied the relationship between code size, churn metrics and energy consumption. He concluded that further study is required to establish relationships if they exist.

2.3.3 Software Metrics

As with any prediction problems in software engineering, software energy consumption prediction necessitates a set of known features (i.e., predictive variables) to characterize the problem and to give an estimation on the energy consumption of the system (i.e., response variable). These features are related to software size, software complexity, and the development process. They are identified with software metrics, which are a standard of measure of a degree to which a software system or process possesses some property [126, 57].

High software energy consumption can be considered as undesirable software deficiency, such as high defect or high cost that needs to be predicted to guide software practitioners in their decision-making process [148]. As seen from software defect and cost/effort estimation prediction research (see section 2.2), software product and process metrics have been successfully utilized in prediction models. There are three major software metric groups: process, product, and people. While product metrics are derived from the software product itself, process metrics are derived from the processes that are employed to build the software product. People metrics relate to the quality of people working on the software projects [126].

In the following subsections, a summary of product and process metrics that are relevant to this dissertation is given.

Product Metrics

Product attributes describe a software product in a way that is dependent only on the product itself. One of the most useful features is the size of the software. This metric can be measured statically without having to execute the product. Product metrics are also called *static code metrics*, because they are directly extracted from the product source code [188]. Static code metrics indicate about the size and complexity of the implemented code. Analysis of static code metrics is useful for ensuring the quality of later stages, such as testing and maintenance, which are the most resource consuming stages of software development [105, 188, 57].

Size Metrics: Information about the size of the software product can be very useful. Size attributes of the design and code can determine the effort required for testing, as well as the effort required to add features. The most commonly used code size metric is *Lines of Code (LOC)*, which is a simple line counts of source code. It is a measure of size for code, not for requirements or design documents. There are a number of different LOC metrics. These include, but are not limited to: [105, 57].

- Total Lines of Code: A total line count of source code, where one line equals one count.
- Blank Lines of Code: A blank lines count of source code.
- Lines of Commented Code: A count of code lines that includes comments. Comments are usually in block forms before the actual implementation of methods, providing a generic description.
- Lines of Code and Comment: A count of code lines that includes both executable statements and comments.
- Lines of Executable Code: A count of the actual code statements that are executable (i.e., total lines of code after the blank and commented lines are subtracted).

Complexity Metrics: Complexity is usually an undesired property of software because, it makes software hard to read and understand and hence is harder to change [77]. There are several decades of research on finding the best complexity metrics that reflects the complexity of code. The most commonly used type of metric is *McCabe Cyclomatic Complexity (CC)* [116]. This measures the number of linearly independent paths of a program execution (the control flow within a module). The greater the number of paths through a module, the higher the complexity. Another type of complexity metric is *Halstead's Complexity Metrics*. Halstead metrics are the total number of operators, total number of operands, unique number of operators, and unique number of operands [57]. While McCabe's complexity directly measures the number of linearly independent paths with a single metric, Halstead's complexity uses the number of operands and operators to calculate the complexity.

Process Metrics

Process metrics provide measurements for software process improvement. They are usually associated with a time scale. Process activities have duration, they occur over time and they may be ordered or related in some way that depends on time. For example, number or fault/defect during formal testing or mean time to failure during testing [57].

Code Churn Metrics: Code churn is a measure of the amount of code change taking place within a software unit over time. It is easily extracted from a system's change history, as recorded automatically by a version control system [134]. *Lines of Code Change (LOCC)* is total number of changed codes between two versions of the same software. These metrics are helpful for assessing the risk factor of code development and implementation (i.e., poor code quality, or high number of post-release defects) by measuring the degree of change in source code. Code churn metrics provide the following information [188]:

- number of added lines of code,
- number of deleted lines of code,
- number of modified lines of code,
- frequency of the code change,
- status of the code (i.e., new, changed, unchanged),
- number of changes made on the code (i.e., commit count),
- number of distinct developers who worked on the code,

- whether the code is modified by a developer, who has not created it, and
- whether a developer is working on that code for the first time.

2.4 Research Question

It has been observed that there is much interest in the energy efficiency of software, but not a lot of focus on the characterization of energy consumption and prediction using code attributes. The goal of this dissertation is to (1) contribute to the effort of reliable energy consumption measurement using code attributes, (2) predict the energy consumption of software using a linear-regression-based approach with learning method, and (3) evaluate cross-product energy consumption prediction approach

This dissertation sought to answer the following Research Question:

RQ: How can we predict energy consumption of software product using static code attributes?

Chapter 3

Research Methodology and Proposed Model

3.1 Research Approach

The focus of this research is to predict the energy consumption of software product. Following this objective, the purpose is two-fold: First to investigate the relationship between software code metrics and energy consumption, second, to characterize the change in energy consumption software over time and to predict the energy consumption of software using software static code attributes. Four open source software products and one closed source commercial software product were examined.

In general, a quantitative approach was used in the research design [45]. Quantitative research aims to obtain a numerical (quantitative) relationship between several variables or alternatives under examination [88].

The broad research approach is the plan or proposal to conduct research, involving philosophical assumptions, research designs, and specific methods. Similar to other research fields, in order to perform research in the software engineering, methods that are available, their limitations and when they can be applied have to be fully understood. With the guidance of the main research approaches put forward by Basili [11] and Tichy et al. [181], and followed by Glass [64, 63] following are the summarized four research approaches in the field of software engineering.

- *Scientific:* A theory is developed to explain a phenomenon; a hypothesis is proposed and data is collected to verify or refute the claims of the hypothesis.

- *Engineering:* Solutions are developed and hypotheses are tested. Based upon the results of the test, the solution is improved until it requires no further improvement.

- *Empirical:* A statistical method is proposed as a means to validate a given hypothesis. Unlike the scientific method, there may not be a formal model or theory describing the hypothesis. Data is collected to verify the hypothesis.

- *Analytical:* A formal theory is proposed and then compared with empirical observations.

Although application of an approach differs from one field to another, some approaches may be seen as variations of others. For example, the engineering and the empirical can be seen as variations of the scientific approach [11, 172].

Empirical research studies play a fundamental role in software engineering [162]. Although software engineering stems from the technical field, it also is recognized as a multi-disciplinary field, which includes interactions between many social and technological backgrounds. To understand how software engineers construct and maintain complex, evolving software systems, there is a need to investigate, not just the tools and processes they use, but also the social and cognitive processes surrounding them [172, 51]. Therefore, empirical research helps to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies using empirical observations or data that is collected.

According to Wohlin [204] there are two types of empirical research approaches in software engineering.

- *Exploratory research* is concerned with studying objects in their natural setting and letting the findings emerge from the observations. This implies that research is primarily informed by qualitative data. The subject is the person, who is taking part in an empirical study in order to evaluate an object.

- *Explanatory research* is mainly concerned with quantifying a relationship or comparing two or more groups with the aim to identify a relationship. The research is often conducted through setting up a controlled experiment. This type of study implies that factors are fixed before the study is launched. This is also referred to as quantitative research, as it is primarily informed by quantitative data.

In software engineering most studies have strongly focused on the quantitative approach; one of the earliest methodologies was presented by Basili et al. [10], and the first experimental methodologies were presented by Wohlin et al. [202] and by Tichy [182]. Qualitative approaches were discussed later by Seaman [166], and case studies in software engineering by Kitchenham et al. [94], Wohlin et al. [203], and Runeson and Host [162]. A comprehensive review of empirical research for software engineering was presented by Shull et al.[172] and a preliminary guideline was presented by Kitchenham et al.[97]. It is possible for qualitative and quantitative research to investigate the same topics, but each of them will address a different type of question.

Depending on the purpose of the empirical investigation, three major types of methods are identified by Wohlin et al. [204] that are believed to be the most relevant to software engineering research: survey, case study and experiment. Runesaon and Host [162] counted action research and Easterbrook at al. [51] added ethnographies to the list afterwards.

Zelkowitz and Wallace [208] and recently Juristo [88, 87] pointed out that experimentation is often misused in the software engineering. Researchers must understand and carry their research out properly when they conduct empirical studies, especially experiments. Over time, many different experimental sub-types have been developed in response to the needs of different fields. As Juristo and Moreno [88] stated, software engineering is similar to medicine with regards to experimentation. Empirical studies in medicine are classified in two broad categories: analytical and descriptive. Descriptive studies aim to generate hypotheses about associations between exposures and outcomes. Analytic studies are then undertaken to test specific hypotheses. Analytical studies are further sub-classified as observational or experimental studies

The differentiating characteristic between observational and experimental studies is that in the latter, the presence or absence of undergoing an intervention defines the groups. By contrast, in an observational study, the researcher does not intervene and simply "observes" and assesses the strength of the relationship between the variables [113, 87].

There are two types of experimental methods: true experimental and field trials, which is same as quasi-experimental method, and three types of observational studies: cohort studies, case-control studies, and cross-sectional studies. Cohort studies can be classified as prospective or retrospective. The word "cohort" has been used to define a set of subjects followed over a period of time. Prospective studies are carried out from the present time into the future. Retrospective cohort studies, also known as historical cohort studies, are carried out at the present time and look to the past to examine events or outcomes [160, 176]. In case-control studies, starting point is the identification of condition of interest, and suitable controls without that condition. Cases and controls are then compared to assess whether there are any differences in their past condition. Cross-sectional studies look at each subject at one point in time only. Data are collected from subject as a single time [113, 160].

Similarly, Zelkowitz and Wallace [208] summarized experimentation in software engineering into three categories: observational, historical, and controlled based on various data collection methods. This study is classified as follows:

- According to classification of Wohlin [203, 204] classification and followed by Juristo and Moreno [88] identification this research is explanatory empirical research that uses quantitative methods to explain the relationship between code characteristics and energy consumption of a software product.

- According to classification of Zelkowitz and Wallace [208] this research is observational retrospective cohort study where mining software repositories and machine learning methods were used.

Figure 3.1 shows adopted research design.

The overall prediction study was designed using methodology adapted from Shearer [170] and Bener et al. [16].

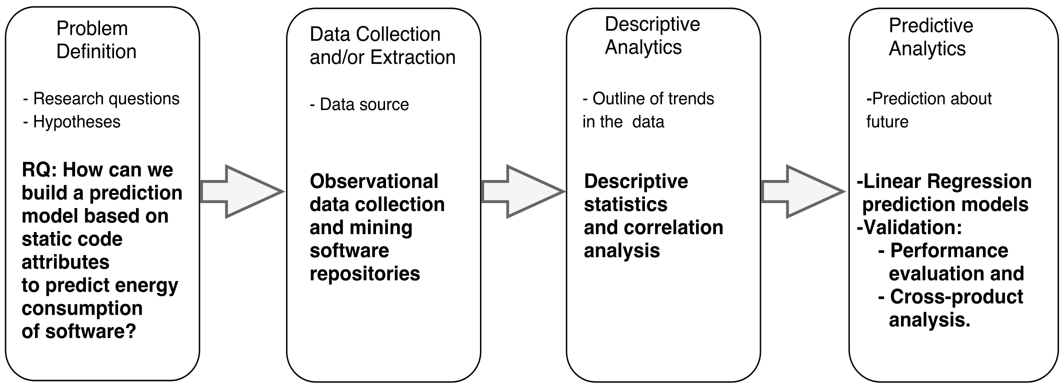

Problem Definition
- Research questions
- Hypotheses
RQ: How can we build a prediction model based on static code attributes to predict energy consumption of software?
Data Collection and/or Extraction
- Data source
Observational data collection and mining software repositories
Descriptive Analytics
- Outline of trends in the data
Descriptive statistics and correlation analysis
Predictive Analytics
-Prediction about future
-Linear Regression prediction models
-Validation:
- Performance evaluation and
- Cross-product analysis.

Figure 3.1: Overview of the study methodology in energy consumption prediction adopted from Bener et al. [16].

Following in the steps of Figure 3.1, the aim is to predict energy consumption of software using code attributes by by examining the relationship between energy consumption and static code metrics. This aim then was transformed into the research question “How can we build a prediction model based on static code attributes to predict energy consumption of software?”. The model is able to learn from the metrics that are available, and can predict the energy consumption of the software product. To do that, it was decided that the inputs of the model are size, complexity and churn metrics. The second step was data collection, which includes observational data collection and mining software repositories. The third step was descriptive analytics which is the simplest way to get an insight of the data [16]. In this research, descriptive statistics, correlation analysis and visualizations were used to highlight the relationships. In the forth step, predictive analytics were designed by using linear regression prediction models. The models were validated using performance evaluation measures and followed by cross-product energy consumption analysis. Details are given in section 3.3.

3.2 Research Methods and Design

Selecting an appropriate research method is important to the success of any research, and must be driven by the research question and the state of knowledge in the area being studied. In empirical software engineering research, Easterbrook et al. [51] points out that it is not easy to decide which research methods are suitable for the research. A combination of research methods may be the most effective in achieving a particular research objective. For example, when a subject area is not well understood, using more than one method may have complementary strengths and when used together can lead to a more comprehensive understanding of a phenomenon. Different research methods serve different purposes; one type of research method may not fit all purposes. Based on purposes, Runeson and Host [162] classify four types of research:

- *Exploratory:* Finding out what is happening, seeking new insights and generating ideas and hypotheses for new research.
- *Descriptive:* Portraying a situation or phenomenon.
- *Explanatory:* Seeking an explanation of a situation or a problem, but not necessarily in the form of a causal relationship.
- *Improving:* Trying to improve a certain aspect of the studied phenomenon.

In empirical software engineering studies, the most common methods used are: survey, experiment, case study, and action research. The most famous guideline for research methods in the empirical software engineering field is proposed by Kitchenham and Pfleeger [150, 98] in a series of articles for ACM SigSoft Software Engineering Notes. In empirical research, data collection is performed by using observations and measurements. Observational data are occurrences that can be simply recorded. Measurement data can be counted, calculated, or quantified. As Erdogmus stated [53], observations provide deeper insight in areas in which measurements serve only as proxies for other constructs.

By reviewing the guidelines, exploratory type retrospective cohort study design was chosen as the best design for the purpose of this dissertation.

In the following subsections, the methods used in this dissertation are described in more detail as they are used in software engineering. Because these methods were adapted from a number of different fields to software engineering, there is no consistent terminology to describe them. The terms and methods and their definitions were used from the software engineering context, which is familiar to software engineers.

Let us explore the methods that are used in this research in details.

3.2.1 Observational Method

The term experiment is often used synonymously with observation. However, researchers must distinguish between experiments, where at least one treatment or controlled variable exists, and observations where there are no treatment or controlled variables [12]. According to Runesan and Host [162], observations can be conducted in order to investigate how a certain task is conducted by software engineers. Basili [12] characterized observational studies by whether or not a set of study variables are determined in advance. According to Basili, a set of study variables are predetermined by the researcher separates the observational study from qualitative studies.

Observational study was conducted in this dissertation to determine possible relationship between energy consumption and software static code metrics across software product releases. All the variables were predetermined prior the study. As classified in Rosenbaum [160] and Mann [113] there are three observational methods: cohort, cross sectional, and case-control studies. Retrospective cohort allow this research to trace back certain product characteristics that are believed to contribute to product energy consumption. A prospective cohort is an empirical method where subjects, according to past or current exposure and follow-up into the future are used to determine if the outcome occurs.

According to the motivation of the RQs, the retrospective cohort method was used in an exploratory fashion to seek the relationships between the energy consumption of a software product and its code attributes. General design methodology follows the process and guidelines provided by Basili et al.[12], Wohlin et al. [204], Perry et al. [149], and Hindle [83]. Figure 3.2 shows the overview of the observational methodology.

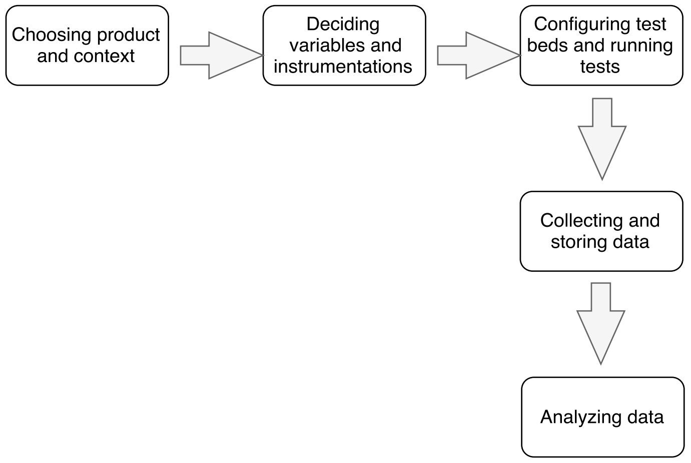

Choosing product and context
Deciding variables and instrumentations
Configuring test beds and running tests
Collecting and storing data
Analyzing data

Figure 3.2: Overview of the observational method.

3.2.2 Mining Software Repositories (MSR)

Data mining for software engineering has been used to describe a broad class of investigations into the examination of software repositories. It mostly focuses on how to improve new and current software projects using past project's data. Software practitioners and researchers recognize the benefits of mining this information to use in many areas. For example, data mining technique is used to obtain historical patterns to inform current issues when managers or developers are not certain about an important decision [123]. The field analyzes and cross-links the rich data available in software repositories to uncover interesting and actionable information about software systems and projects. Historical repositories, run-time repositories, and code repositories are examples of software repositories [71]. Historical information can assist developers in understanding the rationale for the current structure of a software system. Research and applications are now proceeding to uncover the ways in which mining these repositories can help to understand software development and software evolution, to support predictions about software development, to use operation system logs to predict software energy consumption, and to exploit this knowledge in planning future development [133, 71].

There are many open source software repositories, and many methods are available to developers and researchers. One simple and direct method is to check these code files one by one and extract the useful ones. However, the size of data in the available repositories for mining continues to grow rapidly. Therefore, to find useful information easily, we need automated approaches [71]. A number of automatic techniques are present to analyze the software repositories for reuse in software development, such as different search algorithms or association rule miners [3], or hybrid approaches that combine different mining rules and search algorithms.

3.3 Proposed Model

In this research, one or multiple variables linear regression models and fixed/random effect regression model were used as it a widely used model in the literature in analyzing clustered datasets. Number of examples are [75, 76, 4, 1, 109, 58] where fixed/random effect model were used. Fixed/random effect modelling approach was proposed for DB2 dataset. One- or multiple variables regression for each DB2 version groups dataset, MYSQL, Firefox, Vuze, rTorrent datasets to improve decision making for software managers by predicting the energy consumption. Linear regression model is used for regression problems in which the dependent variable is a continuous or discrete value.

3.3.1 Linear Regression:

Linear Regression is a statistical technique for determining the relationship between a single dependent variable (a.k.a output variable, response variable) and one or more independent (a.k.a input, explanatory, predictor) variables. In regression, the desired output consists of all continuous variables [198]. The goal of regression is to predict the value of continuous response variable given the value of predictor variables.

The analysis yields a predicted value for the criterion resulting from a linear combination of the predictors. The linear model for regression, the response variable y is defined as a linear functional form of input variables x. This is often simply known as *linear regression*. This model assumes that the variable of interest, y, is linearly related to an independent variable x. To describe the linear relationship the following equation is used [74, 117, 20]:

$$y = \alpha + \beta x + \epsilon, \tag{3.1}$$

where α is the intercept the value of y when $x = 0$; and β regression coefficient is the slope of the line, defined as the change in y for a one-unit change in x. This model describes a deterministic relationship between the variable of interest y, sometimes called the response variable, and the independent variable x, often called the predictor variable. That is, the linear equation determines an exact value of y when the value of x is given. The parameters α and β can be calculated using the principle of least squares method [117], finding the values of α and β that minimize the sum of squared residuals.

In order to determine the significance of the linear regression model $p-values$ are reported [117]. To measure the strength of the relationship between the response variable, y, and the predictor variable, x, the coefficient of determination, R^2, is used. The R^2 value measures how much of the response variable variation (around its mean) is explained by a linear model. It can take values from 0 to 1, where 0 means none of the response variable variation is explained by the linear model, whereas 1 means all of the response variable variation is explained by the linear model. Therefore, values close to 1 mean a better model fit and a better prediction.

The equation for calculation of R^2 is [117]:

$$(R^2) = \frac{\sum_i (\hat{y_i} - \bar{y_i})^2}{\sum_i (y_i - \bar{y_i})^2}, \tag{3.2}$$

where y_i represents the actual value of observation, while $\hat{y_i}$ represents the estimated value of observation, and $\bar{y_i}$ represents the mean of actual values of observations.

Fixed/Random Effects Regression.

Fixed/Random effects models are extensions of linear regression models for data that are collected and summarized in groups. These models describe the relationship between a response variable and independent variables, with coefficients that can vary with respect to one or more grouping variables. A model consists of fixed effects and/or random effects [62, 85, 175]. Fixed effect terms are usually the conventional linear regression part, and the random effects are associated with control individual groups drawn at random from a population [175].

Using fixed/random effect model, individual differences of groups can be modelled easily by assuming different intercepts for each group. That is, each group is assigned a different intercept value, and the model estimates these intercepts. The model also allows for different slopes where control variables are not only allowed to have differing intercepts, but where they are also allowed to have different slopes. The combination of fixed and random effect model is called mixed-effect model [205]. The standard form of a linear mixed-effects model is:

$$Y_{ij} = \alpha_j + \beta X_{ij} + b_i + \epsilon_{ij}, \tag{3.3}$$

$$\alpha_j = a + c + u_j, \tag{3.4}$$

where Y_{ij} is the response variable for observation i and group j; X_{ij} is the predictor (or independent) variables; α, β are the regression coefficients; and b is random effect.

To write the correlation between any two observations in the same group as:

$$\rho = cor(Y_{ij}, Y_{ij'}) = \sigma_a^2/(\sigma_a^2 + \sigma_e^2), \tag{3.5}$$

where σ_a^2 is a variation across groups (usually called between groups, even if there are more than two) and σ_a^2 is variation within groups.

The key part of this modelling approach is varying coefficients (intercept and slope). Varying-intercept model is the model in which the regressions have the same slope. Varying-intercept and varying-slope model is the model where intercepts and slopes both can vary [205].

The standard form of a varying intercept-varying slope model for groups is:

$$Y_i = \alpha_j + \beta_j X_i + \epsilon_i, \qquad i = 1, ..., n \tag{3.6}$$

$$a_j = a_0 + c_0 X_i + u_{j1}, \qquad j = 1, ..., J, \tag{3.7}$$

$$\beta_j = a_1 + c_1 X_i + u_{j2}, \tag{3.8}$$

where i are observations; j are groups; α and β are varying coefficients; a, c are individual group coefficients; and u_1, u_2 individual group errors.

Performance Evaluation

In order to asses performance of the prediction model, three different measures, Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Prediction at level k (PRED(k)) are used. They were applied to test dataset in machine learning experiment where the data was divided into training and testing. These performance measures are commonly used and suggested to validate prediction [16] and estimation models, especially in software engineering such as defect prediction and effort estimation [14, 16] and reliability estimation [186]. MAE is recommended by Shepperd and MacDonell [171] for being unbiased towards under- or overestimations. The RMSE gives a relatively high weight to large errors. Meaning that the RMSE may be more useful when large errors are particularly undesirable. RMSE is also a popular measure in the machine learning community. An alternative measure, PRED, is used. According to Kitchenham et al. [99], and Port & Korte [152], PRED has been more consistent and robust than other performance measures, since it is simply the percentage of instances in which the model produces an mean root error less than a pre-defined level k. Therefore, it is independent from variance of mean root errors and dataset size.

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It is the average over the test sample of the absolute differences between prediction and actual observations where all individual differences have equal weight. RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It is the square root of the average of squared differences between prediction and actual observation. Both MAE and RMSE express average model prediction error in units of the variable of interest.

Ideally MAE and RMSE should be as close to 0 as possible, whereas PRED(k) should be close to 1. As the error rate decreases, PRED increases. The value of k in the PRED calculation is usually selected as 25 or 30, meaning that the percentage of estimations whose error rates are lower than 25% or 30%. PRED might be a better assessment criterion for software practitioners, since it shows the variation of the prediction error, that is, what percentage of predictions achieve a level of error that was set by practitioners. Hence, it implicitly presents both the error rate (k) and the variation of this error among all predictions made by the analysts. PRED(25) was used to asses the performance of prediction, as recommended in the context of prediction in software engineering. It was also observed that there is not an optimal value for performance measures that fits best for all predictions studies. PRED(25) means "the model should predict at least 75% of data with 25% error or less" [19].

Equations for calculating MAE, RMSE and PRED(k) are:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|, \tag{3.9}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}, \tag{3.10}$$

$$Pred(k) = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i| / y_i \quad if \quad |y_i - \hat{y}_i| / y_i \leq k, \tag{3.11}$$

where y_i represents the actual value of observation, while $\hat{y}_i$ represents the estimated value of observation.

Prediction model acceptance criteria depend on study context and there is no absolute criterion for a good value of RMSE or MAE. In software engineering reliability or cost prediction model studies, it is suggested that a model is good if PRED(25) value is greater than or equal to 0.75 [41, 92, 57]. Conte et al. [41] and Kitchenham [92] suggested error performance of prediction model is smaller or equal to 0.25. PRED(25) $\geq$ 0.75 and RMSE $\leq$ 0.25 composite objective criteria offer good means of verifying a reliability and cost estimation models in software engineering [92]. However, in energy consumption prediction context, there is no criteria has been suggested. Since, there is no suggested criteria in software energy prediction context, only for this study purpose, PRED(25) $\geq$ 0.75 is set for high performance model, 0.50 $\leq$ PRED(25) $<$ 0.75 for medium performance and PRED(25) $<$ 0.50 low performance. Similarly, RMSE (MAE) $\leq$ 0.25 for high accuracy, 0.25 $<$ RMSE (MAE) $\leq$ 0.50 for medium accuracy and RMSE (MAE) $>$ 0.50 low accuracy.

RMSE was selected higher than MAE criterion, since RMSE has a tendency to be larger than MAE as the sample size increases.

3.3.2 Proposed Approach for Data Scarcity

As stated earlier, one of the problems faced by the software energy related studies: the challenge of making energy consumption prediction using data that is usually scarce. Early studies in software engineering suggested that a company needs its own dataset (single-company data) to produce more accurate estimates [91, 100]. However, as Kitchenham et al. [96] states three main problems may occur when relying on single-company data: (1) the time required to accumulate enough data on past projects from a single company may be prohibitive, (2) by the time the data set is large to be of use, technologies used by the company may have changed, and older projects may no longer be representative of current practices, and (3) care is necessary as data needs to be collected in a consistent manner.

Within-product data which are collected in the company from past product releases may predict the same product future releases or different product energy consumption behaviour successfully. Therefore aforementioned problems and cross-product prediction research (see in Chapter 2) efforts have further motivated to use various software product datasets for cross-product energy consumption prediction approach.

This approach, which was adopted from Mendes and Kitchenham [118] and Turhan et al. [189], was employed in two steps:

- *Step 1:* Within-product models were derived for each product datasets. The performance of prediction for these models were computed using RMSE, MAE and PRED(25) on the same data used to build models. This allowed us to compare the prediction accuracy for the cross-product model with the prediction accuracy for the within-product model.
- *Step 2:* All within-product models for each product was used to predict the values of the other products (cross-products). This allowed us to assess how good the within-product model predict another product data.

  Figure 3.3 shows the illustration of the cross-product analysis.

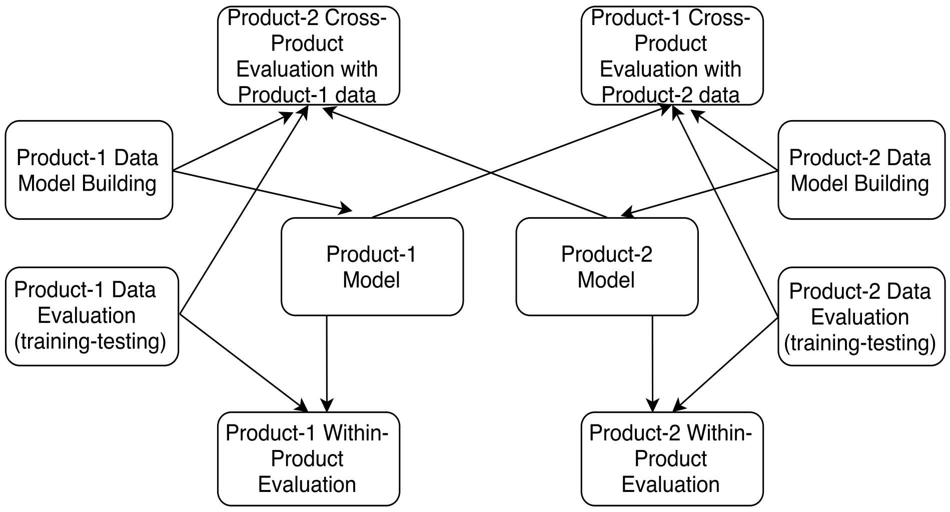

Product-2 Cross-Product Evaluation with Product-1 data
Product-1 Cross-Product Evaluation with Product-2 data
Product-1 Data Model Building
Product-2 Data Model Building
Product-1 Model
Product-2 Model
Product-1 Data Evaluation (training-testing)
Product-2 Data Evaluation (training-testing)
Product-1 Within-Product Evaluation
Product-2 Within-Product Evaluation

Figure 3.3: Illustration of the Cross-Product Analysis.

Chapter 4

Data Collection Methodology and Dataset Construction

This chapter presents the methodology for data construction from observational and mining software repositories and provides a descriptive analysis of the datasets.

As was stated in Chapter 3, for the purpose of providing empirical evidence on the relationship between software static code metrics and energy consumption, a retrospective cohort study was designed including observational data collection and mining software data repositories. Observational data collection consists of running benchmarks on a commercial database software product and then collecting energy consumption data of the product. Mining software data repositories consists of collecting data from data repositories and source code repositories, and extracting product, process and energy consumption data.

4.1 Dataset Construction

Dataset construction consists of two different techniques: (i) observational data collection and (ii) mining software repositories.

4.1.1 Variable Selection

In order to answer the RQ, it was necessary to specify the independent variables that characterise the study. While the independent variables are represented by software static code metrics, the dependent variable is Energy Consumption (EC) (Watt-Hour), measured as power (Watt) over time (hour). The static code metrics are described in detail in the remainder of this section. Measurement details are given in section 4.1.2.

Static Code Metrics: As was described in Chapter 2, many researchers use code attributes to guide software defect prediction. Recently, a few studies [83, 155, 129] have been conducted seeking out a relationship between software code attributes by means of size, complexity, object oriented metrics, and software energy consumption. However, the relationships between static software code metrics and software energy consumption have not been studied with regards to the prediction of software energy consumption. Thus, software static code metrics can be utilized as indicators of software energy consumption. Table 4.1 shows the static code metrics that were selected as the independent variables. These metrics were widely used, in part chosen, as they are easy to use [123]. Size, complexity, and churn attributes can be automatically and easily collected, even for very large systems.Therefore, in this study LOC, LOCC, and cyclomatic complexity metrics (CC and MCC) were used.

Table 4.1: Definitions of the metrics used in this study.

Metric Name	**Acronym**	**Description**
Total Lines of Code	LOC	Total number of lines of code without comments.
Code Churn	LOCC	Total number of lines of code changed between two versions of the same software, as a sum of the added and deleted lines of code.
Cyclomatic Complexity	CC	Measures the amount of decision logic in a single software module.
Modified Cyclomatic Complexity	MCC	Same as cyclomatic complexity, except that each case statement is not counted, entire select case block count as 1.

Total Lines of Code: The number of lines of code is a traditional measure for software size. Counting source lines of code is simple and reliable [57].

Code Churn: It includes collected, edited, added and deleted lines of code in all files of a release. The sum lines of edited, added, deleted lines of code were taken for each release. Churn metrics have been used by various researchers in software quality prediction [139, 138, 130]. It has been shown that churn metrics are significant predictors in software systems. Thus, one code churn metric has been included.

McCabe Cyclomatic Complexity: McCabe's cyclomatic complexity metric, as the code churn metric, is among the most popular predictors in software systems and it is widely used in prediction studies such as in Menzies et al. [122] and in Tosun [186]. It provides a quantitative basis to estimate the code complexity on the basis of the decision structure of a program. Cyclomatic complexity is based entirely on the structure of software's control flow graph. Control flow graphs describe the logic structure of software modules. Cyclomatic complexity of a module is $v(G) = e - n + 2$, where G is a program's flow graph, and e and n are the number of edges and nodes in the control flow graph, respectively. From a metrics perspective, total cyclomatic complexity and total Modified Cyclomatic Complexity (MCC) values of all code files were taken.

Modified Cyclomatic Complexity is a variant of cyclomatic complexity in which switch statements are considered to have the same effect on complexity as if statements, regardless of the number of switch cases [135, 57]. The motivation behind using the cyclomatic complexity metrics in this study is that the more structurally complex (and more loops) a code gets, the more difficult it becomes to test and maintain the code, hence, the likelihood of energy consumption increases.

4.1.2 Observational Data Collection and Analysis

The data collection and analyzing method employed here consists of five steps (see Figure Chapter 3 3.2) adopted from Hindle [83].

- Step 1: Choosing a product and a context,
- Step 2: Deciding upon measurements and instrumentation,
- Step 3: Configuring the test beds and the tests,
- Step 4: Collecting and storing the data, and
- Step 5: Analysing the data.

Step 1: Software Under Study

The first step is to choose a product and the context in which the software product is going to be tested. The main software product under study is IBM DB2 database system for Linux, UNIX and Windows. DB2 is a commonly used relational database management software that has been on the market since 1992 with a considerable market share. In 1983, DB2 for Multiple Virtual Storage version 1 was released and it has been constantly evolving ever since. DB2 was used to indicate a shift from hierarchical databases to the new relational databases such as the Information Management System. DB2 development continued on mainframe platforms as well as on distributed platforms [39]. Tests were run on the "DB2 Advanced Enterprise Server Edition" [36].

Based on availability, four major versions were chosen from among seven major versions of DB2. For a variety of samples, all minor versions from each major version were also chosen.

Table 4.2 shows the list of the DB2 major releases along with a list of the minor DB2 releases used in this study. The release time line of the major and minor releases is shown in Figure 4.1.

Table 4.2: List of four major DB2 releases with their corresponding minor versions used in this study.

Major Release	v.11.1	v.10.5	v.10.1	v.9.5
Minor releases	11.1	10.5.8 10.5.7 10.5.6 10.5.5 10.5.4 10.5.3a 10.5.3 10.5.2 10.5.1	10.1.5 10.1.4 10.1.3a 10.1.3 10.1.2 10.1.1	9.5.10 9.5.9 9.5.8 9.5.7 9.5.6a 9.5.5 9.5.4a 9.5.4 9.5.3b 9.5.3a 9.5.3 9.5.2a 9.5.2 9.5.1

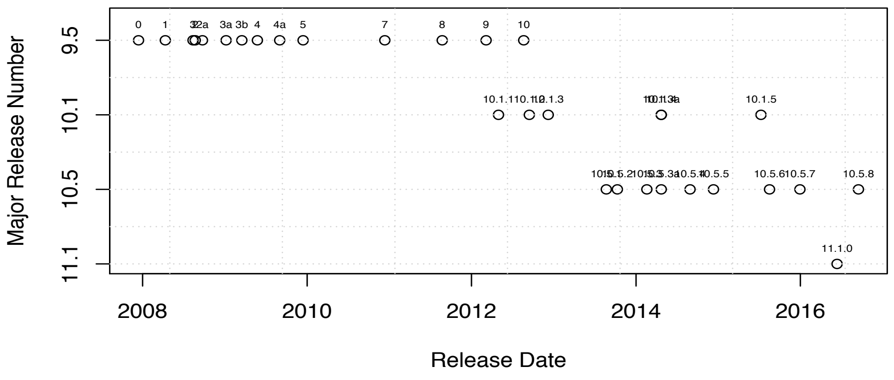


Figure 4.1: Time line of the DB2 releases. Y-axis depicts major release numbers; values above the points depicts minor release numbers. For example, number 1 in the top left corner represents the release date for version 9.5.1.

Step 2: Instrumentation

For the test system, a desktop computer operated by Windows 10 (64-bit) was used. The computer configuration is given in Table 4.3.

Table 4.3: Configuration of the test machine.

	Computer Configurations
Processor	Intel Core i7-6700 CPU @ 4.00GHz 4.01GHz
Memory	32 GB
Operating System	Windows 10 Home (64-bit)

The Meter: The energy measurement device used was “Watts up? PRO” [153]. The device allows direct reading of the measurements to a computer via a Universal Serial Bus (USB) port. It can measure both power (watt) and energy consumption (Wh). Its memory capacity is related to the number of parameters that are stored and the memory mode. When the device is in stop/overwrite mode, 32,000 measurements can be stored; when it is in automatic mode 1000 records can be stored. It was chosen to use the energy (with resolution of 0.1 Wh) measurement in automatic mode, as recommended by technical support [197] and in Miranskyy et al. [129]. Based on the recommendation and the meter manual [197], the accuracy of the power (Watt) measurement was ±98.5% (partially attributed to the shortest sampling interval of one second [196], measurement error ±1.5%). However, the cumulative energy measurement (cumulative Watt-Hours) was performed by the device continuously. The device was sampling the wattage measure 1000 per second and then integrating the data to obtain cumulative energy consumption, which resulted in higher accuracy. Cumulative energy readings were taken during the execution of a given workload [129].

Workload Configuration: Reference workload is an Online Analytical Processing (OLAP) type 5 from Transaction Processing Unit H (TPC-H) [44] benchmark. A benchmark is typically a computer program that performs a strictly defined sets of operations (workload) and returns some form of result describing how the tested system performed [115, 44]. TPC-H is a decision support benchmark and is considered the standard benchmark in the database systems. It consists of a set of 22 business-oriented ad-hoc queries. The data and queries simulate business practices and requirements. This benchmark mimics decision support systems that use large volumes of data, execute complicated queries (with relatively low volume of transactions), and answer business related questions. It has two utilities, namely, Data Base Generator Utility (DBGEN) and Query Generator Utility (QGEN), which generate the test data and queries of the TPC-H workload, respectively. DBGEN is used to generate data to be loaded into the database and QGEN to generate queries to manipulate the data. TPC-H examines a minimum of 1 Gigbayte (GB) of data, executes queries with a high degree of complexity, and gives answers to critical business solutions. The term “workload” is used to refer to the set of all queries used. Each query, based on the TPC-H design, performs one task. For example, one query produces a business report. Detailed technical analysis of the queries is given in Boncz et al. [23] and TPC-H [44]. Workload and queries were applied to each major and minor version.

Database Load Testing and Benchmarking: For benchmarking purposes, the HammerDB tool was used. HammerDB is a graphical open source database load testing and benchmarking tool for Linux and Windows to test databases running on any operating system. It is automated, multi-threaded and extensible with dynamic scripting support. It supports Oracle, SQL Server, DB2, TimesTen, MySQL, MariaDB, PostgreSQL, Greenplum, Postgres Plus Advanced Server, Redis Amazon Aurora and Redshift and Trafodion SQL on Hadoop. The tool includes complete built-in workloads based on industry standard TPC-C and TPC-H benchmarks [70]. The tool has two built-in scripts. One for creating a test schema with configurable data size. The other, for running the workload test with a benchmark selection.

Before starting the tests in this study, test schema were built with 1 GB of raw data and with one virtual user. In the actual test, a total of 22 distinct queries (from TPC-H) associated with 1 GB raw data were executed. In each run, the 22 queries were executed sequentially and measured the amount of energy consumed by the computer and time to finish all 22 queries. The workloads were executed ten times and the average of the results were used in further analysis. Details are given in section 4.1.2.

Software Static Code Metrics Extraction: For DB2 versions, CLOC tool [40] was used to extract size (LOC) and churn (LOCC) metrics. For McCabe complexity (CC and MCC), “PMCCABE” tool [151] for C and C++ were used. Both tools are open software. PMCCABE calculates McCabe cyclomatic complexity for C and C++ source code. PMCCABE includes a non-commented line counter, decomment, which only removes comments from source code; and codechanges, a program used to calculate the amount of change which has occurred between two source trees or files [151].

Step 3: Testbed Setup and Test Scenario

The computers used for the test system, as well as the data collection and analysis were identical to each other (see Table 4.3). The testbed set up is shown in Figure 4.2.

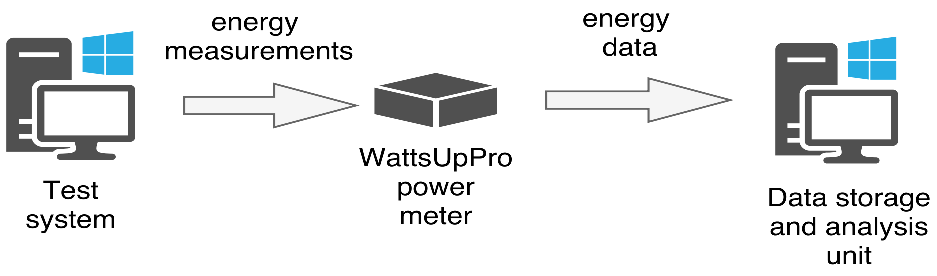


Figure 4.2: Testbed setup used in the DB2 energy data collection.

In order to eliminate the effect of changing environmental conditions, the testbed was setup in the Data Science Laboratory, Ryerson University. The system was dedicated to the test workload and all other background and computational tasks and the network connection were turned off. No other tasks were executed on the test machine concurrently.

As a reference database workload, TPC-H included 22 business oriented ad-hoc queries. For each run, an empty database was generated, which created the required objects. Then, the database was populated with 1 GB of raw data given the size of the hardware platform. All the data and objects were generated before running the queries. As such, the time and energy required for the generation was not included in the measurements. Details of the measurement and data collection are provided in the section 4.1.2 below.

Step 4: Measurement and Data Collection

Energy Consumption: As was previously stated, energy consumption measurements were obtained from WattsUp?Pro power meter instrument. In order to promote stability and consistency of the measurements, the same steps were followed in all of the measurements. Before the tests, each DB2 version was installed and configured as stated in the DB2 manual [114]. First, an empty database was created. Then DB2 test schema were built using HammerDB. Test schema, which included a number of users and data size selections. One user with 1 GB data was selected for each test. After the test schema with 1GB data creation was complete, the actual energy measurement tests were started. The energy measurement steps were adopted from [154] and modified according to the study environment. These steps used were:

- Plug the computer into the WattsUp?Pro device, turn on the computer and start to collect energy measurements;
- Close all unnecessary applications;
- Run the HammerDB and configure the database for the TPC-H test;

- Run TPC-H simultaneously and wait for all the 22 queries to finish. Record query start and end times of the queries;
- Collect the energy consumption data and time to finish all queries;
- Return the computer to its initial state.

Each minor and major versions were run ten times. The energy consumption reading was recorded at the beginning and at the end of all 22 queries. The readings were reported in Watt-hours. The average of ten runs per version was calculated to minimize the error. Start and finish time of all queries were also recorded.

Baseline Measurement: In order to obtain reliable measurements from all of the tests, it was also necessary to measure the idle energy consumption of the hardware and tools used. As previously mentioned, no other workloads were executed when the tests were executed. However, despite trying to minimize the number of externals run by the Operating System (OS), some of them have to be functional to ensure the robustness of the operating system. Additionally, idle database system and HammerDB may consume some amount of energy. In order to determine the impact of such processes on the energy consumption of system, energy consumption of idle system and HammerDB tool were measured and reported. Five runs were utilized, each for approximately one hour. Ten minute intervals were obtained to calculate mean power (watt) and mean energy consumption (Wh). Three different measurements were obtained:

- Running the OS with no application;
- Running the OS with HammerDB installed (no connections to the database were made);
- Running the OS with HammerDB and DB2 v.11.1 installed (no connections to the database were made).

The results are presented in Table 4.4. It can be seen that the OS alone and the OS with the HammerDB tool consumed almost the same amount of mean power in idle state. DB2 consumes a smaller amount of mean power (0.2W) on top of the power consumed by the OS and the HammetDB. However, the consumption remains almost constant throughout the tests. Therefore, it can be said that the baseline consumption does not affect the analysis.

Table 4.4: Baseline system average power (watt) and energy consumption (Wh) measurements.

Baseline Setup	**Average Power (Watts), Standard deviation**	**Average Energy Consumption (Wh), Standard deviation**
OS alone	36.117±0.425	6.058±0.072
OS with Hammer DB	36.183±0.356	6.043±0.067
OS with HammerDB and DB2 v.11.1	36.208±0.516	6.042±0.102

Software Metrics Extraction

The metrics, which were explained in section 4.1.1, of the software under study were extracted by downloading the source codes from their respected repositories. Then, the each of the source code files were uploaded into the code analysis tools to extract the metrics. As suggested by Miranskyy et al. [129], source code files of test cases were eliminated before computing the source code metrics. Source code files of test cases were eliminated, since the code from them is not included in the production binaries of the database.

Source code files of test cases and the production binaries of the database engines did not.

Step 5: Data Analysis Methodology

The analysis of the data was performed using R statistical analysis software (v.3.2.4)[1]. The following analysis and modelling techniques were applied to the data (most of them are described in [117, 132]).

Descriptive statistics consists of procedures used to summarize and describe the important characteristics of a set of measurements [117]. They are numerical descriptive measures that characterize a population. In this research, descriptive statistics characterize the sample which are used to infer population properties. The most common parameters are population minimum, maximum, mean, standard deviation, and variance.

Pearson's correlation is used to measure the strength of the linear relationship between two variables [117]. The correlation is represented with the *correlation coefficient* r ($-1 \leq r \leq +1$) and its significance. A value of r fairly close to 1 indicates a very strong linear relationship between two variables [117]. In order to determine whether a result is statistically significant, *p-value* is computed. *p-value* is the probability of obtaining a result equal to or more extreme than what was actually observed, when the null hypothesis is true [117]. In correlation, the null hypothesis is that there is no correlation between two variables. The *p-value* was adopted compared to the significance of the relation. In this study, 0.05 significance level was taken,

[1]http://www.r-project.org/

which is compared with the *p-value* ≤ 0.05. The correlation coefficient has to be properly interpreted. In order to interpret the correlation coefficient values with the significance of the relationship, several labelling systems [84, 180] have been proposed. Keep in mind that a correlation only indicates the presence or absence of a relationship, not the nature of the relationship; correlation does not imply causation. The strength of the correlation is described in Table 4.5, as per [84]. For example, $r = 0.76$ would be interpreted as a "strong positive correlation".

Table 4.5: Interpretation of the correlation coefficient.

Correlation Coefficient, r	**Interpretation**
.90 to 1.00 (-.90 to -1.00)	Very strong positive (negative) correlation
.70 to .90 (-.70 to -.90)	Strong positive (negative) correlation
.50 to .70 (-.50 to -.70)	Moderate positive (negative) correlation
.30 to .50 (-.30 to -.50)	Weak positive (negative) correlation
.10 to .30 (-.10 to -.50)	Very Weak positive (negative) correlation

4.2 Mining Software and Data Repositories

In this section four different open source software products were analyzed. The products are MySQL, Firefox, Vuze, and BitTorrent. Data sets were collected from their repositories using data mining suggested in Menzies et al.[122], Bener et al. [16], and Hindle [83]. The data collection procedure is shown in Figure 3.1. To collect raw data from source code repositories, automated scripts and queries were implemented (see Appendix A). Figure 4.3 shows a schematic of the data collection procedure. The source code of products were collected from their version control systems after that static code metrics were extracted using source codes. MySQL energy consumption and software metrics data were collected from data sources in Miranskyy et al. [128]; Firefox, Vuze and BitTorrent energy consumption data were collected from data sources in Hindle [83]. It was explained in Hindle [81, 83] that EC and time measurements were collected by running different test cases created by Hindle on Firefox, Vuze and BitTorent. There was no benchmark run performed.

The first software product to be tested was MySQL, a popular open source database software written in C and C++. The second product was Firefox, a popular C++ implemented consumer-oriented open-source web-browser maintained by the Mozilla foundation. The third product was Azeurus, now known as Vuze, a popular Java-based Peer-to-Peer (P2P) BitTorrent client. The fourth product was BitTorrent, which included library libTorrent and its client

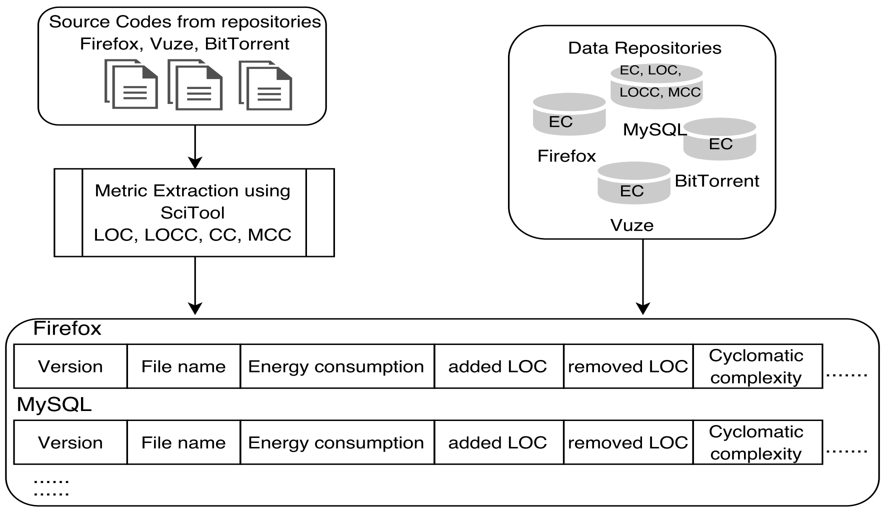


Figure 4.3: Metric data collection procedure illustration from available resources.

application, rTorrent. Much like Vuze, it is a BitTorrent client, but it is meant to run on UNIX shells.

MySQL energy consumption and software metrics data were extracted from the study conducted by Miranskyy et al. [129]. For Firefox, Vuze and rTorrent, software source codes were extracted from their respective repositories. Energy consumption and LOCC data were collected from Hindle's study [83][2].

Table 4.6 summarizes the datasets that were extracted from their repositories.

Table 4.6: A summary of the datasets extracted from repositories.

	MySQL	**Firefox**	**Vuze**	**rTorrent**
Binaries	40	509	45	18
Releases	40	43	3	18
Source	Subversion	Nightlies	Subversion	Tarballs
Language	C and C++	C++ and JavaScript	Java	C++
Size Metrics	LOC	LOC	LOC	LOC
Churn Metrics	LOCC	LOCC	LOCC	LOCC
Energy Consumption	Wh	Wh	Wh	Wh
Time Spent	hour	hour	hour	hour

[2]Available at: https://github.com/abramhindle/green-data-msr/tree/master/green-mining/data

In Table 4.6, for Firefox, “Nightlies” are evening compilations from their version control system between 2009 and 2010. “Tarballs” are tar.gz archives of source code, “Releases” are publicly named versions such as 3.0b1 and 5.1.1, while “Binaries” are successful builds [83].

The versions of MySQL have two different table storage engines options, namely MyISAM and InnoDB. InnoDB offers ACID-compliant (atomicity, consistency, isolation and durability) transaction features, which is important for confirming data consistency and integrity in the database. Prior to v.5.5, MyISAM was used as a default storage engine, which is better suited for analytical workloads [137, 136]. When extracting the source code metrics of MySQL, source code distribution of MyQSL, MyISAM and InnoDB directories were also reviewed. It was determined that there are 7 major directories in the MySQL main source code. One of them is allocated for storage engines, the MyISAM directory. In the main MyISAM directory, there are programs for handling files, rows, and keys. The programs in the other storage engine directories essentially fulfill the same functions [137, 136].

In the data repository of MYSQL ([129]), different combinations of storage engines (MyISAM and InnoDB), data size (1 GB and 3 GB), and energy consumption and time spent measurements are available. In order to be consistent with DB2 dataset measurements, 1 GB data size, energy consumption, and time spent measurements were selected from MyISAM and InnoDB data. For static code metrics, MyISAM and InnoDB engine files that belong to each engine directory were combined separately.

Software Metrics Extraction:

The metrics were extracted from their respected source codes. First, all source codes were downloaded from their repositories. Second, each of the source code files were uploaded into the code analysis tools. Last, code metrics were extracted using code analysis tools. “Understand” Static Code Analysis Tool v.4.0 (build number 864) from SciTools [165] was used for Firefox, Vuze and rTorrent software metric extraction. “Understand” is a commercially licensed tool; however, a student licence was obtained for this study. It has broad support for many different programming languages. It can analyze a very large, complex software project. MySQL metrics were collected from Miranskyy et al. [129]. The details of the extraction method are given in section 4.1.2

Five different metrics were extracted from the version history and code base in the software products. These metrics were added LOC, deleted LOC, total LOC, CC, and MCC. Energy consumption and time spent measurement data were collected from their related study data repositories.

Total Lines of Code (LOC): Total lines of code in all files of a release were collected for all datasets and total lines of code of all files were computed for each release.

Code Churn (LOCC): Added LOC and deleted LOC in all files of a releases of all software products were collected. The total lines of code change (LOCC) for each release were computed from added LOC and deleted LOC (see Table 4.1).

Cyclomatic Complexity (CC) and Modified Cyclomatic Complexity (MCC): McCabe's cyclomatic complexity and modified complexity metrics for each file in a release were extracted as described in section 4.1.2. For the total complexity values of each version, complexity metrics of all files were summed. When extracting metrics three steps were followed. The first step was extracting all the product's versions source codes from their respective code repositories and storing them in a local machine. The second step was removing test files from all the source codes. The final step was collecting complexity metrics using the SciTool Understand metric extraction tool.

4.3 Descriptive Statistics

Before constructing a prediction model, descriptive statistics and relations among the attributes [74, 117] must be computed. Descriptive statistics are information about the characteristics of variables, central values, and variability. Relation analysis among variables identifies whether or not there are statistical relationships between variables. Correlations are the most commonly used analysis.

4.3.1 Descriptive Statistics of Observational Data: DB2

A set of statistics was derived from the collected data. For a single version of DB2, a total of ten runs were executed, each composed of around 3000 observations (one per second) of the energy consumption measurement. As for the calculation of statistics, the mean value of each run was calculated. The energy consumption measurement was computed as follows:

$$EC_{i,j} = EC_{i,j\ \text{(last reading)}} - EC_{i,j\ \text{(first reading)}}, \tag{4.1}$$

where $EC_{i,j}$ represents energy consumption for the i-th release of interest (major or minor) and j-th run. $EC_{i,j\ \text{first reading}}$ and $EC_{i,j\ \text{last reading}}$ represent the energy consumption readings from the WattsUp?Pro device when the test run started and finished, respectively. When the computer was turned on energy measurement started immediately. After the system reached its steady state, benchmark queries were run. Since energy measurements were taken continuously, the first reading needed to be subtracted from the last reading in order to be consistent. Then, the mean of 10 runs of each release were calculated according to Equation 4.2:

$$EC_i = (EC_{i,1} + EC_{i,2} + ... + EC_{i,10})/10, \tag{4.2}$$

where EC_i represents energy consumption for the i-th release of interest (major or minor). The same method was applied to all versions of DB2.

Moreover, the relative percentage of difference between each two release values was calculated using the Equation 4.3:

$$\frac{(new \quad release \quad value) - (old \quad release \quad value)}{(old \quad release \quad value)} \times 100. \tag{4.3}$$

Table 4.7 represents descriptive statistics about energy consumption (Wh) measurements and Table 4.8 represents descriptive statistics about time spent (hour) measurements collected from 32 DB2 releases.

The descriptive statistics tables report minimum (Min.), maximum (Max.), mean, median, Standard Deviation (SD), and variance. In datasets with large variations and possibility of outliers, reporting both median and mean is preferred since the median is less sensitive to extreme values or outliers [117].

Table 4.7 shows that the mean values of newer versions (v.10.1.0-v.11.1.0) are 17.6% smaller than the older versions (v.9.5.0.-v.9.5.10). Energy consumption decreases when a newer version is released. This can also be seen in Figure 4.4.

Figure 4.4 displays the results of 320 runs of 32 different distinct releases of DB2 from version 9.5.0 to version 11.1.0. Each version is a box-plot consisting of a set of 10 runs. Test runs were performed more than 10 times for each version because of random failures such as failure to connect to the WattsUp? Pro, old tests not shutting down, or an erroneous run. At the end, 10 successful runs were taken into account for each version.

Figure 4.4 shows that DB2 v.10.1 is relatively stable in terms of EC, and time spent, but it can fluctuate between versions and measurements. The difference in means between v.9.5 versions and v.10.1 versions is about 1.13 Wh. Earlier versions of DB2 (v.9.5) have higher mean EC, but also high fluctuations. This may result in poor general performance, since performance is expected to be similar throughout the versions. Figure 4.4 also shows that v.9.5 consumed 22% higher EC (5.84 Wh) than v.10.1.0 (4.75 Wh), 20% higher than v.10.5.1 (4.67 Wh), and 34% higher than v.11.1 (4.45 Wh). Regarding minor versions of v.9.5, most releases stay close to a mean 5.8 Wh and 6.0 Wh. Variance among v.10.1 and its minor versions are smaller than variances of the v.10.5 versions. Moreover, mean values of v.10.1 versions are approximately 3.5% lower than the mean values of the v.10.5 versions (4.90 Wh and 5.07 Wh respectively). The lowest EC value is obtained from v.11.1 with a value of 4.45 Wh, which is the latest release

Table 4.7: Energy consumption (Wh) statistics overview.

Version	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
v.9.5.0	5.50	6.00	5.84	5.90	0.164	0.027
v.9.5.1	5.90	6.40	6.04	6.00	0.151	0.023
v.9.5.2	5.70	6.40	5.99	6.00	0.191	0.036
v.9.5.2a	5.40	6.00	5.81	5.80	0.296	0.087
v.9.5.3	5.80	6.60	6.30	6.40	0.305	0.093
v.9.5.3a	5.80	6.10	5.89	5.90	0.099	0.009
v.9.5.3b	5.60	6.40	5.95	5.90	0.236	0.056
v.9.5.4	5.70	6.10	5.88	5.85	0.168	0.028
v.9.5.4a	5.80	6.50	6.10	6.11	0.202	0.041
v.9.5.5	5.70	6.30	6.03	6.10	0.200	0.040
v.9.5.6a	5.60	6.30	6.00	5.95	0.245	0.060
v.9.5.7	5.80	6.50	6.13	6.10	0.200	0.040
v.9.5.8	5.70	6.50	6.07	6.00	0.275	0.075
v.9.5.9	5.90	6.30	6.08	6.10	0.113	0.013
v.9.5.10	5.70	6.10	5.90	6.00	0.139	0.020
v.10.1.0	4.40	5.20	4.75	4.70	0.259	0.067
v.10.1.1	4.80	5.40	5.10	5.10	0.200	0.040
v.10.1.2	4.70	5.40	5.03	5.00	0.211	0.045
v.10.1.3	4.60	5.30	4.94	4.90	0.201	0.040
v.10.1.3a	4.60	5.10	4.83	4.85	0.141	0.020
v.10.1.4	4.70	5.30	4.93	4.95	0.170	0.029
v.10.1.5	4.40	5.10	4.69	4.60	0.246	0.061
v.10.5.1	4.40	5.10	4.67	4.70	0.211	0.045
v.10.5.2	4.60	5.40	5.04	5.00	0.259	0.067
v.10.5.3	4.90	5.40	5.13	5.15	0.194	0.038
v.10.5.3a	5.00	5.70	5.41	5.35	0.266	0.071
v.10.5.4	5.10	5.60	5.38	5.40	0.181	0.033
v.10.5.5	4.70	5.20	5.00	5.00	0.188	0.035
v.10.5.6	4.60	5.40	5.03	5.05	0.200	0.040
v.10.5.7	4.70	5.30	5.01	5.50	0.172	0.030
v.10.5.8	4.70	5.30	4.91	4.85	0.191	0.036
v.11.1.0	4.10	4.70	4.45	4.50	0.184	0.034

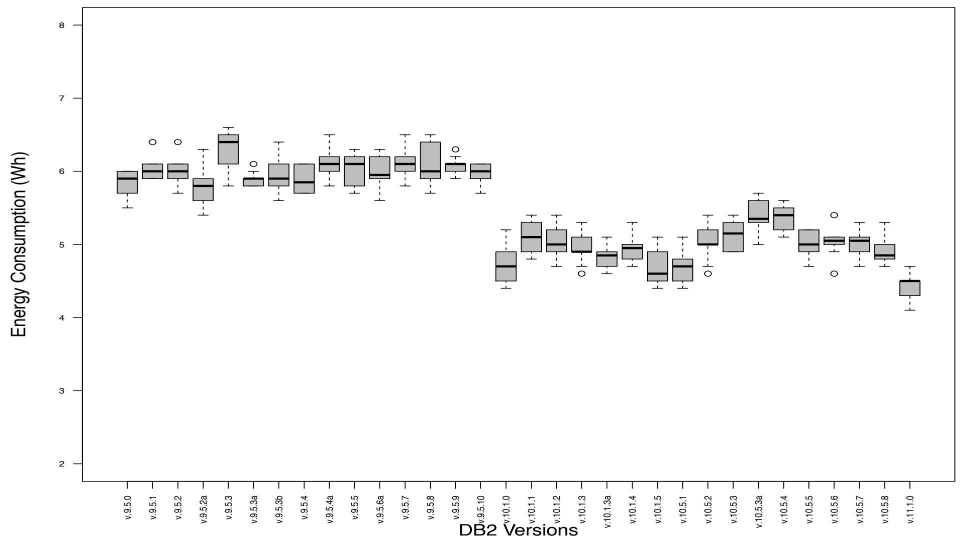


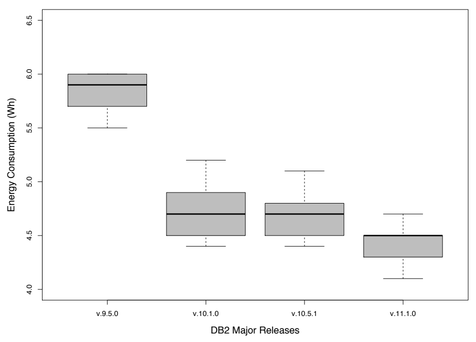
Figure 4.4: Energy consumption (Wh) box-plots of tested versions of DB2.

of the DB2.

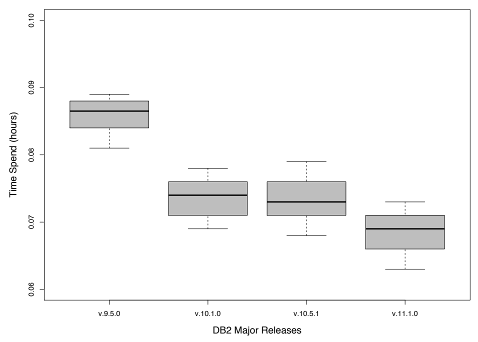
To summarize, DB2 consumed less energy and became more energy efficient over the releases. Energy consumption fluctuated in v.9.5 minor versions. On the other hand, it was more stable across v.10.1 versions. This may be an indication that DB2 is getting more efficient as the product matures.

By reviewing the DB2 functionality and feature changes in [114] it can be seen that after v.10.1 was released, most of the changes were implemented for performance enhancement. One of the major enhancements is for making the DB2 engine more efficient with new features. These changes may result in increasing the speed of queries and consuming fewer computational resources [114]. The biggest change is in the parallelization capabilities of the optimizer to enable workloads to better utilize multi-core processors [174]. Although CPU usage was not analyzed in this study, this may be the reason why energy consumption decreased. For example, a 19% decrease in EC was obtained from v.9.5.10 to v.10.1.

It was also noticed that when the product matures there are increases in EC within its major release of a version. For example, there was a 15% increase from v.10.5.1 to v.10.5.4. It was suspected that every new feature and functionality may lead to increases in energy consumption because of the cumulative effects of the software feature interactions. Thus, individual features may work as expected, but the modification of an existing feature or addition of new features



(a) Energy Consumption(Wh)



(b) Time Spent (hour)

Figure 4.5: DB2 major versions energy consumption (Wh) and time spent (hour) box-plots.

can trigger an unanticipated cumulative effect not anticipated in the original development plan.

Figure 4.5 displays the results of EC and time spent results for 40 runs of 4 major DB2 versions from v.9.5 to version 11.1. Each version is a box-plot consisting of a set of 10 successful runs. In the Figure 4.5, major version 10.5 did not have the 10.5.0 major release available to test. Therefore, v.10.5.1 was taken as a first version of v.10.5. As can be seen from Figure 4.5(a) the highest energy consumption value was obtained from v.9.5 with a value of 5.84 Wh, followed by v.10.1 and 10.5.1 with the values of 4.75 Wh and 4.67 Wh, respectively. The lowest energy consumption was obtained from v.11.1 with 4.45 Wh. The same trend was obtained from the time spent values, which are shown in the Figure 4.5(b)

In Table 4.8 and Figure 4.6 similar trends for energy consumption can be obtained. There are noticeable variations between the v.9.5 minor versions. On the other hand, time values are more stable across the v.10.1 (except 10.1.3a), especially in the later versions from v.10.1.5 to 10.5.3. This can be explained by the introduction of new features. When the feature matures, it further lowers the time spent (from v.10.5.5 to v.11.1). Overall, early versions (v.9.5) spend 19.9% more time to execute the test queries than the 10.1 versions, 5.9% more than the 10.5 versions, and 12% more than 11.1 version. This time behaviour suggests that query optimization to improve performance may lead to energy saving benefits.

Due to the availability of the DB2 source codes, 29 out of 32 releases were able to be analyzed for LOC, LOCC, CC and MCC. In Table 4.9 minimum, maximum, mean, standard deviation and variance of metrics collected from 29 DB2 releases are reported. Due to confidentiality of the IBM projects, actual metrics values of DB2 versions were not reported.

Table 4.8: Time spent (hour) statistics overview.

Versions	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
v.9.5.0	0.081	0.089	0.086	0.087	0.002	5.8E-6
v.9.5.1	0.088	0.093	0.090	0.089	0.002	2.8E-6
v.9.5.2	0.085	0.093	0.90	0.091	0.003	8.6E-6
v.9.5.2a	0.081	0.093	0.086	0.086	0.004	13.2E-6
v.9.5.3	0.087	0.097	0.093	0.093	0.003	11.6E-6
v.9.5.3a	0.083	0.088	0.086	0.086	0.002	2.5E-6
v.9.5.3b	0.083	0.092	0.087	0.087	0.003	8.0E-6
v.9.5.4	0.082	0.091	0.087	0.088	0.003	8.6E-6
v.9.5.4a	0.086	0.096	0.092	0.092	0.003	7.7E-6
v.9.5.5	0.087	0.094	0.090	0.091	0.002	6.9E-6
v.9.5.6a	0.086	0.096	0.091	0.091	0.003	11.7E-6
v.9.5.7	0.088	0.098	0.093	0.092	0.003	8.2E-6
v.9.5.8	0.083	0.094	0.087	0.088	0.002	11.8E-6
v.9.5.9	0.088	0.094	0.091	0.090	0.002	2.9E-6
v.9.5.10	0.087	0.091	0.090	0.091	0.001	1.8E-6
v.10.1.0	0.069	0.078	0.072	0.074	0.003	10.0E-6
v.10.1.1	0.073	0.08	0.076	0.077	0.002	5.6E-6
v.10.1.2	0.072	0.081	0.076	0.076	0.003	7.2E-6
v.10.1.3	0.069	0.079	0.075	0.074	0.003	8.5E-6
v.10.1.3a	0.067	0.073	0.069	0.069	0.002	3.6E-6
v.10.1.4	0.072	0.08	0.075	0.075	0.002	5.4E-6
v.10.1.5	0.065	0.077	0.072	0.073	0.007	13.8E-6
v.10.5.1	0.068	0.079	0.073	0.073	0.003	11.2E-6
v.10.5.2	0.067	0.076	0.073	0.072	0.004	13.2E-6
v.10.5.3	0.07	0.077	0.073	0.073	0.003	9.4E-6
v.10.5.3a	0.073	0.083	0.079	0.079	0.003	8.5E-6
v.10.5.4	0.073	0.079	0.077	0.077	0.002	4.1E-6
v.10.5.5	0.067	0.073	0.07	0.07	0.002	4.0E-6
v.10.5.6	0.065	0.071	0.070	0.070	0.002	3.3E-6
v.10.5.7	0.068	0.076	0.073	0.073	0.003	6.9E-6
v.10.5.8	0.069	0.078	0.072	0.071	0.003	7.6E-6
v.11.1	0.063	0.073	0.068	0.069	0.003	10.2E-6

Data Distribution Analysis: To visualize the fit of the distribution, Q-Q plots plots were examined and assessed for how closely the data points follow the fitted distribution line. Figure 4.7 (a-b) shows Q-Q plots, which are graphical techniques for assessing whether a data is normally distributed [37]. Figure 4.7 (a-b), the outer dotted lines form a 95% confidence

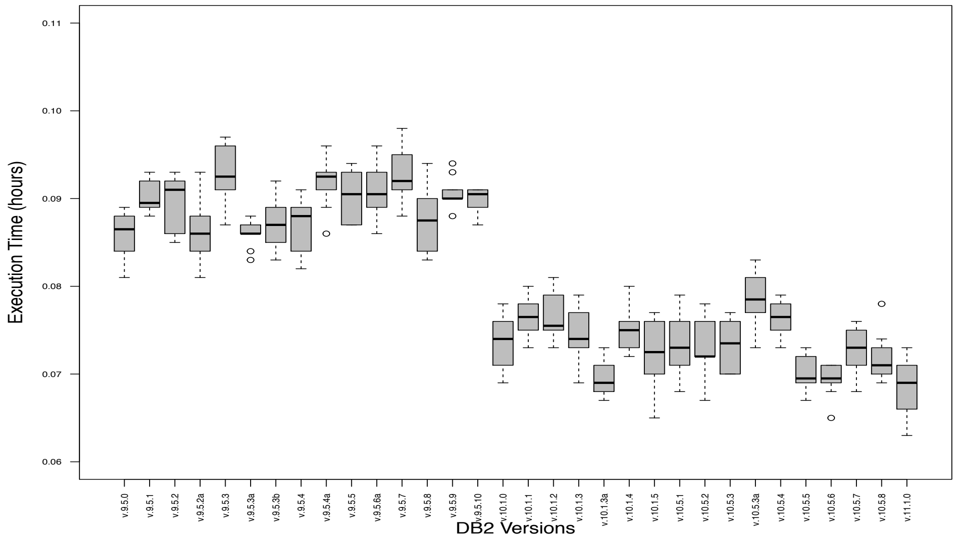


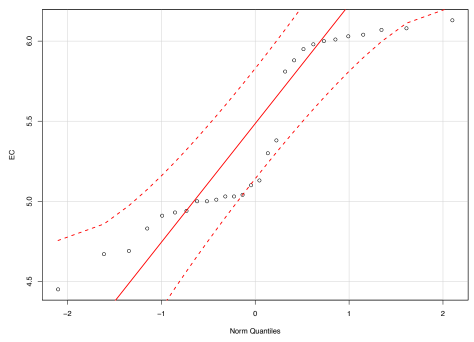
Figure 4.6: Time spent (hour) box-plots of tested versions of DB2.

Table 4.9: DB2: Descriptive statistics of software metrics, energy consumption, time spent, and software metrics.

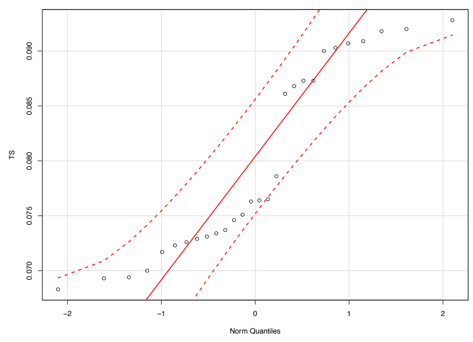
Metrics	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
Energy Consumption (Wh)	4.45	6.11	5.39	5.13	0.570	0.325
Time Spent (hour)	0.068	0.093	0.079	0.076	8.523E-03	7.264E-05

band. The use of the confidence band in the plot is that when values fall within the band, the distribution is approximately normal.

Figure 4.7 (a-b) and Figure 4.8 (a-b) show data distribution and density plots. Figure 4.7 (a-b) shows that all data values fall within the confidence band, meaning that the data follows approximately normal distribution. On the other hand, shape of the distribution has multiple peaks (see data density Figure 4.8 (a-b)). This means that distribution is multimodal. A multimodal distribution often represents a mixture of different populations in the dataset [117]. In DB2 dataset these different groups reflect v.9.5, v.10.1, v.10.5 and v.11.1. One common approach for analyzing this group data structure is to analyze groups separately. Therefore, DB2 versions were analyzed separately. The first group included major version 9.5 and its minor versions. The second group included version 10.1 and its minor versions. The third group included version 10.5 and its minor versions. In the following sub-paragraphs, groups of

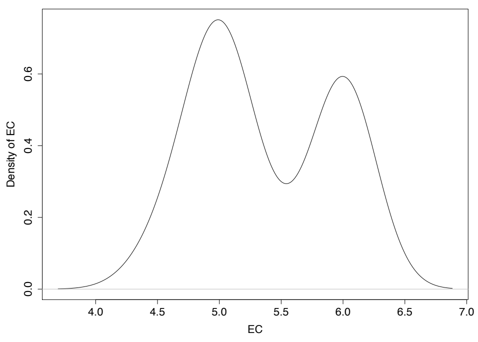


(a) Q-Q Plot of Energy Consumption (Wh) Distribution

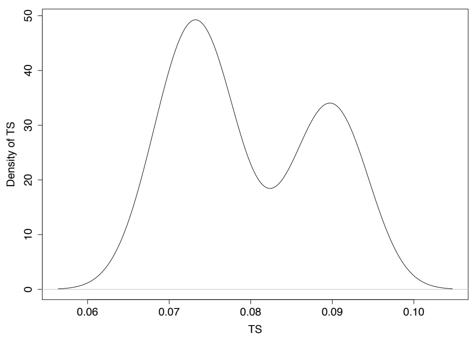


(b) Q-Q Plot of Time Spent (hour) Distribution

Figure 4.7: Q-Q Plots of energy consumption (Wh) and time spent (hour), the outer dotted lines 95% confidence band.



(a) Time Spent (hour)



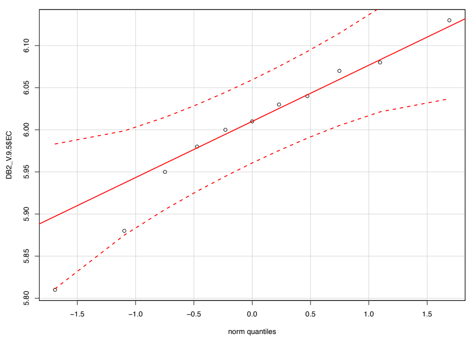
(b) Time Spent (hour)

Figure 4.8: Density graphs of energy consumption (Wh) and time spent (hour).

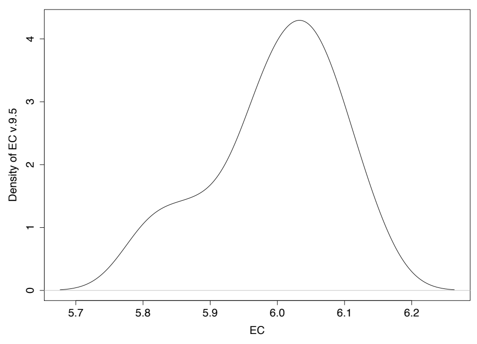
DB2 data distributions were given. Version 11.1 has only one data point; therefore, it was not included in the group analysis. But, it was included in the correlation analysis.

v.9.5 Data Distribution Analysis: Q-Q plot and density plot were examined and assessed for how closely the data points followed the fitted distribution line.

Figure 4.9 (a) shows that all values fall within the confidence band; therefore, the v.9.5 data is approximately normally distributed. The Figure 4.9 (b) density plot also shows that the EC of v.9.5 data distribution is approximately normal.



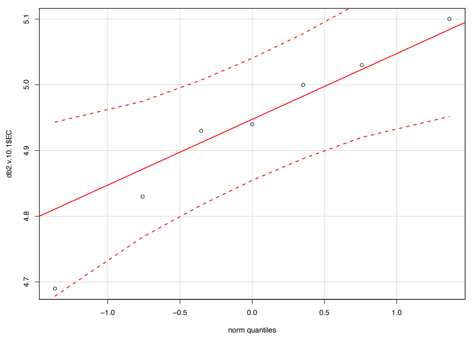
(a) Q-Q Plot of Energy Consumption(Wh) Distribution



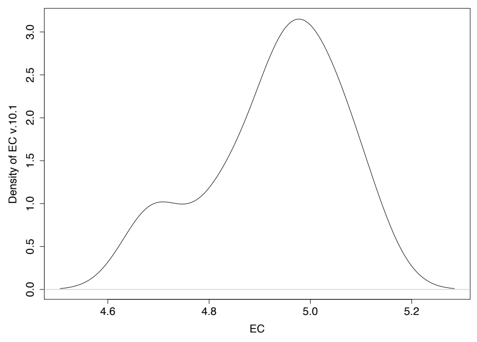
(b) Density Plot of Energy Consumption (Wh) Distribution

Figure 4.9: Q-Q Plot (a) and Density Plot (b) of energy consumption (Wh) of DB2 v.9.5. In (a) the outer dotted lines 95% confidence band

v.10.1 Data Distribution Analysis: Q-Q plot and density plot were examined and assessed for how closely the data points follow the fitted distribution line.



(a) Q-Q Plot of Energy Consumption (Wh) Distribution

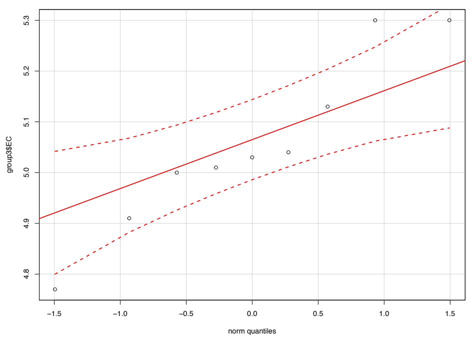


(b) Density Plot of Energy Consumption (Wh) Distribution

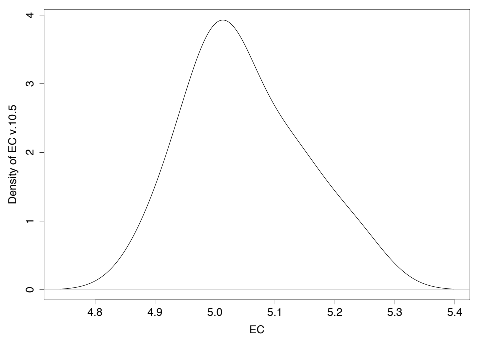
Figure 4.10: Q-Q Plot (a) and Density Plot (b) of energy consumption (Wh) of DB2 v.10.1. In (a) the outer dotted lines 95% confidence band.

Figure 4.10 (a) shows that all values fall within the confidence band; therefore, the v.10.1 data is approximately normally distributed. Figure 4.10 (b) density plot also shows that EC of version 10.1 data distribution is approximately normal.

v.10.5 Data Distribution Analysis: Figure 4.11 (a-b) show that EC of version 10.5 data is approximately normally distributed.



(a) Q-Q Plot of Energy Consumption (Wh) Distribution



(b) Density Plot of Energy Consumption (Wh) Distribution

Figure 4.11: Q-Q Plot (a) and Density Plot (b) of energy consumption (Wh) of DB2 v.10.5. In (a) the outer dotted lines 95% confidence band

4.3.2 Descriptive Statistics of Mined Datasets

Dataset MySQL:

For MySQL, there are two storage engine dataset statistics to be reported. Table 4.10 and Table 4.11 show MySQL data using the MyISAM storage engine. Table 4.12 and Table 4.13 show MySQL data using the InnoDB storage engine. Table 4.10 shows MyISAM source code metrics that were extracted from only the MyISAM directory files. Similarly, Table 4.12 shows InnoDB source code metrics that were extracted from only the InnoDB directory files. These files were extracted from the MYSQL main source code files and metrics were computed automatically using scripts.

Table 4.11 shows MySQL data using MyISAM storage engine. Code metrics were calculated by subtracting files that belong to InnoDB storage engine directories. In this case, files contain MyISAM files and other system files. Similarly, Table 4.13 shows MySQL data using the InnoDB storage engine. Code metrics were calculated by subtracting files that belong to MyISAM storage engine directories. In this case, files contained innoDB files and other system files.

In Table 4.10 minimum, maximum, mean, standard deviation and variance of metrics collected from 40 releases of MYSQL using MyISAM engine were reported.

By examining Table 4.10 and Table 4.11 mean values of energy consumption and time spent were shown to be similar to median values. On the other hand, mean values of code metrics were not close to median values, specially in LOCC. All of the code metrics show some skewness, and LOCC is the most severe. While LOC, LOCC and CC show skewness in a positive direction, MCC shows skewness in a negative direction.

Table 4.10: MYSQL - MYISAM engine directories only: Descriptive statistics of energy consumption, time spent and software metrics.

Metrics	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
Energy Consumption (Wh)	23.02	34.30	28.27	27.93	3.49	12.17
Time Spend (hour)	0.19	0.30	0.24	0.24	0.02	9.01E-04
LOC	33036	34765	33992	34006	381	1.45E+05
LOCC	0	67315	2416	34	10761	1.24E+08
CC	6490	6745	6592	6593	63	3970
MCC	6032	6274	6125	6104	63.2	3988

Table 4.11: MYSQL using MYISAM engine files and system files: Descriptive statistics of energy consumption, time spent and software metrics.

Metrics	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
Energy Consumption (Wh)	23.02	34.30	28.27	27.93	3.49	12.17
Time Spend (hour)	0.19	0.30	0.24	0.24	0.02	9.01E-04
LOC	981942	1586869	1251148	1211503	212190	4.5E+10
LOCC	595	1180376	6392	69935	212536	4.47E+10
CC	118043	162696	139631	137739	17867	3.24E+08
MCC	108324	148101	127739	125565	16056	2.53E+08

Table 4.12 and Table 4.13 show minimum, maximum, mean, standard deviation and variance of metrics collected from MYSQL running on the InnoDB engine.

Table 4.12: MYSQL - InnoDB engine directories only: Descriptive statistics of energy consumption, time spent and software metrics.

Metrics	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
Energy Consumption (Wh)	14.85	25.22	19.21	17.96	2.85	8.13
Time Spend (hour)	0.13	0.22	0.16	0.15	0.02	6.11E-04
LOC	86283	205838	153736	119313	45155	2.04E+09
LOCC	8	265691	24716	1166	64097	4.11E+09
CC	10328	24867	17782	14035	6083	3.70E+07
MCC	9960	22665	16615	12994	5482	3.0E+07

Similar to MyISAM results, mean and median values of energy consumption, time spent

Table 4.13: MYSQL using InnoDB engine files and system files: Descriptive statistics of energy consumption, time spent and software metrics.

Metrics	Min.	Max.	Mean	Median	SD	Variance
Energy Consumption (Wh)	14.85	25.22	19.21	17.96	2.85	8.13
Time Spend (hour)	0.13	0.22	0.16	0.15	0.02	6.11E-04
LOC	1034991	1759305	1370332	1295908	231597	5.40E+10
LOCC	603	1301242	88068	7107	254310	6.51E+10
CC	122915	175800	150692	147393	18384	3.39E+08
MCC	112954	161082	138122	134957	16548	2.73E+08

and code metrics were shown to be similar, except LOCC. Mean of LOCC are higher than median values.

When comparing only MyISAM and InnoDB engine datasets, InnoDB software code metrics (Table 4.12) have higher variations than MyISAM software metrics (Table 4.10). When comparing without InnoDB engine file metrics (Table 4.11) and without MyISAM engine file metrics (Table 4.13) results, there is much variation in both data sets. This means a large portion of LOC is added/deleted in every InnoDB release. On the other hand, there are smaller number of code modifications done in every release of MyISAM when compared to InnoDB. InnoDB files are more complex than MyISAM files (Tables 4.10 and 4.10). When comparing energy consumption and time spent, the MyISAM engine spent more time running the benchmark queries, and therefore consumed more energy than InnoDB engine. Conversely, the InnoDB engine has more LOC, LOCC and more complex files than MyISAM, which may indicate that implementation of InnoDB into the system needs larger updates on the entire system rather than small modifications/ updates for specific features in the software.

Although a benchmark [44] was run on both systems and its queries were the same, using MyISAM engine MYSQL versions consumed on average 5 times more energy than DB2 versions. Similarly, MYSQL using the InnoDB engine consumed, on average, 5.7 times more energy than DB2 versions. Comparing complexity of source files indicates that DB2 files were, on average, 8 times more complex than MYSQL files. Comparing LOC with LOCC in DB2, a small portion of DB2 LOC is added/deleted on average across the releases and a large portion LOC was maintained in every release. Similarly, in MyISAM around 6% of LOC was added/deleted in every release. On the other hand, in InnoDB, a large portion, around 16%, of LOC was added/deleted in every release. This may suggest that implementing and maintaining the InnoDB storage engine required large updates on the entire system. On the other hand, MyISAM is maintained with small modifications in every release.

Dataset Firefox:

Firefox is a popular open-source web-browser developed and published by the Mozilla Foundation. The original dataset includes nightly builds provided by Mozilla on their FTP site with the mozilla-1.9.2 main branch of Firefox 3.6 [81]. Generation of tests, measurements and collection of data details are given in Hindle's studies [80, 81, 83]. Original dataset has nighlty builds for 2009-2010 with versions ranging from 2.0 to 3.6, focusing mostly on 3.6 compilations for the main Mozilla Firefox branch. In Hindle [80], it was mentioned that each nightly build was run 3 times and recorded. In this research analysis, the mean value of all 3 runs was computed and used.

Table 4.14 shows minimum, maximum, mean, standard deviation and variance of metrics collected from 366 nightly builds of Firefox. From the original data published by Hindle [79], only the churn metric (LOCC) was obtained for 366 nightly builds. For LOC, CC and MCC, all 366 nightlies' source codes were mined from the Firefox source code repository (Mozilla-Mercurial repositories) and loaded to the SciTool Understand metric extraction tool.

After reviewing the metric results, it was noticed that a large portion of LOC is maintained with small modifications done in every nightly. By looking at the LOCC, it was noticed that files are added/deleted every 4-6 weeks on average. This indicates that nightlies contain small modifications/updates rather than large updates on the entire system. And big changes are made every 4-6 weeks. Therefore, to avoid these small changes that may create a noise in the dataset and to capture changes better in both energy consumption and metrics, the sub-dataset was created. This new sub-dataset includes only 32 monthly builds including their energy consumption, time spent, and code metric values.

For further notation clarification, the Firefox dataset with 366 nightlies is called "Firefox-1" and Firefox dataset with only monthly builds is called "Firefox-2".

The Firefox-1 dataset shows small variations between energy consumption and time spent values across the versions compared to the DB2 and MYSQL datasets. It also shows very high variance in LOC, LOCC, CC and MCC. Mean and median values of the energy consumption and time spent values are very similar. Similarly, software metrics' mean values are similar to the median values, except LOCC. Mean of LOCC is higher that that of the median value of LOCC.

Firefox-1 dataset shows many outliers. As was previously stated, in order to avoid outliers Firefox-2 dataset was created as a sub-dataset. Descriptive statistics of the Firefox-2 dataset are presented in Table 4.15.

Table 4.14: Firefox-1: Descriptive statistics of energy consumption, time spent and software metrics, $N = 366$.

Metrics	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
Energy Consumption (Wh)	1.51	2.86	2.41	2.47	0.169	2.88E-02
Time Spent (hour)	0.063	0.105	0.099	0.103	0.0063	4.87E-05
LOC	4951849	5698068	5440730	5353679	226810	5.14E+10
LOCC	0	407527	28764	115	93297	8.70E+09
CC	812387	915776	867200	848662	37757	1.43E+09
MCC	773327	869416	803909	822626	36832	1.35E+09

In the Firefox-1, the nightlies introduce noise into the dataset and it was hard to capture energy consumption change associated with code metrics. Therefore, the Firefox-2 dataset were used for prediction modelling purposes.

Table 4.15: Firefox-2: Descriptive statistics of energy consumption, time spent and software metrics, $N = 32$.

Metrics	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
Energy Consumption (Wh)	2.19	2.87	2.49	2.48	0.116	3.49E-03
Time Spend (hour)	0.089	0.105	0.103	0.102	0.0038	1.47E-05
LOC	4951849	5697871	5242529	5114790	249562	6.23E+10
LOCC	0	812169	306828	320879	196340	3.85E+10
CC	812387	911772	844478	830511	30658	9.39E+08
MCC	773332	865630	802705	790136	29487	8.69E+08

Table 4.15 shows that Firefox-1 and Firefox-2 have similar variations in metrics except for EC. Firefox-2 dataset has less variation in EC than Firefox-1. LOCC values have higher variation in the Firefox-2 dataset, which may indicate that 4-6 weekly builds contain large updates on the entire system rather than small changes for specific feature in the software. Firefox introduced a new release plan in 2011, which indicated that a new release was to be introduced every 6 weeks instead of every year. However, the data from 2009 to 2010 showed short cycle Firefox builds that still include major enhancements similar to new release enhancements.

Dataset VUZE:

Vuze is a popular Java based open-source BitTorrent client [9]. It is an end-to-end software application for BitTorrent [178]. BitTorrent is a popular peer-to-peer file-sharing protocol often blamed for much IP infringement, but it is also an effective method of distribution for large legal files, such as the Ubuntu Linux CDs [83]. In the original data from Hindle [80, 81, 83], there are 45 subversion revisions starting from revision number 26730 on September 14, 2011 to revision number 26801 on December 15, 2011 (3 months); between those 2 revisions there are a total of 45 subversions of data available. From the original data, EC, time spent and LOCC metrics were obtained. For LOC, CC and MCC metrics extraction, Vuze versions source codes were mined from its code repository and loaded in to the metric extraction tool (Understand-SciTool).

Table 4.16 presents the descriptive statistics of the Vuze dataset. Descriptive statistics include minimum, maximum, mean, standard deviation and variance of metrics collected from 45 subversions of Vuze.

Table 4.16: VUZE: Descriptive statistics of energy consumption, time spent, and software metrics.

Metrics	Min.	Max.	Mean	Median	SD	Variance
Energy Consumption (Wh)	3.12	3.77	3.57	3.58	0.161	0.025
Time Spend (hour)	0.137	0.166	0.158	0.159	7.293E-03	5.318E-05
LOC	2021000	2026269	2023390	2024137	1992.2	3.968E+06
LOCC	0	544	46.48	10	97.265	9460.61
CC	270434	271146	270778	270861	248	61546
MCC	267635	268347	267979	268062	248	61546

This dataset shows a very small variation in EC, time spent and all the metrics except LOC. Moreover, mean and median values are similar across the releases.

Comparing all datasets, Vuze has less LOC than the others and does not show drastic changes in code (LOCC), which indicates that subversions contain small changes or updates not including major enhancements for specific features in the product. Moreover, total complexity of source files in release was less than that of DB2, Firefox and MYSQL.

Dataset rTorrent:

rTorrent is a BitTorrent client, it runs on UNIX shells. Each rTorrent version works with a dedicated library version, libTorrent. libTorrent is a feature, which is complete C++ Torrent implementation and focuses on efficiency and scalability. It runs on embedded devices and desktops. In the original data from Hindle [79] for rTorrent, 18 snapshot versions were tested between rTorrent version 0.3.0 (2005) and rTorrent 0.8.9 (2011), and libTorrent versions between 0.6.4 (2005) and 0.13.0 (2011). rTorrent is the front-end client that makes calls to libTorrent in order to download a torrent from a BitTorrent cloud of peers. rTorrent and libTorrent are generally developed together and not all versions of libTorrent are compatible with rTorrent and vice-versa. In the original publications by Hindle [80, 83, 79], there are 18 rTorrent version with their respected libTorrent versions. He explained that, 18 libTorrent were linked to different rTorrent versions and total 40 combinations of rTorrent-libTorrent pairs were built, tested and measured. Similar to Firefox and Vuze, EC, time spent and LOCC metrics were obtained from Hindle's original data [79]. For LOC, CC and MCC metrics extraction, rTorrent and libTorrent versions source codes were mined from their code repositories and loaded into the metric extraction tool (Understand-SciTool).

In the original data, it was noticed that there were some irregularities in measurements such as abnormal drop in energy consumption or time spent. These abnormalities were removed before the data was used in this study. Table 4.17 shows minimum, maximum, mean, standard deviation and variance of metrics collected from 40 combinations of rTorrent and libTorrent.

Table 4.17: Combination of rTorrent and libTorrent: Descriptive statistics of energy consumption, time spent, and software metrics.

Metrics	**Min.**	**Max.**	**Mean**	**Median**	**SD**	**Variance**
Energy Consumption (Wh)	3.60	3.82	2.37	3.73	0.30	8.834E-02
Time Spent (hour)	0.16	0.17	0.10	0.16	0.01	1.695E-04
LOC	62132	87268	72199.77	77857	8527.05	7.271E+07
LOCC	165	20410	7571.42	5861	5687.62	3.234E+07
CC	11024	15462	12809.06	13797	1477	2.183E+06
MCC	10665	14939	12406.03	13376	1434.47	2.057E+06

From Table 4.17 it can be seen that this dataset shows a small variation in EC and time spent. Moreover, mean and median values are similar across the releases. On the other hand, variations in software code metrics are high. Compared with the other analized products, the combination of rTorrent and libTorrent datasets have the smallest LOC and LOCC, and contained the least complex files, when compared to the other products. This small LOC and LOCC may indicate that releases do not contain large updates on the entire system.

4.4 Correlation Analysis

To investigate the relations, Pearson correlations were computed to check whether metric pairs had a statistical relation. Correlation significance (*p-value*) is given in each correlation table. Significance is indicated by "*" and strength of the correlations are interpreted according to Table 4.5. In this study, the significance level was set at 5% ($\alpha = 0.05$).

Hypotheses Formulation: Hypotheses were formulated for each system analyzed. If the *p-value* was less than or equal to a significance level (α), then it is reported that the results are statistically significant at level 0.05. The following hypotheses were tested to identify the relationship between energy consumption and code metrics.

Although Hindle's [83] study does not show correlation between LOC and energy consumption, and lines of code change and energy consumption, he pointed out high *p-values* results in his analysis. In order to make any statement about existence of relationships between LOC and CC, LOCC and EC, there needs more data and more study. Thus, hypotheses H1 and H2 were produced below.

Hypothesis 1a (Null): Energy Consumption has no significant relation with Lines of Code.

Hypothesis 1b (Alternative): Energy Consumption has a significant positive relation with Lines of Code.

Hypothesis 2a (Null): Energy Consumption has no significant relation with Lines of Code Change.

Hypothesis 2b (Alternative): Energy Consumption has a significant relation with Lines of Code Change.

Code complexity estimates complexity of execution paths. There may exist a relation between complexity of paths and energy consumption; on the other hand, there may exist a simple code path (e.g., a non-complex for-loop with a large number of iterations) consuming most of the computational resources and, in turn, energy. Therefore, hypotheses H3 and H4 were formulated below.

Hypothesis 3a (Null): Energy Consumption has no significant relation with McCabe Complexity.

Hypothesis 3b (Alternative): Energy Consumption has a significant relation with McCabe Complexity.

Hypothesis 4a (Null): Energy Consumption has no significant relation with McCabe Complexity.

Hypothesis 4b (Alternative): Energy Consumption has a significant relation with McCabe Modified Complexity.

For all datasets, the statistics associated with *p-values* are given in section 4.4.1.

4.4.1 Dataset: DB2 Correlation Analysis

Pearson correlation tests were run using R statistical software version 3.2.4 (2016-03-10). The reasons for using R are for its simplicity of language of implementation and its readily available functions that can be used within its library. The R Stats Package [146] was used to compute correlations.

Results of the correlation test for metric pairs are presented in Table 4.18 and a panel correlation matrix is given in Figure 4.12. Short abbreviations are mapped to metric explanations in Table 4.1. The interpretation is given in Table 4.5.

Table 4.18: DB2: Pearson's correlation coefficients, r, and significance (*p-values*) in brackets.

	TS	**LOC**	**LOCC**	**CC**	**MCC**
EC	0.960 (<2.2e-16)	-0.912 (5.547e-12)	-0.307 (0.112)	-0.911 (6.593e-12)	-0.909 (8.515e-12)
TS		-0.949 (3.616e-15)	-0.209 (0.284)	-0.948 (5.63e-15)	-0.947 (6.755e-15)
LOC			0.169 (0.388)	0.998 (<2.2e-16)	0.998 (<2.2e-16)
LOCC				0.165 (0.398)	0.164 (0.404)
CC					0.999 (<2.2e-16)

From Table 4.18, EC-TS, EC-LOC, EC-CC, EC-MCC, TS-LOC, TS-CC, TC-MCC, LOC-CC, LOC-MCC, CC-MCC pairs have very strong correlations with coefficients greater than zero and *p-value*<0.05. Correlations between metrics are also presented in Figure 4.12. The upper triangle region shows correlation coefficients with "*" indicating a significant *p-value*. Strong relations (*p-value*<0.05) were considered while forming linear regression-based models in Chapter 5 5. EC-TS is expected to have very strong correlations because, when time increases, energy consumption increases. This strong correlation was found in Miranskyy et al. [129]. CC-MCC is also expected to have very strong correlation since MCC is derived from CC. As a result of the correlation analysis of DB2, energy and time are governed mainly by LOC, CC and MCC. On the other hand, LOCC has no effect.

From Figure 4.4 and Table 4.12, it is seen that the observations are grouped into three clusters. This clustered data structure is derived from different versions of DB2 data. The first cluster contains v.9.5 and its minor version data. The second cluster contains v.10.1 and its

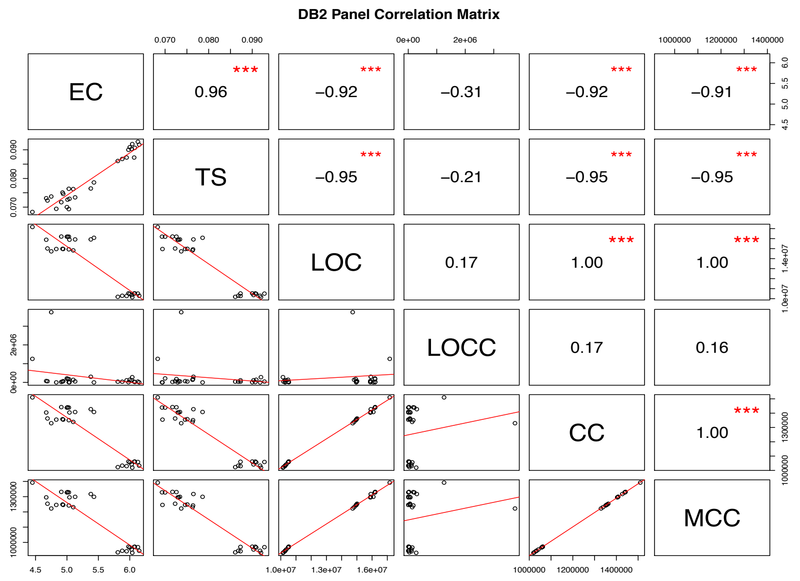


Figure 4.12: DB2 panel correlation matrix for energy consumption, time spent and software metrics. The lower triangle shows scatter plots and smoothed line. The upper triangle region shows the Pearson correlation coefficient and significance indicated by "*" symbols: ***0.0001, ** 0.001, *0.01, $^{\cdot}$0.05.

minor versions. The third cluster contains version 10.5, its minor versions and v.11.1. Cameron [29] suggested that failure to analyze within-cluster correlation can lead to misleadingly small standard errors, and consequent mislead to large correlation coefficients and low *p-values*. In order to avoid misleading correlation coefficients, within-cluster correlations were also analysed.

DB2 Group Correlation Analysis

DB2 Version 9.5 Correlation Analysis: Table 4.19 presents DB2 version 9.5 metric pairs correlations.

Table 4.19: DB2 version 9.5: Pearson's correlation coefficients, r, and significance (p-values) in brackets.

	TS	LOC	LOCC	CC	MCC
EC	0.76 (0.010)	0.60 (0.056)	0.13 (0.072)	0.61 (0.05)	0.61 (0.05)
TS		0.33 (0.34)	0.38 (0.28)	0.36 (0.31)	0.36 (0.31)
LOC			-0.56 (0.09)	0.99 (<2.2e-16)	0.99 (<2.2e-16)
LOCC				-0.55 (0.102)	-0.55 (0.102)
CC					0.99 (<2.2e-16)

Table 4.20: DB2 version 10.1: Pearson's correlation coefficients, r, and significance (p-values) in brackets.

	TS	LOC	LOCC	CC	MCC
EC	0.73 (0.06)	-0.67 (0.06)	0.25 (0.58)	-0.71 (0.072)	-0.71 (0.072)
TS		-0.27 (0.56)	-0.01 (0.97)	-0.29 (0.53)	-0.29 (0.53)
LOC			-0.82 (0.02)	0.99 (<2.2e-16)	0.99 (<2.2e-16)
LOCC				-0.78 (0.036)	-0.78 (0.036)
CC					0.99 (<2.2e-16)

From the Table 4.19, strong correlations with coefficients greater than 0.70 are EC-TS, CC-MCC and moderate correlations with coefficients greater than 0.50 are EC-LOC, EC-CC, EC-MCC, LOC-LOCC, LOC-CC, LOCC-MCC and LOC-MCC. Correlations between metric pairs EC-TS, EC-CC, EC-MCC are significant at 0.05 (p <0.05). Similar correlations pairs were obtained from v.9.5 and from all versions (Table 4.18). From all DB2 versions of data, EC-LOC, EC-CC, and EC-MCC pairs have very strong positive correlations. On the other hand, from version 9.5, these pairs have positive moderate correlations.

DB2 Version 10.1 Correlation Analysis: Table 4.20 presents DB2 version 10.1 metric pairs correlations. From Table 4.19, strong correlations with coefficients greater than 0.70 are EC-TS, EC-CC, EC-MCC, LOC-LOCC, LOC-CC, LOC-MCC, LOCC-CC, LOCC-MCC, and CC-MCC, and moderate correlations with coefficients greater than 0.50 are EC-LOC, EC-CC, EC-MCC. Similar to all DB2 versions of data, correlations (Table 4.18), from version 10.1 data, EC-LOC, EC-CC, and EC-MCC pairs have strong negative correlations.

DB2 Version 10.5 and 11.1 Correlation Analysis: Table 4.21 shows DB2 version 10.5 and version 11.1 metric pairs correlations.

Table 4.21: DB2 version 10.5 and 11.1: Pearson's correlation coefficients, r, and significance (*p-values*) in brackets.

	TS	**LOC**	**LOCC**	**CC**	**MCC**
EC	0.71 (0.02)	-0.67 (0.03)	-0.64 (0.04)	-0.63 (0.05)	-0.62 (0.05)
TS		-0.60 (0.06)	-0.45 (0.19)	-0.65 (0.04)	-0.66 (0.04)
LOC			0.89 (0.0005)	0.96 (<2.2e-16)	0.95 (<2.2e-16)
LOCC				0.88 (0.0008)	0.87 (0.001)
CC					0.99 (<2.2e-16)

From Table 4.21, strong correlations with coefficients greater than 0.70 are EC-TS, LOC-LOCC, LOC-CC, LOC-MCC, LOCC-CC, LOCC-MCC, and CC-MCC, and moderate correlations with coefficients greater than 0.50 are EC-LOC, EC-LOCC, EC-CC, EC-MCC, TS-LOC, TS-CC, and TS-MCC. Similar to all DB2 versions of data, correlations (Table 4.18), from version 10.5 and version 11.1 data, EC-LOC, EC-CC, and EC-MCC pairs have negative correlations.

4.4.2 Dataset: MYSQL Correlation Analysis

Table 4.22 and Table 4.23 present correlations between metrics of MYSQL using only MYISAM directories and only InnoDB directories, respectively. Moreover, Table 4.24 shows correlations between metric pairs, which are from all systems and only MyISAM directory files (which means without InnoDB directory files). Table 4.25 shows correlations between metric pairs, which are from all MYSQL system files and only InnoDB directory files (which means without MyISAM directory files).

Table 4.22: MYSQL-MyISAM only engine directories Pearson's correlation coefficients and significance (*p-value*) in brackets.

	TS	**LOC**	**LOCC**	**CC**	**MCC**
EC	0.999 (<2.2e-16)	-0.315 (0.046)	-0.159 (0.325)	0.232 (0.149)	0.120 (0.461)
TS		-0.323 (0.042)	-0.157 (0.333)	0.232 (0.149)	0.120 (0.459)
LOC			-0.085 (0.598)	0.552 (0.00021)	0.589 (6.376e-05)
LOCC				-0.336 (0.033)	-0.295 (0.063)
CC					0.986 (<2.2e-16)

Table 4.23: MYSQL-InnoDB only engine directories Pearson's correlation coefficients and significance (*p-value*) in brackets.

	TS	**LOC**	**LOCC**	**CC**	**MCC**
EC	0.998 (<2.2e-16)	0.250 (0.119)	0.057 (0.722)	0.178 (0.269)	0.173 (0.287)
TS		0.233 (0.147)	0.065 (0.688)	0.161 (0.320)	0.156 (0.336)
LOC			-0.107 (0.509)	0.992 (<2.2e-16)	0.994 (6.376e-05)
LOCC				-0.001 (0.9938)	-0.007 (0.968)
CC					0.998 (<2.2e-16)

Table 4.24: MYSQL-MyISAM and all system file directories Pearson's correlation coefficients and significance (*p-values*) in brackets.

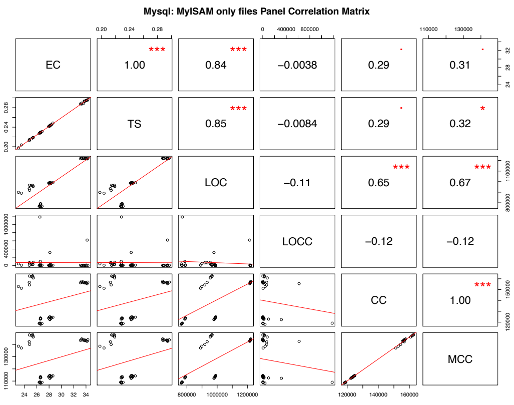
	TS	**LOC**	**LOCC**	**CC**	**MCC**
EC	0.999 (<2.2e-16)	0.843 (1.669e-11)	-0.003 (0.982)	0.287 (0.077)	0.314 (0.051)
TS		0.846 (1.107e-11)	-0.008 (0.959)	0.289 (0.074)	0.316 (0.049)
LOC			-0.112 (0.495)	0.651 (7.143e-06)	0.669 (3.083e-06)
LOCC				-0.116 (0.482)	-0.117 (0.479)
CC					0.999 (<2.2e-16)

Table 4.25: MYSQL-InnoDB and all system file directories Pearson's correlation coefficients and significance (*p-values*) in brackets.

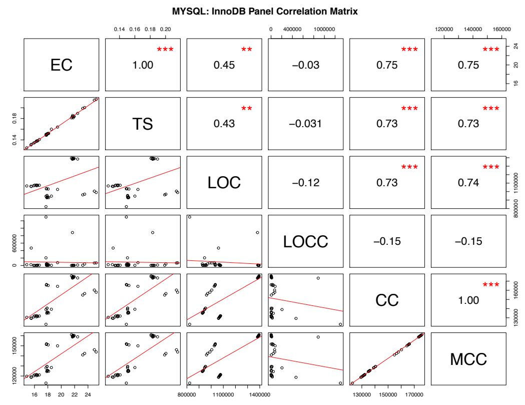
	TS	**LOC**	**LOCC**	**CC**	**MCC**
EC	0.998 (<2.2e-16)	0.451 (0.003)	-0.030 (0.855)	0.752 (3.387e-08)	0.748 (4.427e-08)
TS		0.428 (0.006)	-0.031 (0.851)	0.733 (1.103e-07)	0.729 (1.439e-07)
LOC			-0.118 (0.473)	0.732 (1.183e-07)	0.737 (8.379e-08)
LOCC				-0.145 (0.377)	-0.147 (0.372)
CC					0.999 (<2.2e-16)

When taking into account MyISAM directory files and all MYSQL system files, except InnoDB directory files (Table 4.24), very strong correlations were obtained between EC-TS, CC-MCC, strong correlations were obtained between EC-LOC, TS-LOC, and moderate correlations with coefficients greater than 0.50 were obtained between LOC-CC, LOC-MCC metric pairs. When taking into account InnoDB directory files and all system files, except MyISAM directory files (Table 4.25), similar to Table 4.24, very strong correlations were obtained in EC-TS and CC-MCC metric pairs. Additionally, strong correlations were obtained in EC-CC, EC-MCC, TS-CC, TS-MCC, LOC-CC, LOC-MCC, LOC-CC and LOC-MCC metric pairs. In contrast to Table 4.24, correlations were obtained between EC-LOC and TS-LOC at 0.05 level.

Figure 4.13 shows MYSQL panel correlation matrices for energy consumption, time spent and software metrics. These relations were considered when building the regression model.



(a) MYSQL without InnoDB engine directories panel correlation matrix



(b) MYSQL without MyISAM engine directories panel correlation matrix

Figure 4.13: MYSQL panel correlation matrix for energy consumption, time spent and software metrics. Diagonal shows metric name. Lower triangle shows scatter plots and smoothed line. Upper triangle region shows Pearson correlation coefficient and significance indicating with symbols: ***0.0001, ** 0.001, *0.01, ·0.05.

When considering only storage engine files, none of the correlations are strong enough to suggest a relationship except EC-TS and CC-MCC. EC-TS and CC-MCC are expected to have strong correlations because of their high dependencies. When considering each engine file together with all system files such as MyISAM engine files and all MYSQL system files, similar relations to the DB2 dataset were obtained. But, the difference is that in the DB2 dataset, EC-LOC, EC-CC, EC-MCC, TS-LOC, TS-CC and TS-MCC relations are on the negative side, but in the MYSQL dataset, these relations were on positive side.

4.4.3 Dataset: Firefox Correlation Analysis

Table 4.26 and Table 4.27 present correlations between metrics in the Firefox-1 and Firefox-2 datasets, respectively. When considering all nightly builds (Firefox-1), only EC-TS, LOC-CC, LOC-MCC and CC-MCC pairs have very strong correlations. When considering only monthly builds (Firefox-2), very strong correlation pairs are obtained as in dataset Firefox-1. Additionally, EC-LOC, EC-LOCC, EC-CC, EC-MCC, LOCC-CC and LOCC-MCC pair correlations are pronounced at 0.05 significance. But, these correlation coefficients are weak (correlations coefficients lie between 0.30-0.50).

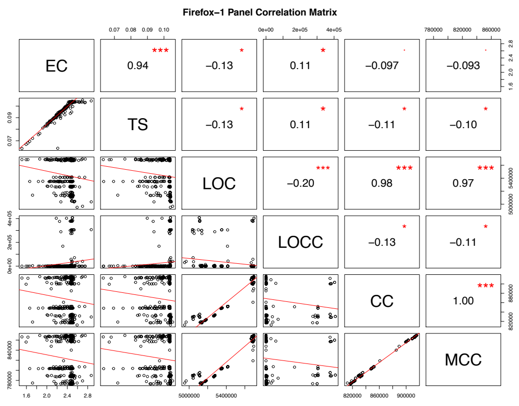
Figure 4.14 (a)-(b) shows Firefox-1 and Firefox-2 panel correlation matrices. It can be seen that when the Firefox data is less noisy, relationships are more pronounced. Therefore,

Table 4.26: Firefox-1 Pearson's correlation coefficients and significance (*p-values*) in brackets.

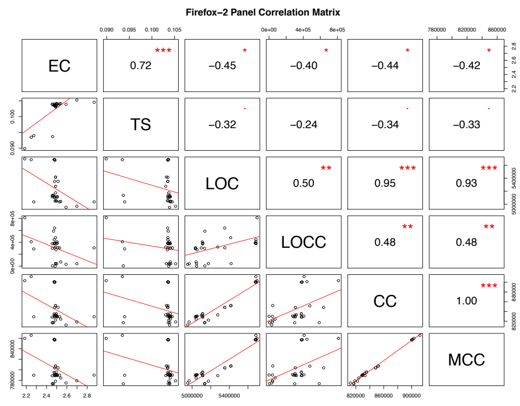
	TS	LOC	LOCC	CC	MCC
EC	0.941 (<2.2e-16)	-0.125 (0.016)	0.113 (0.031)	-0.096 (0.064)	-0.093 (0.076)
TS		-0.127 (0.015)	0.105 (0.044)	-0.106 (0.042)	-0.103 (0.049)
LOC			-0.198 (0.0001)	0.975 (<2.2e-16)	0.972 (<2.2e-16)
LOCC				-0.130 (0.013)	-0.127 (0.033)
CC					0.999 (<2.2e-16)

Table 4.27: Firefox-2 Pearson's correlation coefficients and significance (*p-values*) in brackets.

	TS	LOC	LOCC	CC	MCC
EC	0.718 (5.414e-06)	-0.449 (0.011)	-0.404 (0.024)	-0.435 (0.014)	-0.423 (0.017)
TS		-0.319 (0.080)	-0.243 (0.187)	-0.341 (0.061)	-0.331 (0.069)
LOC			0.503 (0.003)	0.954 (<2.2e-16)	0.934 (1.591e-14)
LOCC				0.484 (0.005)	0.478 (0.006)
CC					0.998 (<2.2e-16)



(a) Firefox-1 Panel Correlation Matrix



(b) Firefox-2 Panel Correlation Matrix

Figure 4.14: Firefox panel correlation matrix for energy consumption, time spent and software metrics. Lower triangle shows scatter plots and smoothed line. Upper triangle region shows Pearson correlation coefficient and significance indicating with symbols: ***0.0001, ** 0.001, *0.01, $\cdot$0.05.

Firefox-2 dataset helps to see the relations between metrics. This can also be seen in Figure 4.14 (a)-(b) scatterplots region. For this reason, only Firefox-2 dataset results were considered during model construction. In later sections,"Firefox" refers to the Firefox-2 dataset.

4.4.4 Dataset: VUZE Correlation Analysis

From Table 4.28, except between EC-TS, LOC-CC, LOC-MCC and CC-MCC (with the correlation coefficient being 0.9), none of the correlations are strong enough to suggest a relation. Scatter plots in Figure 4.15 show relationships between metric pairs. The Vuze dataset contains only subversion revisions. Descriptive statistics in Table 4.16 also show that unlike DB2, MYSQL, and Firefox datasets, the Vuze dataset does not have a large amount of LOC or code changes (LOCC) in between subversions. Therefore, in this dataset, it is hard to capture any relationship between EC and software code metrics.

Table 4.28: Vuze Pearson's correlation coefficients and significance (*p-values*) in brackets.

	TS	**LOC**	**LOCC**	**CC**	**MCC**
EC	0.997 (<2.2e-16)	-0.012 (0.935)	0.024 (0.876)	0.0005 (0.997)	0.0005 (0.997)
TS		-0.009 (0.949)	0.027 (0.858)	0.001 (0.994)	0.001 (0.994)
LOC			0.179 (0.237)	0.995 (<2.2e-16)	0.995 (1.591e-14)
LOCC				0.176 (0.247)	0.176 (0.247)
CC					0.999 (<2.2e-16)

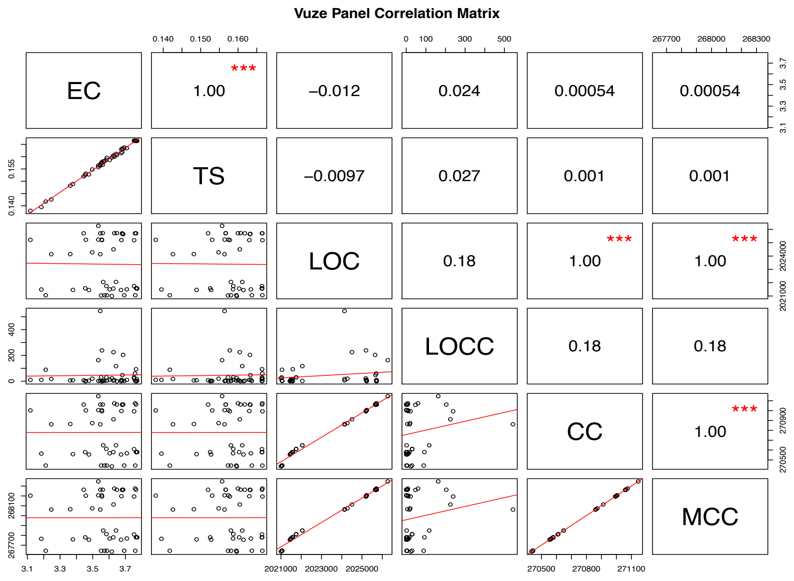


Figure 4.15: Vuze panel correlation matrix for energy consumption, time spent and software metrics. Lower triangle shows scatter plots and smoothed line. Upper triangle region shows Pearson correlation coefficient and significance indicating with symbols: $^{***}0$, ** 0.001, $^{*}0.01$, $\cdot 0.05$.

4.4.5 Dataset: rTorrent Correlation Analysis

The last dataset analyzed was rTorrent. The rTorent dataset was constructed as three datasets. The first dataset includes rTorrent versions metrics and the second dataset includes libTorrent version metrics. Table 4.29 presents correlations which were obtained from only rTorrent versions. Table 4.30 presents correlations obtained only libTorrent versions. As explained in section 4.3.2, rTorrent dataset was analized and abnormalities were removed. The third dataset was constructed from the combination of rTorrent version and its respected libTorrent version. Table 4.31 shows correlations obtained from the third dataset containing metrics calculated by summing rTorrent versions metrics and their respected libTorrent version metrics.

	TS	LOC	LOCC	CC	MCC
EC	0.999 (<2.2e-16)	0.154 (0.539)	0.522 (0.026)	0.204 (0.416)	0.216 (0.388)
TS		0.165 (0.512)	0.516 (0.028)	0.216 (0.389)	0.228 (0.362)
LOC			-0.088 (0.729)	0.978 (2.14e-12)	0.981 (7.76e-13)
LOCC				-0.079 (0.753)	-0.059 (0.815)
CC					0.998 (<2.2e-16)

Table 4.29: rTorrent Pearson's correlation coefficients and significance (*p-values*) in brackets.

	TS	LOC	LOCC	CC	MCC
EC	0.999 (<2.2e-16)	0.099 (0.693)	-0.006 (0.979)	0.069 (0.782)	0.070 (0.781)
TS		0.096 (0.703)	-0.009 (0.969)	0.067 (0.790)	0.068 (0.789)
LOC			-0.487 (0.040)	0.991 (9.591e-16)	0.991 (2.418e-15)
LOCC				-0.489 (0.039)	-0.491 (0.038)
CC					0.999 (<2.2e-16)

Table 4.30: libTorrent Pearson's correlation coefficients and significance (*p-values*) in brackets.

From Table 4.29 except EC-TS, LOC-CC, LOC-MCC and CC-MCC, the only correlated pairs are EC-LOCC and TS-LOCC with a moderate correlation coefficient of 0.52. From Table 4.30, except for the very strong correlated pairs EC-TS, LOC-CC, LOC-MCC and CC-MCC, there are no pairs that show significant correlation.

Table 4.31: Torrent (sum of rTorrent and libTorrent) Pearson's correlation coefficients and significance (*p-values*) in brackets.

	TS	LOC	LOCC	CC	MCC
EC	0.778 (3.93e-09)	0.417 (0.007)	-0.186 (0.249)	0.419 (0.007)	0.423 (0.007)
TS		0.263 (0.101)	-0.019 (0.904)	0.252 (0.116)	0.258 (0.107)
LOC			-0.168 (0.297)	0.998 (<2.2e-16)	0.998 (<2.2e-16)
LOCC				-0.173 (0.286)	-0.169 (0.295)
CC					0.999 (<2.2e-16)

From Table 4.31,in addition to very strongly correlated pairs, which are EC-TS, LOC-

CC, LOC-MCC and CC-MCC, new correlation pairs of EC-LOC, EC-CC and EC-MCC were obtained with a significance of 0.01 with correlation coefficients being 0.417, 0.419 and 0.423, respectively. These results are presented in the panel correlation matrix Figure 4.16.

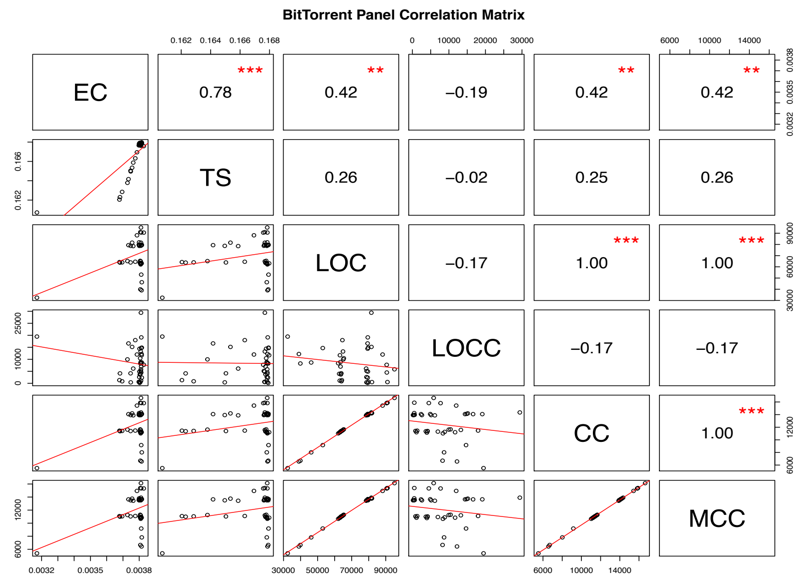


Figure 4.16: Torrent (sum of rTorrent and libTorrent) panel correlation matrix for energy consumption, time spent and software metrics. Diagonal shows metric name. Lower triangle shows scatter plots and smoothed line. Upper triangle region shows Pearson correlation coefficient and significance indicating with symbols: $^{***}0$, ** 0.001, $^{*}0.01$, $\cdot 0.05$.

In Chapter 5, model construction, the rTorrent dataset refers to the third dataset, which includes the sum of rTorrent and libTorrent metric values.

4.4.6 Hypothesis Testing and Results

Hypothesis 1: Except in the Vuze dataset, energy consumption has moderate to strong correlations with LOC at a significance level of 0.05 in all datasets. In datasets DB2 version 9.5, rTorrent, and MySQL, this relation is in positive direction. Thus, H_1b is accepted for DB2 version 9.5, rTorrent, and MySQL datasets. On the other hand, in dataset DB2 version 10.1, version 10.5, Firefox, correlation between EC and LOC is in a negative direction. Therefore, H_1b is rejected and H_1a is accepted for DB2 version 10.1, version 10.5, and Firefox.

EC improvement/deterioration seen in DB2 version 10.1, version 11.1 and Firefox datasets is probably due to local code optimization that do not alter EC, but the change in the EC is incidental. For example, addition of new features that change the LOC, may improve EC. In each major release, new features are added, which increase LOC, and some local code optimizations are made, which result in decreasing EC.

Hypothesis 2: Except in DB2 v.10.5 and the v.11.1 dataset, energy consumption shows no correlation with lines of code change at a significance level of 0.05 in all datasets. There is sufficient evidence at the 0.05 level to conclude that there is no significant relationship in between energy consumption and lines of code change. Therefore, H_2b is rejected and H_2a is accepted for all dataset.

Hypothesis 3: Except in the Vuze dataset, energy consumption has moderate to strong correlations with McCabe cyclomatic complexity at a significance level of 0.05 in all datasets. In datasets DB2 version 9.5, rTorrent, and MySQL the relation is in a positive direction. On the other hand, in dataset DB2 version 10.1, version 10.5, Firefox, correlation is in a negative direction. Thus, H_3b is accepted for all datasets except Vuze.

Hypothesis 4: Except in Vuze dataset, energy consumption has moderate to strong correlations with modified McCabe cyclomatic complexity at a significant level of 0.05 in all datasets. In datasets DB2 version 9.5, rTorrent, and MySQL the relation is in a positive direction. On the other hand, in dataset DB2 version 10.1, version 10.5, Firefox, correlation is in a negative direction. Thus, H_4b is accepted for all datasets except Vuze.

In the Firefox data, when outliers were removed and only monthly builds with large code changes (LOCC) were considered, EC-LOC and EC-LOCC pair correlations were pronounced at the 0.05 significance level. The results of Firefox reveals EC-LOC, EC-CC, EC-MCC, LOCC-CC and LOCC-MCC metric pairs correlations at 0.05 significance level. In the rTorrent data, when rTorrent versions and their respected libTorrent version metric values were summed, EC-LOC, EC-CC and EC-MCC correlations were obtained.

Therefore, when noise is removed, correlations are significant enough to suggest relations between EC-LOC, EC-CC and EC-MCC metric pairs.

Chapter 5

Proposed Prediction Models

5.1 Proposed Models

Based on the descriptive analysis of datasets in Chapter 4, fixed/random effects regression model were built for dataset DB2. The classical linear regression models were built for each dataset. The explanation for using linear regression model can be found in Chapter 3. All models were implemented in R. According to the correlation analysis results, one and/or multi-variable linear regression models are presented for each dataset in this chapter.

In the model construction, all software metrics were used as predictive variables to predict energy consumption (response variable). Since MCC is derived from CC, one of them was used in the model according to the correlation results with EC in Chapter 4.

5.1.1 Designing the Regression Model

Model assessment and performance comparisons are given in Chapter 3. The steps used in this study were adopted from Alpaydin [7], Bishop [20], and Montgomery [132].

General steps were adopted from Alpaydin [7]:

- *Aim:* to compare different one- and/or multi-variable regression models on five different datasets. Data collection and processing are explained in Chapter 4.
- *Selecting the response variable:* Response variable is energy consumption.
- *Choosing factors and levels:* The factors are input variables, standardization of datasets, and fixed learning algorithm. The input variables are software code metrics: LOC, LOCC, CC, and MCC. Standardization technique is z-score and algorithm is regression.

- *Choosing design:* Generally, given dataset, some part is used as the train set and the rest is used for testing. How this division is done is important. In practice, 70-30% or 60-40% are commonly used configurations. In this study, 70% is used for training and 30% is used for testing the algorithm. Replication is also important. Replication number depends on the dataset size. In this study, 100 replications of partitioning training-testing data were used.

- *Performing the prediction:* Predictions were conducted using R version 3.2.4, linear model toolbox from the R Stats Package [146].

- *Analyzing the results:* Performance assessments were done using a variety of measures given in Chapter 2.

In prediction models, a learner is targeted as having the highest generalization accuracy and minimal complexity (so that its implementation is cheap in time) and is robust. In this study the learner is a linear regression, details are given in Chapter 2. A schematic description of general learning components are illustrated in Figure 5.1. This process was done for each dataset. Learning system and data processing were replicated 100 times.

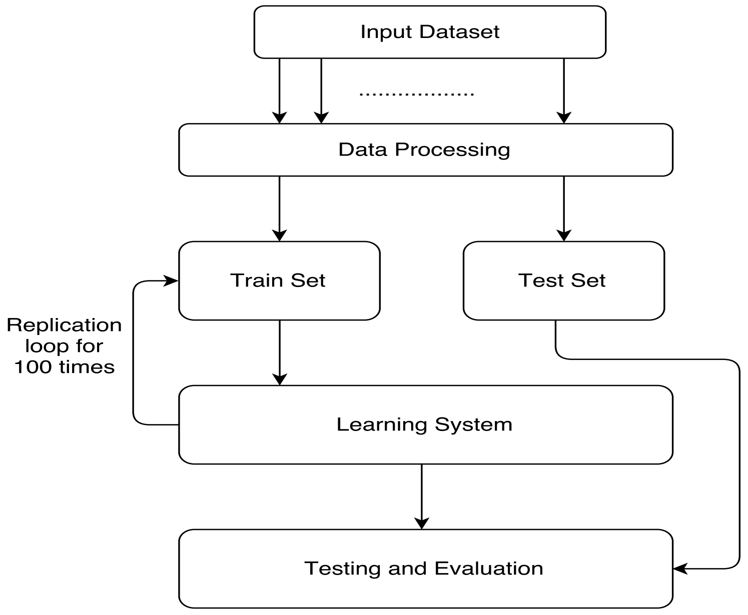


Figure 5.1: General Steps of Model Assessment and Performance Comparisons.

The model construction follows the R script-1 (see Appendix B) and R script-2 (see Appendix C). The model assessment and performance comparisons were conducted as follows:

- Step-1: Z-score standardization were applied for each dataset.
- Step-2: The dataset was divided into 70% training and 30% testing dataset.
- Step-3: Linear model was first trained on a training set, and tested on a testing set.
- Step-4: Performance measurements were computed and results were obtained. Then the process were repeated 100 times starting from Step-2 for different training and testing partition in the same dataset.

Checking for Regression Assumptions: Even though *p-values* and R^2 were used, the results of regression analysis must be checked to ensure they satisfy the necessary regression assumptions. The regression assumptions [117] are:

1. The parameters of the linear regression model must be numeric and linear.
2. The mean of the residuals must be close to 0.
3. For two-variable models, there must be no strong multicollinearity between the predictors.
4. The variance in the predictors must be larger than 0.
5. The residuals must be normally or approximately normally distributed.

The normal quantile-quantile plot (Q-Q plot) is the most commonly used and effective diagnostic tool for checking normality of the residuals. To support the graphical methods, the numerical methods and formal normality tests should be performed before making any conclusion about the normality of the data [159]. The Shapiro-Wilk normality test was used [169] to check the normality of the residuals in each model. The Shapiro-Wilk normality test is more effective and sensitive in small sample sizes ($n < 50$) [169, 159]. Therefore, it is suitable for all datasets in this study. The test result is given with W statistics and *p-value.* If the *p-value* is less than alpha level (0.05), then there is evidence that the distribution is not normal. On the contrary, if the *p-value* is greater than the alpha level, then error distribution is normal. The value of W statistic lies between zero and one. Small values of W lead to the rejection of normality, whereas a value of one indicates normality of the data [159].

All assumptions and Shapiro-Wilk results were checked for each dataset models and the residuals against both the predicted values and the explanatory variables were plotted. The assumption results are presented after the models.

5.1.2 Model Construction

According to the descriptive analysis results given in Chapter 4, linear fixed/random effects regression model were constructed for DB2 dataset, and single- and/or multi-variable regression models were constructed for DB2 version group datasets, MYSQL, Firefox, Vuze and rTorrent datasets. The models, their significance and their performance assessments are given in their related dataset subsections. Before performing prediction, z-score standardization were performed. The Standardization was performed before testing and training split on single data, and it is performed in each data set first and then union is taken before the prediction takes place. Z-score standardizes the variables, so that they are centered around 0 with a standard deviation of 1. It aims to re-express all variables similar in scale, therefore it may make it easier to compare regression coefficients. It is important when comparing measurements that have different scales, but it is also a general requirement for many learning algorithms. Since the input variables are on different scales, the standardization needs to be done before the prediction. With z-score standardization all the variables were rescaled so that they have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$ where μ is the mean (average) and σ is the standard deviation from the mean. Z-scores of the variables are calculated as follows [117]:

$$z\text{-score} = \frac{x - \mu}{\sigma}, \tag{5.1}$$

$$\beta' = \beta * \frac{\sigma_x}{\sigma_y}, \tag{5.2}$$

where σ_x standard deviation of x independent variable, and σ_y standard deviation of y dependent variable. In order to get standardized coefficients, replace every variable (y and all the x) in the regression with its standardized equivalent (from equation 5.2, subtract the mean off every observation, and divide that by the standard deviation), and then run the regression model. When standardized variables are used in regression, coefficients are standardized regression coefficients. Standard regression coefficients come from fitting the equation to standardized variables. Similarly, unstandardized regression coefficients come from fitting the equation to the unstandardized variables. It is not coefficients that get standardized, but variables [61].

Linear Regression with Single Predictor Variable.

Linear regression models were constructed using a single predictor variable for predicting the response variable. These models are formed based on the results of correlations given in Chapter 4. In Datasets DB2, MYSQL, and Firefox correlations were strong to moderate and relations were successfully defined. On the other hand, in the dataset Vuze and rTorrent, most of the pair correlations were moderate to weak. However, since their *p-values* are less than 0.05, they were considered when constructing models.

Single predictor linear regression is represented as the following model:

$$Y = \alpha + \beta X + \epsilon, \tag{5.3}$$

where Y is the response variable; X is the predictor (or independent) variable; α and β are the regression coefficients.

Linear Regression with Multiple Predictor Variable.

Linear regression models were constructed using two predictor variables for predicting the response variable. These models were formed based on the results of correlations given in Chapter 4. Two predictor variables were chosen for the model as independent variables affecting the final model. In order to avoid multicollinearity in three and more predictor variables and overfitting, linear regression model with two independent variables were proposed. Highly correlated variables were not feed into the same model.

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \epsilon, \tag{5.4}$$

where Y is the response variable; X_1 and X_2 are the predictor (or independent) variables; and α, β_1, and β_2 are the regression coefficients.

Linear Regression with Fixed/Random Effects.

Fixed/Random effects models are extensions of linear regression models. Using fixed/random effect model, individual differences of groups can be modelled easily. The importance of this model is varying coefficients (varying intercept and slope) [62].

The form of one random effect model is:

$$Y_{ij} = \alpha + \beta_j X_{ij} + b_i + \epsilon_{ij}, \tag{5.5}$$

where where Y is the response variable; X are the predictor (or independent) variables; β_j is effect of predictor variables in each dataset; and b_i (i=1, 2, 3) is effect of version (random).

In DB2 dataset, there is noticeable variation between versions (see Figure 4.4), and this variation should be accounted for in a model. In order to account for by-version variation, random effect was added to the model in overall DB2 regression model. This modelling approach can be considered a linear regression where intercepts, and possibly slopes, are allowed to vary by group and includes a categorical input variable representing group membership [62].

It is important to mention that all one- and multi-variable models and fixed/random effect models were run using standardized variables. The standardization was done by computing z-scores for each of dependent and independent variables. Therefore, the model coefficients (slope of regression lines), which were given in the datasets model result sections were standardized coefficients.

5.1.3 Models and Results for Dataset: DB2

Fixed/Random Effect Model for DB2 Dataset

In Chapter 4, section 4.3.1, it was observed that the DB2 data had a group structure where each version formed a group in the dataset. In grouped data structure, members of each group had common characteristics within groups [26, 15]. In DB2 data, each group consisted of major version and its corresponding minor versions. A common challenge in grouped data is within-group observation dependence [158, 175]. That is, members in the same group tend to be alike and share similar attitudes and behaviors relative to members from other groups. There may be unobserved group characteristics that affect the outcomes of the each group. This may result in biased estimates of parameter standard errors (e.g., regression coefficient), and possible substantive mistakes when interpreting the importance of one or another predictor variable.

One approach to modelling grouped data is using the fixed/random effect model approach [26]. Another approach is to analyze at the level of the individual group [75].

In order to consider group characteristics, fixed/random effect linear regression model is proposed for DB2 data. In this modeling approach, one or more effects are added to the model variables, which are considered as fixed and/or random effects. These effects essentially give structure to the error term ϵ and characterize individual group variation [26, 15]. This modelling approach has been widely used in data analysis of various domains [76, 4, 1, 109, 215, 58, 212, 195]. The advantage of the fixed/random effect modelling is that it provides much more flexibility and it take group-specific effect in which the model quantify regression effects for each group [201]. Unobservable group factors may affect the outcome of the regression. Fixed/random effect modelling approach has been recommended to use for clustered data analysis to consider unobservable group factor effect and it also provides a mitigation of omitted variable bias if there is a concern [205].

There are many definitions of fixed versus random effects [62]. Fixed effects are usually defined as varying coefficients that are not themselves modeled. The variable which is of interest is usually referred to as fixed effect; the variable which is a blocking factor/control variable (of interest) is random [62]. Gelman and Hill [62] suggested to always use random effects and focus on the description of the model itself (for example, varying intercepts and constant slopes), with the understanding that batches of coefficients will themselves be modeled.

The choice between fixed and random effects can be made based on the Hausman specification test [66, 46]. The test checks the differences between estimates. Random effects are preferred under the null hypothesis (*p-value* greater than α) due to higher efficiency of the estimators, while under the alternative (*p-value* smaller than α) fixed effects is at least consistent and thus preferred.

Based on Gelman and Hill [62], DB2 "version" was added as random effect. In the model, version was taken as a control variable and software metrics were taken as predictive variables. In the random effect model, each version was assigned a different intercept. Given that different intercept of each version group, the model had taken version variability into account.

An advantage of random effects is that time invariant variables can be included. Moreover, random effect models can be easily generalized to more than two groups [185]. This is an important consideration in DB2, since there are more than two version groups present in the dataset. In order to support random effect choice, the Hausman specification test was applied to one-variable and multi-variable models. The results are given in the Table 5.1.

Table 5.1: DB2 Hausman specification test results for choosing fixed or random effect.

Model	**Predictive Variables**	**chisq**	**df**	**p-value**
1	LOC	1.412	1	0.235
2	LOCC	0.649	1	0.420
3	CC	2.537	1	0.111
4	MCC	2.512	1	0.113
5	LOC, LOCC	5.797	2	0.055
6	LOCC, CC	1.607	2	0.448
7	LOCC, MCC	1.377	1	0.502

Table 5.1 shows that all models are not significant (*p-value* > 0.05). This means that "version" as a random effect is more suitable to model DB2 grouped data.

After determining the random effect model, model grouping factor were tested whether dataset fits random intercept or random intercept and slope. A random intercept only model and random intercept and slope models were constructed. To model comparison, analysis of variance (ANOVA) test were performed [201]. The varying intercept models were run and the estimates were saved, then random intercept and slope models were run and the estimates were saved, then the ANOVA test were performed.

Figure 5.2 shows Chi-Square values, the associated degrees of freedom (df) and the *p-values*. The *p-value* results of the test showed that the varying intercept and slope model does not fit the data any better than the varying intercept model. Although the test showed that there was no differences between the models, varying intercept and slope models were adopted to identify the individual version group differences.

Table 5.2: DB2 varying intercept or varying intercepts and slope model results.

Model	**Predictive Variables**	**Chi-Sq**	**df**	**p-value**
1	LOC	1.434	2	0.488
2	LOCC	0.853	2	0.653
3	CC	1.384	2	0.501
4	MCC	1.415	2	0.492
5	LOC, LOCC	1.648	5	0.895
6	LOCC, CC	1.11	5	0.953
7	LOCC, MCC	1.21	5	0.943

Models are presented in Table 5.3 with random effects for individual version groups. Model coefficients are shown in the intercept and slope (regression coefficient) columns. It is important to mention that these coefficients are standardized coefficients (see section 5.1.2). For the effect of each predictive variable, different intercept and different slopes were obtained. Model evaluations are given with *p-values* and R^2. Models#1-4 represent one predictive variable with random effect; Models#5-7 represent with two predictive variables. *P-values* give the significance of of each predictive variable in the effect of version.

Results in Table 5.3 show that the coefficients of the models by version are assigned different intercept and different slope. That is expected, given that the model was constructed to take by-version variability into account. The by-version coefficients (intercept and slope) for the effect of predictive variables are different for each version. However, version 10.1 and 10.5 values are quite similar to each other. It is also noticed that the slopes are in different directions for each version. This means that, despite individual version variation, there is also inconsistency in how predictive variables affect the energy consumption in different version groups. For example, EC of version 9.5 tends to go up with increasing LOC. On the other hand, EC of version 10.5 goes down with increasing LOC. *P-values* of LOC, CC and MCC were significant in the models, but *p-values* of LOCC were not significant in any of the models.

When comparing the one variable models, Model#3 has the highest R^2 followed by Model#4 and Model#1. Among the two-variable models, Model#5 has the highest R^2, followed by Model#6 and Model#7. When comparing all the models, two-variables models have better fit than the one-variable models. For example, R^2 of Model#5 increased 67% over Model#1. Model#2 has the lowest significance and the lowest R^2. This was expected because there is no significant correlation between EC-LOCC pairs. When LOCC was used in the two variable

Table 5.3: DB2 Dataset: Random effect models coefficients and model evaluations.

Model	Variables	Versions	Intercept	Slope	p-value	R-squared
1	LOC	v.9.5	-0.229	0.157	0.009**	0.534
		v.10.1	-0.637	0.438		
		v.10.5	0.867	-0.596		
2	LOCC	v.9.5	1.216	0.040	0.475	0.052
		v.10.1	-0.671	0.202		
		v.10.5	-0.544	-0.242		
3	CC	v.9.5	-0.033	0.016	0.007**	0.666
		v.10.1	-0.450	0.216		
		v.10.5	0.483	-0.232		
4	MCC	v.9.5	-0.031	0.015	0.008**	0.657
		v.10.1	-0.463	0.223		
		v.10.5	0.494	-0.238		
5	LOC LOCC	v.9.5	6.34e-18/0.041	-5.83e-18/-0.021	0.004** 0.145	0.893
		v.10.1	-6.41e-17/-0.281	5.89e-17/0.145		
		v.10.5	5.78e-17/0.241	-5.31e-17/-0.123		
6	LOCC CC	v.9.5	0.093/0	-0.044/3.53e-15	0.129 0.004**	0.865
		v.10.1	-0.289/0	0.136/-1.04e-14		
		v.10.5	0.195/0	-0.092/6.81e-15		
7	LOCC MCC	v.9.5	0.059/0	-0.025/1.93e-16	0.131 0.004**	0.865
		v.10.1	-0.245/0	0.102/-4.32e-14		
		v.10.5	0.186/0	-0.077/4.45e-14		

Significance: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

models, it managed to increase the R^2 (e.g., Model#1 and Model#5).

Performance evaluations of the models are presented in Table 5.4.

Regarding accuracy evaluation of the models, all the models show high accuracy. Among all the models, Model#5 has the lowest prediction error (highest accuracy) followed by Model#6 and Model#7. Among the one-variable models, Model#1 has the highest accuracy.

In summary, two-variable models achieve high R^2, low RMSE (0.282 in Model#5) and low MAE (0.197 in Model#5). On the other hand, by looking at the inconsistency in coefficients of version groups (Table 5.3), the success of these models in terms of EC prediction in the DB2 dataset may not be conjectured.

Table 5.4: DB2 dataset: Random effect model performance evaluations

Model	Predictive Variables	R-squared	MAE	RMSE	PRED(25)
1	LOC	0.534	0.227	0.292	0.831
2	LOCC	0.052	0.231	0.317	0.470
3	CC	0.666	0.237	0.306	0.892
4	MCC	0.657	0.238	0.307	0.766
5	LOC, LOCC	0.893	0.197	0.282	0.866
6	LOCC, CC	0.865	0.205	0.286	0.810
7	LOCC, MCC	0.865	0.206	0.287	0.818

Checking Random Effect Model Assumptions

Before making more inferences about the DB2 random effects model, the underlying distributional assumptions should be checked. There are two basic distributional assumptions for the mixed-effects models [35]:

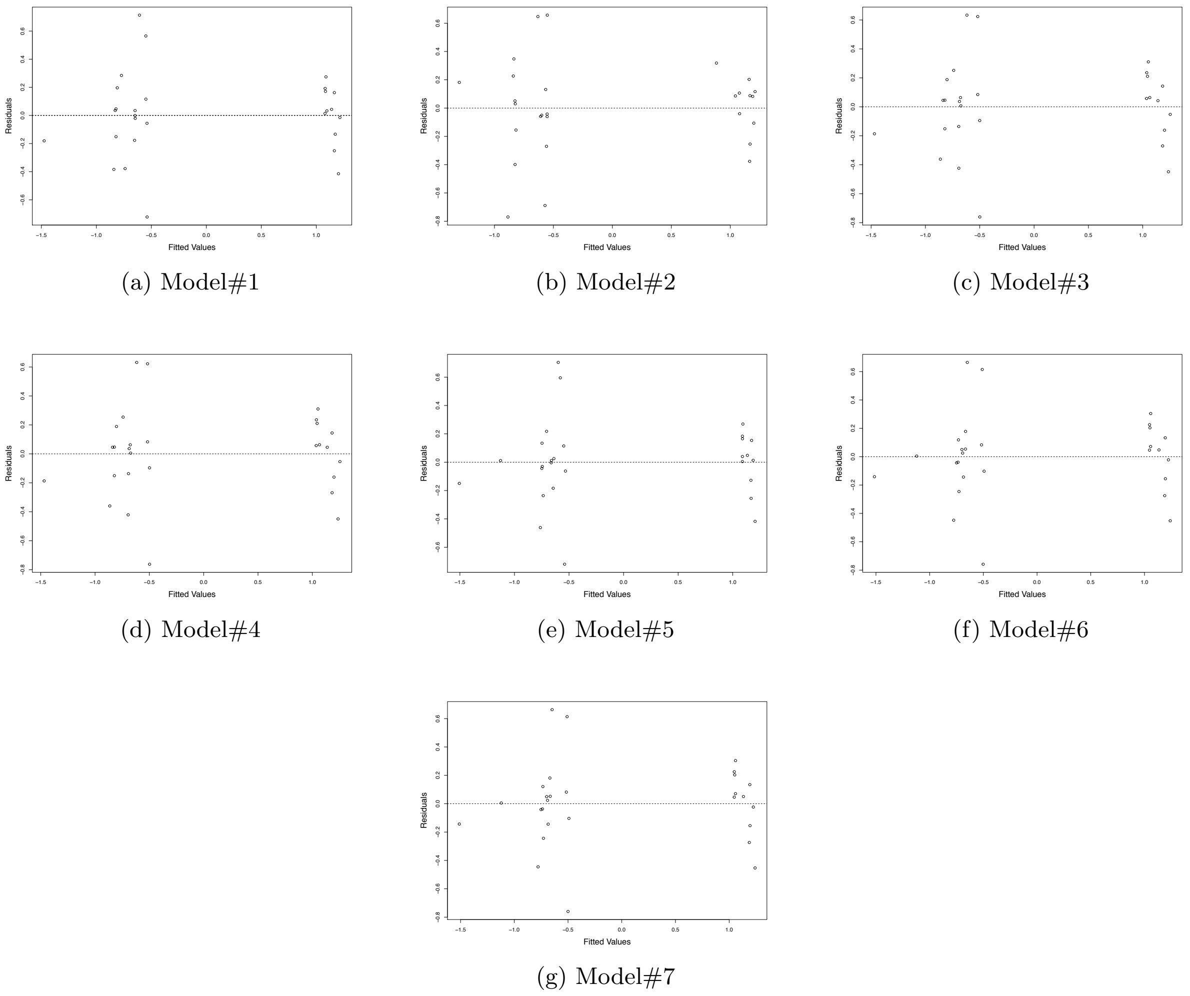
- *Assumption 1:* Errors are independent and error variance are non-constant. Residual plots must be free of any patterns.
- *Assumption 2:* The random effects are normally distributed and are independent for different groups.

The most useful methods for checking these assumptions are based on diagnostic plots of the residuals, the fitted values, and the estimated random effects plots. Fitted values versus residuals plot was used to check Assumption 1, normal quantile-quantile plot (Q-Q plot) plot of estimated random effects was used to check Assumption 2. The normal Q-Q plot is the most commonly used and effective diagnostic tool for checking normality of the data.

Assumption 1: Figure 5.2 shows that the residual plots are not free of patterns. This suggests that other than the version effect, there may be other factors effecting the behaviour of the response variable. They are outside of the scope of this dissertation, future directions for exploring these variables are provided in section 6.4.

Assumption 2: Figures 5.3 (a-g) show normal Q-Q plots of all the DB2 mixed effect models.

To support Q-Q plots, standard normal quantiles versus random effect quantiles plots are given in Figures 5.4 (a-g).



(a) Model#1 (b) Model#2 (c) Model#3

(d) Model#4 (e) Model#5 (f) Model#6

(g) Model#7

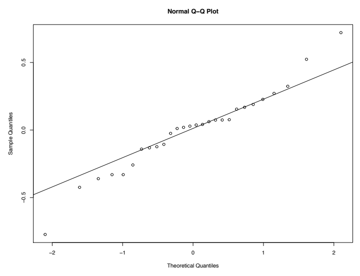
Figure 5.2: Residual scatter plots of standardized residuals versus fitted values for for DB2 models.

From Figures 5.3 (a-g) and 5.4 (a-g) random effect model residuals show approximately normal distribution except, Model#2 where the predictive variable is LOCC.

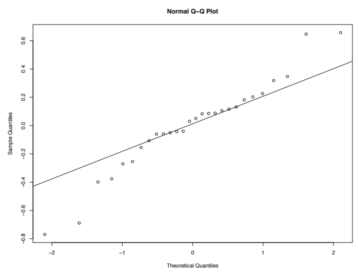
To support the graphical methods, the Shapiro-Wilk test were performed. The test results are given in Table 5.5.

The Shapiro-Wilk test results show that all *p-values* are greater than the alpha level, which results in all the residuals being sufficiently normally distributed.

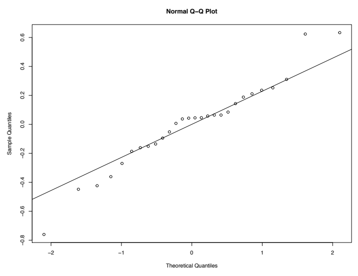
In summary, Assumption 2 plots and the Shapiro-Wilk results support the normality assumption of the residuals of the models. On the other hand, Assumption 1 plots do not support



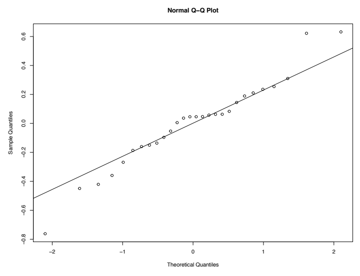
(a) Model#1



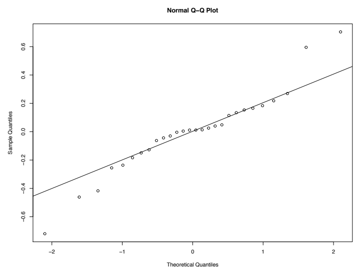
(b) Model#2



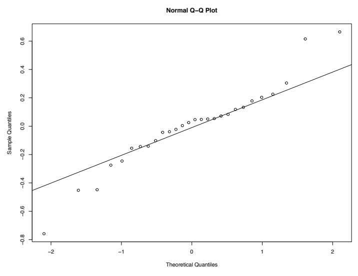
(c) Model#3



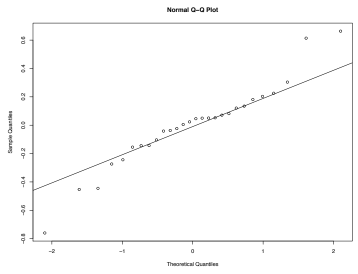
(d) Model#4



(e) Model#5



(f) Model#6

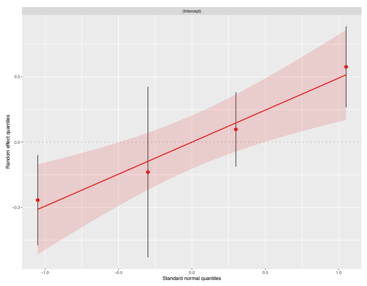


(g) Model#7

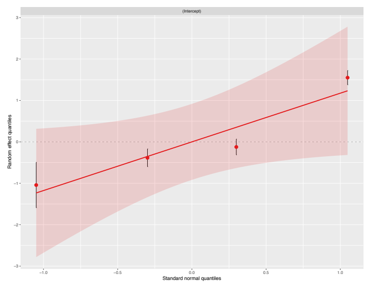
Figure 5.3: Normal Q-Q plots of residuals for DB2 models.

normality, the residuals do not show random patterns. This suggests that there may be a model fitting issue. One approach to address fitting issue of a model is improving model performance by increasing model flexibility. To increase model flexibility, more predicting variables should be added to the model. However, in this study, there is no such variable available in the dataset.

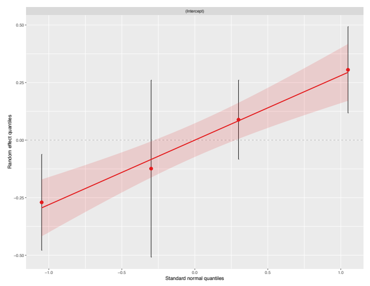
Although, fixed/random effect models are useful for making predictions about new groups, with small group numbers it reduces to a classical regression model. As more data become available it makes sense to predict more. In small datasets it is hard to learn so much, and it is not necessarily beneficial to fit a more complex model when the resulting uncertainties will be so substantial.



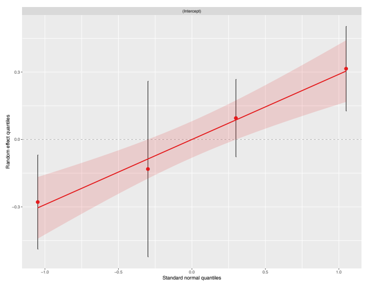
(a) Model#1



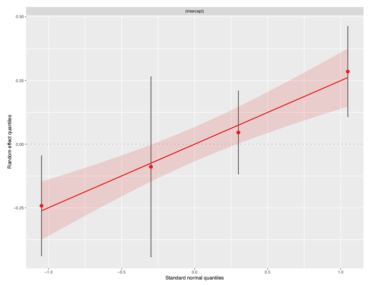
(b) Model#2



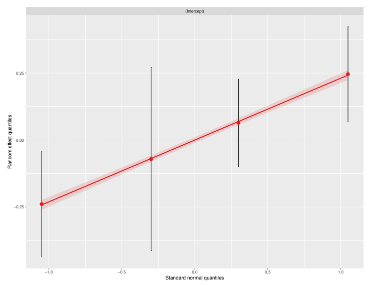
(c) Model#3



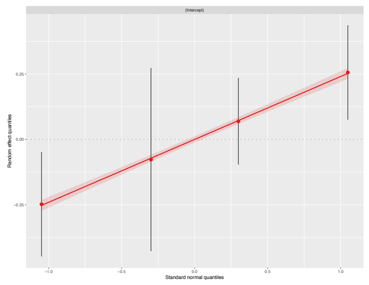
(d) Model#4



(e) Model#5



(f) Model#6



(g) Model#7

Figure 5.4: Standard normal quantiles versus random effect quantiles for DB2 models.

In order to overcome non-normality of residuals, DB2 version groups were also analyzed individually in section 5.1.4.

5.1.4 Models and Results for DB2 Version Groups

DB2 Version 9.5 Models and Results

Table 5.6 shows model assessments and their prediction performance on the DB2 version 9.5 group data. Results reveal that the models and their predictor variables are not significant at the α level. Moreover, prediction errors (MAE and RMSE) of all models are very high. Except Model#1 and Model#5, all models have very low prediction performance with high

Table 5.5: DB2 mixed-effect model residual Shapiro-Wilk test results

Model	**Predictor Variables**	**W**	**p-value**
1	LOC	0.944	0.146
2	LOCC	0.956	0.269
3	CC	0.949	0.196
4	MCC	0.963	0.404
5	LOC, LOCC	0.938	0.104
6	LOCC, CC	0.948	0.176
7	LOCC, MCC	0.948	0.177

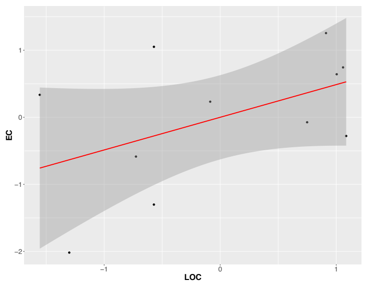
Table 5.6: Prediction performance of models on dataset DB2-version 9.5.

Model	**Predictor Variables**	**p-value**	**R-squared**	**MAE**	**RMSE**	**PRED(25)**
1	LOC	0.128	0.237	0.883	1.080	0.75
2	LOCC	0.984	4.199e-05	0.926	1.222	0.4
3	CC	0.126	0.239	0.894	1.098	0.52
4	MCC	0.125	0.241	0.861	1.079	0.49
5	LOC, LOCC	0.229	0.308	1.079	1.079	0.63
6	LOCC, CC	0.236	0.303	1.021	1.427	0.375
7	LOCC, MCC	0.232	0.306	1.049	1.276	0.45

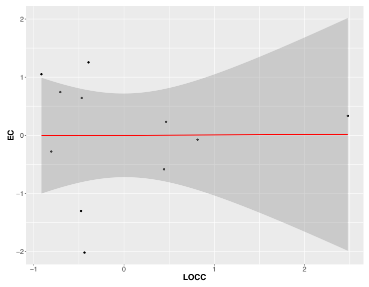
Significance: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

prediction errors. Poor prediction accuracy and performance may be explained by the lack of data. Accuracy of the model on training and testing data could be poor because the learning algorithm did not have enough data to learn from. Performance can be improved by increasing the amount of training dataset. The data points are already limited in the DB2 group data, therefore increasing the training set did not have much effect on the performance of the models in the group. Figures 5.5 (a-g) show DB2 v.9.5 regression models graphs. The graphs also support model performance results.

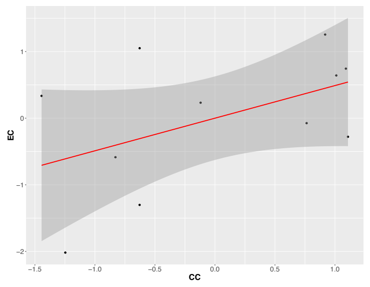
In summary, by reviewing significance and performance results of DB2 version 9.5 models, only Model#1 with predictor variable LOC may have potential to show better prediction performance than the other models when the training data is available.



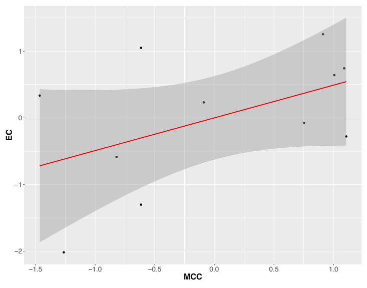
(a) Model#1



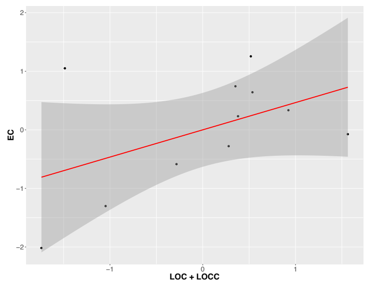
(b) Model#2



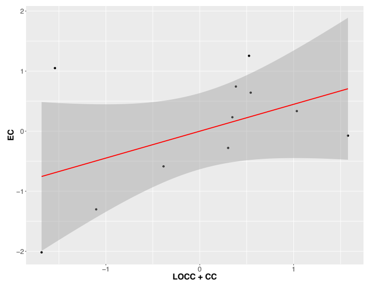
(c) Model#3



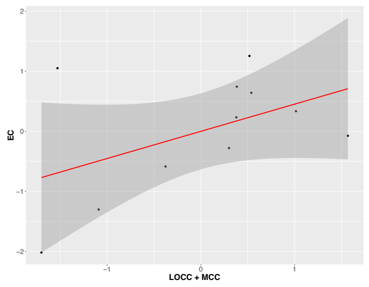
(d) Model#4



(e) Model#5



(f) Model#6



(g) Model#7

Figure 5.5: Regression model graphs for DB2 v.9.5; the grey area show the 95% confidence intervals.

Checking Regression Assumptions: DB2-version 9.5.

Assumption 1: The parameters of the linear regression model are numeric and p-values are significant (Table 4.19).

Assumption 2: The mean of the residuals are close to zero (Model#1=-1.5e-17, Model#2 = 1.1e-17, Model#3= 2.8e-17, Model#4=-4.1e-17, Model#5= -3.8e-17, Model#6= -1e-16, Model #7=-1.5e-17).

Assumption 3: For two-variable models, multicollinearity must be sufficiently low between the predictors. If Variance Inflation Factor (VIF)>10, there is an indication for multicollinearity to be present. All two-variable models VIF values are smaller than 10. Thus, this assumption holds true for all the two-variable models.

Assumption 4: The variance in the predictors must be larger than 0. All the variance in the predictors is greater than 0, thus this assumption holds true for all the predictors.

Assumption 5: The residuals must be normally or approximately normally distributed. Figure 5.6 (a-g) show Q-Q plots of the model residuals.

To support the graphical method of normality check, the Shapiro-Wilk test was applied to the DB2 version 9.5 group. Results are presented in Table 5.7.

Table 5.7: DB2 v.9.5 residuals Shapiro-Wilk test results.

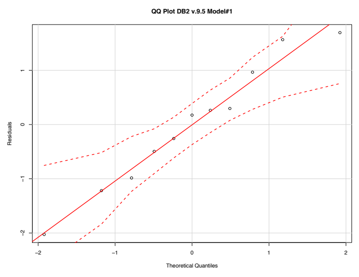
Model	**Predictor Variables**	**W**	**p-value**
1	LOC	0.889	0.970
2	LOCC	0.586	0.975
3	CC	0.938	0.945
4	MCC	0.925	0.974
5	LOC, LOCC	0.536	0.941
6	LOCC, CC	0.961	0.979
7	LOCC, MCC	0.572	0.944

Table 5.7 shows that all model‘s residual *p-values* are greater than 0.05. W statistics of Model#1, Model#3, Model#4, and Model#6 are closer to one. Assumption check results suggest all assumptions hold true for all the DB2 version 9.5 models.

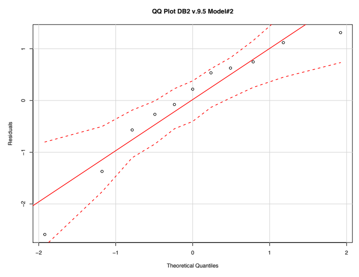
In summary, the results suggest EC-LOC and EC-CC relationships and provide base for future EC prediction studies. However, the amount of training data set should be considered for increasing the models prediction performance (Table 5.6).

DB2 Version 10.1 Models and Results

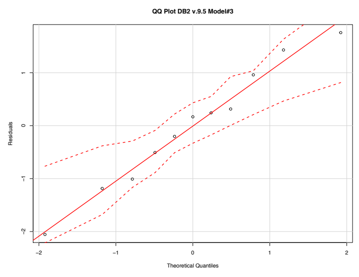
Table 5.8 shows model assessments and their prediction performance on DB2 version 10.1 group data. Results reveal that the models and their predictor variables are not significant at the α level. Moreover, prediction accuracy (MAE and RMSE) of all models was very poor. Except Model#1, all models had very low prediction performance (PRED(25)) with low accuracy. Similar to DB2 version 9.5 data results, low prediction accuracy and performance of group 10.1



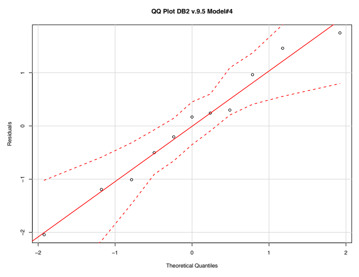
(a) Model#1



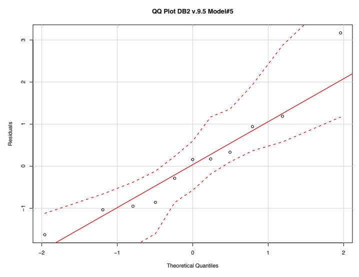
(b) Model#2



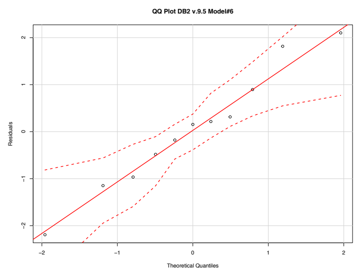
(c) Model#3



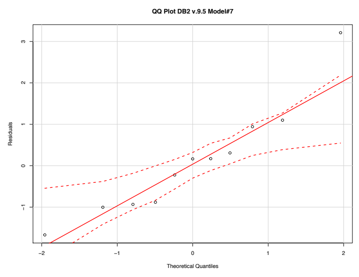
(d) Model#4



(e) Model#5



(f) Model#6



(g) Model#7

Figure 5.6: Residual plots for DB2 version 9.5 models; the dashed lines show the 95% confidence intervals.

may be explained by the lack of training data. The data are already limited in DB2 version groups, therefore increasing the training set may not effect group model performance. Figures 5.7 (a-g) show that DB2 v.9.5 regression graphs. The graphs also support model performance results.

In summary, significance and performance results of DB2 version 10.1 models suggest EC-LOC, EC-CC, and EC-MCC relationships. These results provide base for future EC prediction studies. However, the amount of training data set should be considered to increasing the model‘s prediction performance (5.8).

Table 5.8: Prediction performance of models on dataset DB2-version 10.1.

Model	Predictor Variables	p-value	R-squared	MAE	RMSE	PRED(25)
1	LOC	0.101	0.445	1.172	1.319	0.656
2	LOCC	0.588	0.062	5.986	9.629	0.495
3	CC	0.074 ·	0.502	1.043	1.223	0.495
4	MCC	0.072 ·	0.506	1.083	1.187	0.460
5	LOC, LOCC	0.084 ·	0.709	4.063	9.069	0.495
6	LOCC, CC	0.065 ·	0.749	4.604	6.495	0.500
7	LOCC, MCC	0.062 ·	0.744	5.774	6.206	0.495

Significance: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Checking Regression Assumptions: DB2-version 10.1.

Assumption 1: The parameters of the linear regression model are numeric and p-values are significant (Table 4.20).

Assumption 2: The mean of the residuals are close to 0 (Model#1=4.7e-17, Model#2=2.3e-17, Model#3=7.1e-17, Model#4=6.3e-17, Model#5=7.9e-17, Model#6=-7.9e-16, Model#7=-2.7e-17).

Assumption 3: For two-variable models, multicollinearity must be sufficiently low between the predictors. If Variance Inflation Factor (VIF)>10, there is an indication for multicollinearity to be present. All two-variable models VIF values are smaller than 10. Thus, this assumption holds true for all the two-variable models.

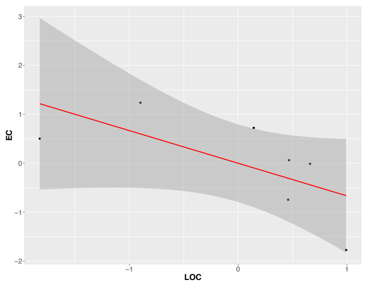
Assumption 4: The variance in the predictors must be larger than 0. All the variance in the predictors is greater than 0, thus this assumption holds true for all the predictors.

Assumption 5: The residuals must be normally or approximately normally distributed. Figures 5.6 (a-g) Q-Q plots of the model residuals.

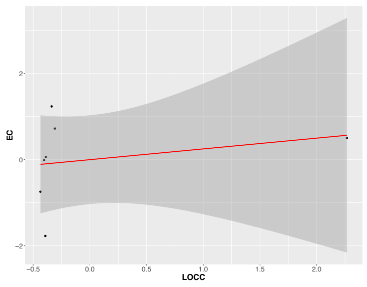
To support the graphical method of normality check, the Shapiro-Wilk test was applied to the DB2 version 10.1 group. Results are presented in Table 5.9.

From Figure 5.8 and Table 5.9, all model residuals show sufficiently normal distribution.

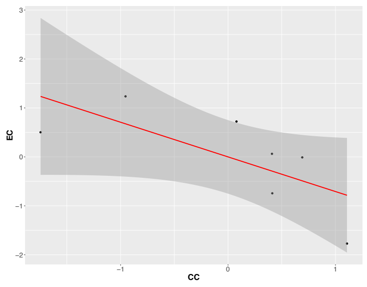
The prediction performance results in Table 5.8 and assumption check results suggest similar relations to DB2 v.9.5. EC-LOC and EC-CC relations may be used for prediction of EC in



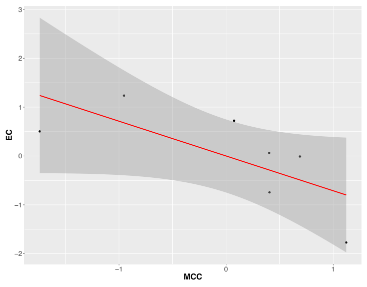
(a) Model#1



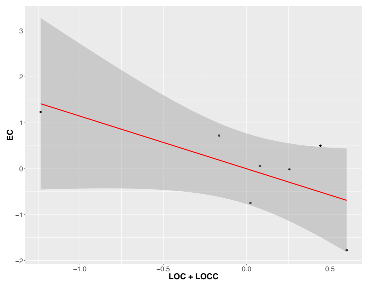
(b) Model#2



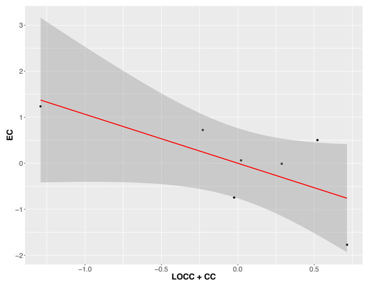
(c) Model#3



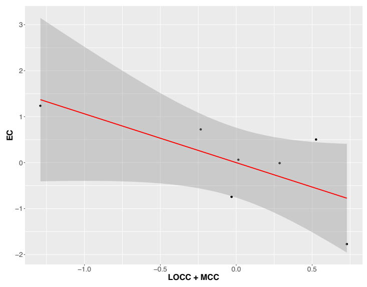
(d) Model#4



(e) Model#5



(f) Model#6



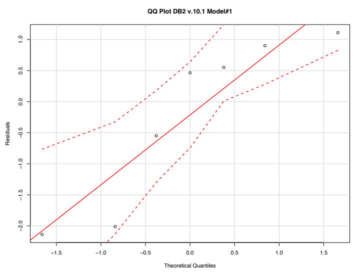
(g) Model#7

Figure 5.7: Regression model graphs for DB2 v.10.1; the grey area show the 95% confidence intervals.

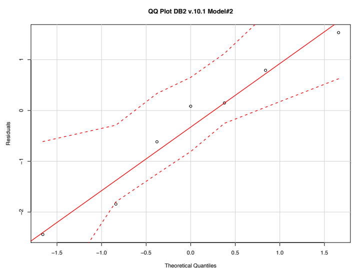
DB2 dataset. However, the amount of training data set should be considered to increasing the model‘s prediction performance.

DB2 Version 10.5 and 11.1 Models and Results

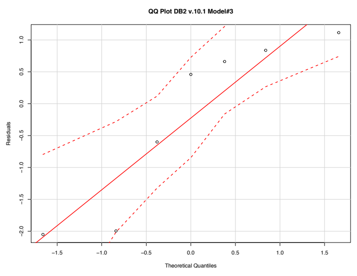
Table 5.10 presents model assessments and their prediction performance on DB2 version 10.5 and 11.1 data. Results show that only single-variable models are significant. Prediction accuracy (MAE and RMSE) of all models are very low. Except Model#1, all the models had low prediction performance (PRED(25)) with low prediction accuracy. Similar to DB2 v.9.5



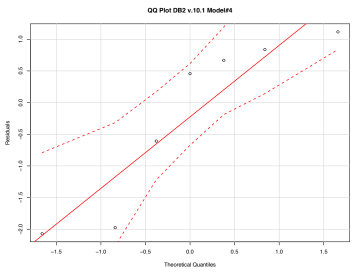
(a) Model#1



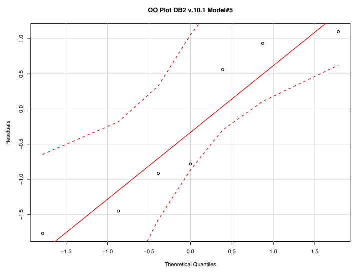
(b) Model#2



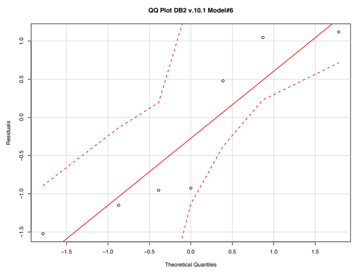
(c) Model#3



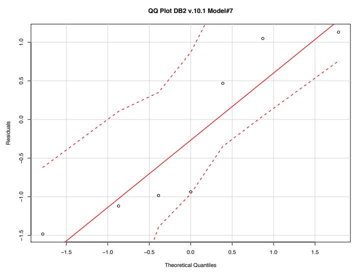
(d) Model#4



(e) Model#5



(f) Model#6



(g) Model#7

Figure 5.8: Residual plots for DB2 version 10.1 models; the dashed lines show the 95% confidence intervals.

and v.10.1 group data results, low prediction accuracy and poor performance of group 10.5 and 11.1 models may be explained by the lack of training data. Figures 5.9 (a-g) show DB2 v.9.5 regression models graphs. The graphs also support model performance results.

In summary, significance and performance results of DB2 version 10.5 and 11.1 models results suggest similar relations to DB2 v.9.5 and v.10.1. EC-LOC, EC-CC, and EC-MCC relations may base for future prediction of EC in DB2 dataset. However, the amount of training data set should be considered to increasing the model‘s prediction performance.

Table 5.9: DB2 v.10.1 residuals Shapiro-Wilk test results

Model	Predictor Variables	W	p-value
1	LOC	0.339	0.901
2	LOCC	0.874	0.966
3	CC	0.234	0.882
4	MCC	0.224	0.880
5	LOC, LOCC	0.562	0.931
6	LOCC, CC	0.311	0.896
7	LOCC, MCC	0.288	0.892

Table 5.10: Prediction performance of models on dataset DB2 v10.5-v.11.1.

Model	Predictor Variables	p-value	R-squared	MAE	RMSE	PRED(25)
1	LOC	0.032*	0.455	0.890	0.847	0.67
2	LOCC	0.045*	0.412	0.882	1.185	0.50
3	CC	0.050*	0.390	0.862	0.972	0.50
4	MCC	0.050*	0.384	0.875	0.972	0.50
5	LOC, LOCC	0.113	0.463	1.041	1.291	0.50
6	LOCC, CC	0.141	0.428	0.985	1.118	0.50
7	LOCC, MCC	0.142	0.427	1.018	1.239	0.50

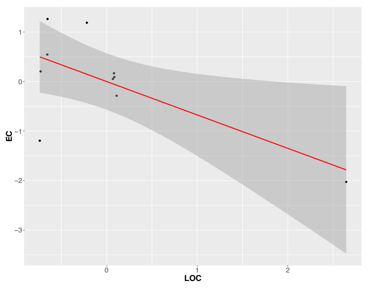
Significance: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Checking Regression Assumptions: DB2 v.10.5-11.1

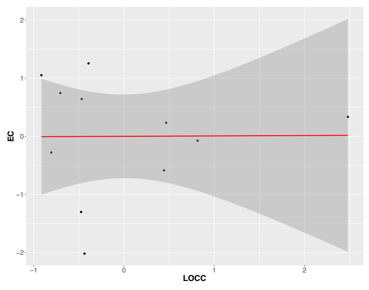
Assumption 1: The parameters of the linear regression model are numeric and linear (Table 4.21) all p-values are close to 0.

Assumption 2: The mean of the residuals are close to 0 (Model#1=3.1e-17, Model#2 =8.3e-17, Model#3= 5.5e-18, Model#4=4.4e-17, Model#5=-7.7e-17, Model#6=8.3e-17, Model #7=6.1e-17).

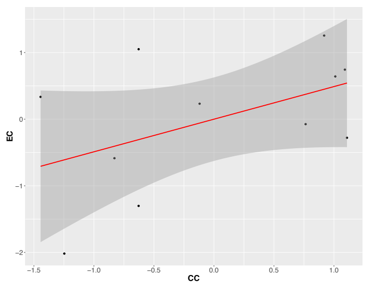
Assumption 3: For two-variable models, multicollinearity must be sufficiently low between the predictors. If Variance Inflation Factor (VIF)>10, there is an indication for multicollinearity to be present. All two-variable models VIF values are smaller than 10. Thus, this assumption holds true for all the two-variable models.



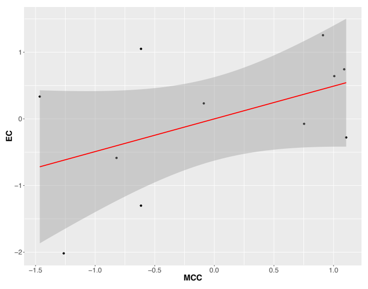
(a) Model#1



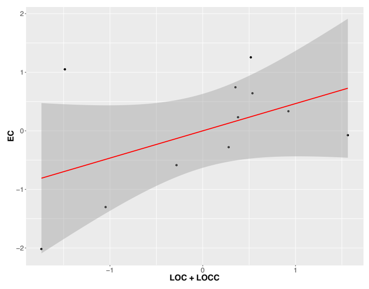
(b) Model#2



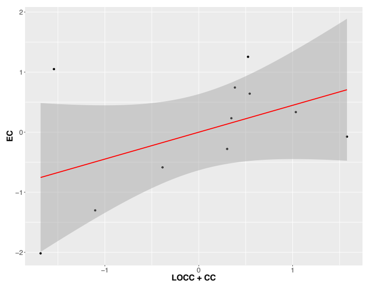
(c) Model#3



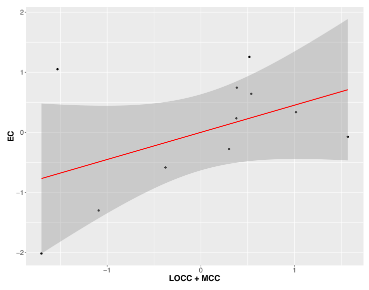
(d) Model#4



(e) Model#5



(f) Model#6



(g) Model#7

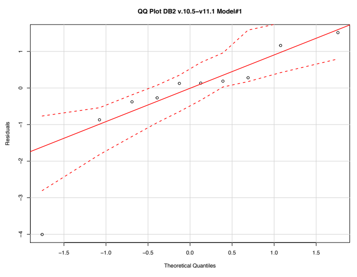
Figure 5.9: Regression model graphs for DB2 v.10.5-v11.1; the grey area show the 95% confidence intervals.

Assumption 4: The variance in the predictors must be larger than 0. All the variance in the predictors is larger than 0, thus this assumption holds true for all the predictors.

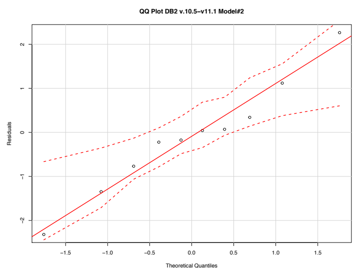
Assumption 5: The residuals must be normally or approximately normally distributed. Figure 5.10 (a-g) show Q-Q plots of the model residuals.

To support the graphical method of normality check, the Shapiro-Wilk test was applied to v.10.5-v11.1 dataset. Results are presented in Table 5.11.

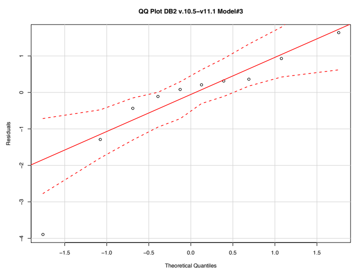
Table 5.7 shows that all models residuals *p-value* are smaller than 0.05 except Model#2. W statistics of all the models are closer to one. Thus, except Model#1, all the model residuals



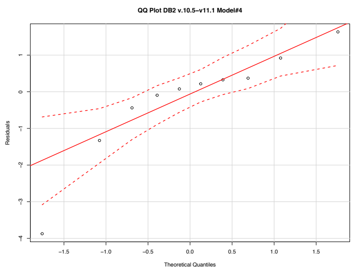
(a) Model#1



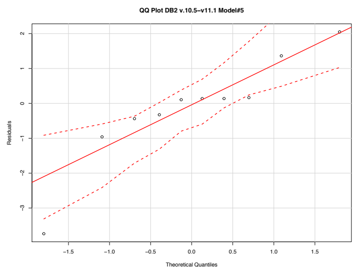
(b) Model#2



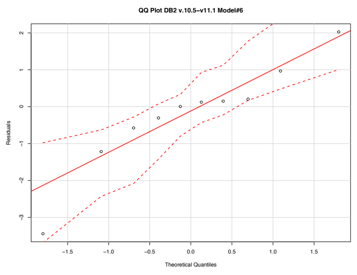
(c) Model#3



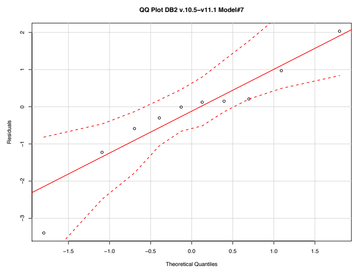
(d) Model#4



(e) Model#5



(f) Model#5



(g) Model#5

Figure 5.10: Residual plots for DB2 v10.5-v11.1 models; the dashed lines show the 95% confidence intervals.

follow sufficiently normal distribution.

The prediction performance results in Table 5.10 and assumption check results suggest similar relations to DB2 v.9.5 and v.10.1 group data results. EC-LOC and EC-CC relations may be used for prediction EC in DB2 dataset. However, the amount of training data set should be considered to increasing the model‘s prediction performance.

Summary of DB2 individual group model analysis: Individual DB2 version group model graphs Figure 5.5,5.7 and 5.9 showed that there was inconsistency in regression coefficients of versions.This result also support inconsistent results in random effect varying intercept

Table 5.11: DB2 v.10.5-v11.1 residuals Shapiro-Wilk test results.

Model	Predictor Variables	W	p-value
1	LOC	0.880	0.130
2	LOCC	0.959	0.780
3	CC	0.927	0.417
4	MCC	0.930	0.443
5	LOC, LOCC	0.869	0.097
6	LOCC, CC	0.927	0.424
7	LOCC, MCC	0.930	0.450

and slope coefficients Table 5.3. The results also showed that all models have very low accuracy (MAE and RMSE) and medium prediction performance. This may be due to the fact that the effect of versions is not represented in individual analysis. Additionally, the individual group data were quite small (5-10 subversions), therefore lack of individual group data may results in low accuracy. If more data had been available, the prediction performance would be improved.

5.1.5 Models and Results for Dataset: MYSQL

Table 5.12: Prediction performance of models on dataset MYSQL-MyISAM.

Model	Predictor Variables	p-value	R-squared	MAE	RMSE	PRED(25)
1	LOC	1.67e-11 ***	0.753	0.465	0.560	0.817
2	LOCC	0.981	0.218	1.023	1.030	0.712
3	CC	0.0765 .	0.292	0.797	1.002	0.694
4	MCC	0.0511 .	0.265	0.771	1.009	0.729
5	LOC, CC	1.441e-14***	0.829	0.430	0.457	0.823
6	LOC, MCC	2.569e-14***	0.824	0.423	0.451	0.843

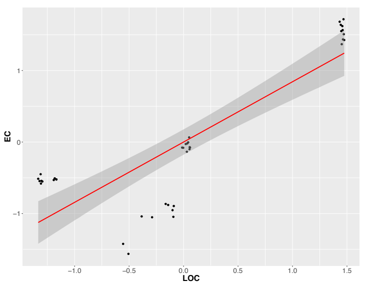
Significance: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Table 5.12 presents model assessments and their prediction performance on Dataset MyISAM. Results show that among the one variable models, regarding significance and R^2, Model#1 has the highest significance followed by Model#3 and Model#4. Regarding prediction performance, Model#1 has the highest performance (PRED(25) being 1.84) followed by Model#3 and Model#4. All the MAE and RMSE results are somewhat similar, except Model#2. Similar to dataset DB2, Model#1 with LOC predictor is very successful at predicting EC in MyISAM. Among the two-variable models, Model#5 and Model#6 had similar

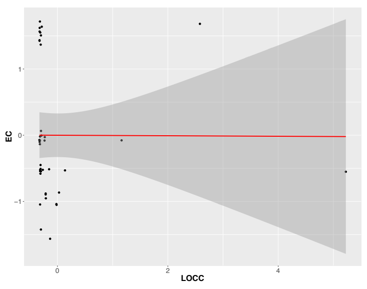
R^2 values. Regarding prediction performance, Model#5 has better prediction with PRED(25) being 0.84 and the lowest MAE and RMSE with values of 0.358 and 0.346, respectively.

CC shows moderate significance (0.07), but low predictive performance in the one-variable model. When it is used in two-variable models (Model#5), it managed to reduce prediction error in terms of RMSE down to 18%, MAE down to 7.5% with 82% of predictions (PRED(25)).

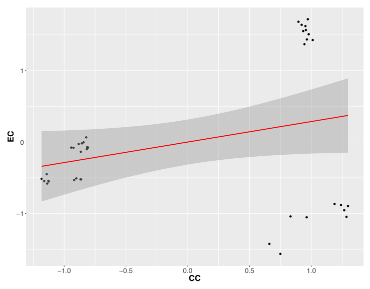
When comparing all the models, Model#6 has the best prediction performance, followed by Model#5 and Model#1, respectively. When comparing R^2 values, Model#5 has 10% increased in R^2 than Model#1. Model#2 has the lowest significance, lowest R^2 and lowest performance. This was expected because there is no significant correlations between LOCC-EC pairs. Figures 5.11 (a-f) show MYSQL-MyISAM regression models graphs. They also support model performance results.



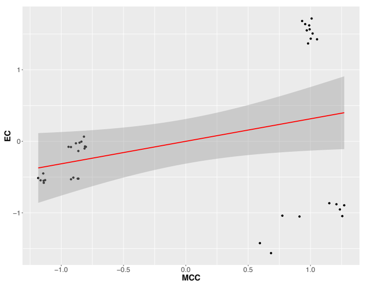
(a) Model#1



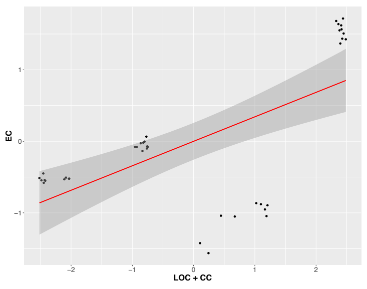
(b) Model#2



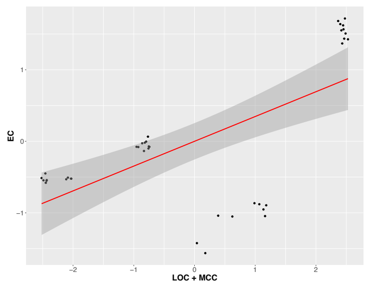
(c) Model#3



(d) Model#4



(e) Model#5



(f) Model#6

Figure 5.11: Regression model graphs for MyISAM; the grey area show the 95% confidence intervals.

In summary, two-variable models achieve low RMSE rates (0.457 in Model#5), with low MAE (0.430 in Model#5). In terms of PRED(25), similar to error performance, Model#5 and Model#6 have the highest predictive performance (PRED(25)) followed by the one variable model (Model#1).

Checking Regression Assumptions: MyISAM

Assumption 1: The parameters of the linear regression model are numeric and linear (Table 4.24), all *p-values* are close to 0.

Assumption 2: The mean of the residuals are close to 0 (Model#1=8.5e-18, Model#2= 8.8e-17, Model#3= 1.2e-17, Model#4= 6.6e-18, Model#5= 1.9e-17, Model#6= 9.2e-18).

Assumption 3: For two-variable models, multicollinearity must be sufficiently low between the predictors. If Variance Inflation Factor (VIF)>10, there is an indication for multicollinearity to be present. Model#5 VIF=1.73, Model#6 VIF=1.81. All VIF's are smaller than 10, thus this assumption holds true for these models.

Assumption 4: The variance in the predictors must be larger than 0. All the variance in the predictors is much larger than 0, thus this assumption holds true for all the predictors (LOC=4.5e+10, LOCC=2.6e+10, CC=3.2e+8, MCC=2.6e+8).

Assumption 5: The residuals must be normally or approximately normally distributed. As in the case of DB2, Figure 5.12 (a-f) shows that models do not follow normal distribution.

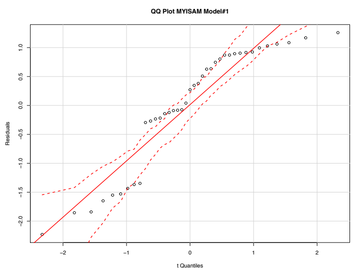
To address these non-normally residual distributions, log transformation was applied to variables, then error distributions were checked. However, log transformation could not overcome the non-normality issue.

From Table 5.12, Model#2 shows no significance (*p-value*= 0.981), predictive performance PRED(25) was 0.71 and it has the highest error rate ($RMSE$ = 1.03) among the MyISAM models. Moreover, predictor LOCC showed no correlation with EC (Table 4.24). Therefore, error plot Model#2 (Figure 5.12 (b) shows non-normal distribution. Although assumptions 1-5 hold true for all the MyISAM models, Figures 5.12 (a-f) show non-normally-distributed residuals.

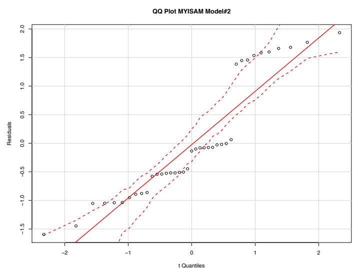
To support the graphical method of normality, the Shapiro-Wilk test was applied to MyISAM dataset. Results of the test are presented in Table 5.13. Although all model‘s residual *p-value* are smaller than 0.05, Model#1, Model#3, and Model#4 W statistics are closer to one.

In summary, findings suggest relationships between EC and LOC, EC and CC, EC and MCC. These relations may provide a foundation for EC prediction on Dataset MYSQL-MyISAM.

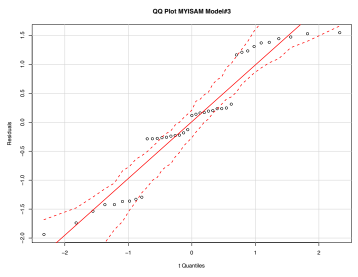
Table 5.14 presents model assessments and their prediction performance on Dataset Innodb. One-variable model results show that, regarding significance, R^2, and prediction performance (PRED(25)) Model#3 has the highest significance and performance followed by Model#4 and Model#1. All two-variable models show significance, and their R^2 values are 8% higher than



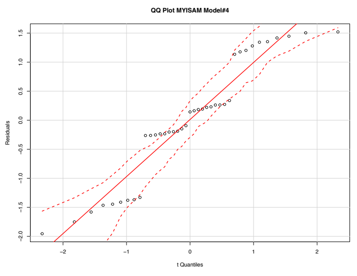
(a) Model#1



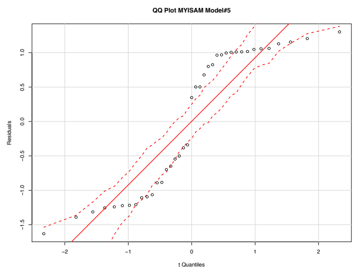
(b) Model#2



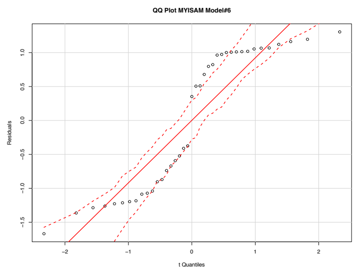
(c) Model#3



(d) Model#4



(e) Model#5



(f) Model#6

Figure 5.12: Residual plots for MYISAM models; the dashed lines show the 95% confidence intervals.

Table 5.13: MyISAM residuals Shapiro-Wilk test results

Model	Predictor Variables	W	p-value
1	LOC	0.920	0.001
2	LOCC	0.862	0.0002
3	CC	0.918	0.008
4	MCC	0.916	0.006
5	LOC, CC	0.842	7.105e-05
7	LOC, MCC	0.842	6.816e-05

one-variable models. They also show 8% better prediction performance (PRED(25)) on average than one-variable models. Regarding model accuracy, MAE and RMSE results of all InnoDB models are similar and had medium accuracy, except model#2 which had low accuracy.

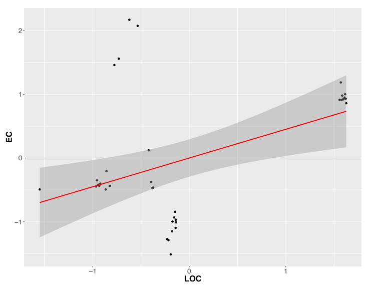
When comparing all the models, Model#6 had the highest prediction performance, followed by Model#5 and Model#4, respectively. When comparing their R^2, two-variables model Model#5 has 11% increased in R^2 than Model#1. Similar to other MyISAM and DB2 datasets, Model#2 with LOCC did not show any significance and it had low accuracy and low performance. These results were expected, because there was no significant correlation found between

Table 5.14: Prediction performance of models on dataset MYSQL-InnoDB.

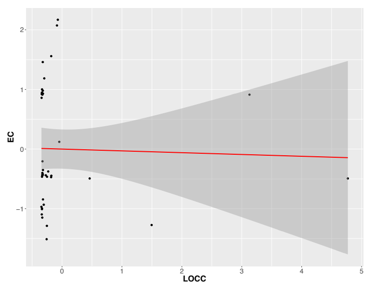
Model	Predictor Variables	p-value	R-squared	MAE	RMSE	PRED(25)
1	LOC	0.00397 **	0.544	0.590	0.574	0.747
2	LOCC	0.855	0.158	0.760	0.849	0.612
3	CC	3.39e-08 ***	0.566	0.436	0.659	0.791
4	MCC	4.43e-08 ***	0.560	0.449	0.622	0.801
5	LOC, CC	1.215e-07***	0.604	0.458	0.580	0.850
6	LOC, MCC	1.527e-07***	0.595	0.466	0.686	0.852

Significance: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

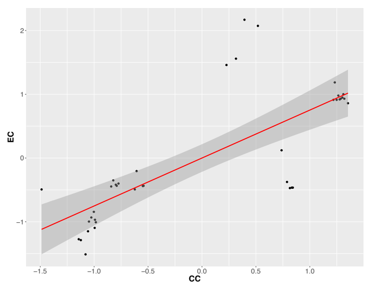
EC and LOCC. Figures 5.13 (a-f) show MYSQL-InnoDB regression models graphs. They also support model performance results.



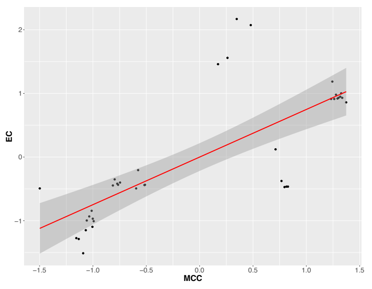
(a) Model#1



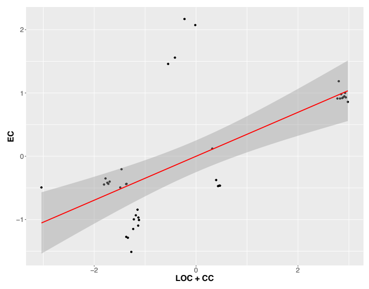
(b) Model#2



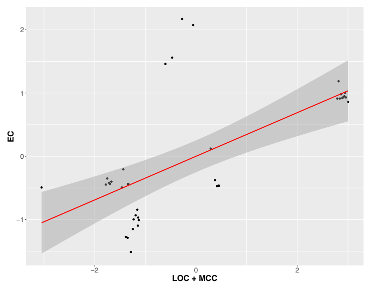
(c) Model#3



(d) Model#4



(e) Model#5



(f) Model#6

Figure 5.13: Regression model graphs for MYSQL-InnoDB; the grey area show the 95% confidence intervals.

In summary, although Model#3, Model#4, and Model#5 are significant and have high prediction performance, their accuracy is in medium level (see section 3.3.1).

Checking Regression Assumptions: InnoDB

Assumption 1: The parameters of the linear regression model are numeric and linear (Table 4.25), all *p-values* are close to 0.

Assumption 2: The mean of the residuals are close to 0 (Model#1=1.4e-18, Model#2 =1.1e-16, Model#3= 6.3e-18, Model#4= 7.5e-17, Model#5= 2.1e-17, Model#6= 1.5e-17).

Assumption 3: For two-variable models, multicollinearity must be sufficiently low between the predictors. If Variance Inflation Factor (VIF)>10, there is an indication for multicollinearity to be present. Model#5 VIF=2.15, Model#6 VIF=2.19. All VIF's are smaller than 10, thus this assumption holds true for the models.

Assumption 4: The variance in the predictors must be larger than 0. All the variance in the predictors is much larger than 0, thus this assumption holds true for all the predictors (LOC=3.2e+10, LOCC=6.5e+10, CC=3.5e+8, MCC=2.8e+8).

Assumption 5: The residuals must be normally or approximately normally distributed. Although Figure 5.14 (a-f) show that models do not follow normal distribution, especially Model #2.

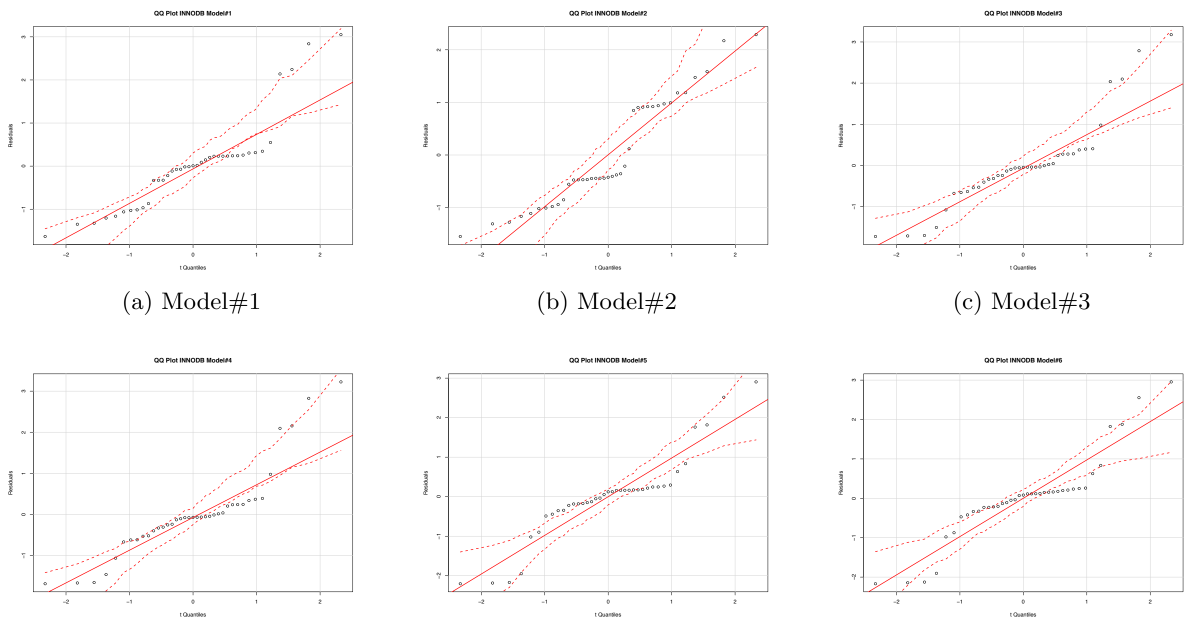
Model#2 had no significance ($p - value = 0.855$) and predictive performance PRED(25) being 0.61 and RMSE 0.85 (Table 5.14) predictor LOCC had no correlation with EC (Table 4.25). Although, assumptions 1-5 hold true for all the InnoDB models, Figures 5.14 (a-f) show non-normally-distributed residuals.

To support the graphical method of normality check, the Shapiro-Wilk test were applied to the InnoDB dataset. Results are presented in Table 5.13.

Table 5.15: InnoDB residuals Shapiro-Wilk test results.

Model	Predictor Variables	W	p-value
1	LOC	0.909	0.004
2	LOCC	0.862	0.0002
3	CC	0.877	0.0005
4	MCC	0.863	0.0.0002
5	LOC, CC	0.884	0.0007
7	LOC, MCC	0.877	0.0005

Although all models residuals *p-value* are smaller than 0.05, only Model#1 *W* statistic is closer to one.



(a) Model#1 (b) Model#2 (c) Model#3

(d) Model#4 (e) Model#5 (f) Model#6

Figure 5.14: Residual plots for InnoDB models; the dashed lines show the 95% confidence intervals.

As a result, the one-variable model with LOC and the two-variable models with LOC, CC and MCC are suggested in MYSQL dataset. If there had been more data representing MYSQL, the prediction accuracy would be improved.

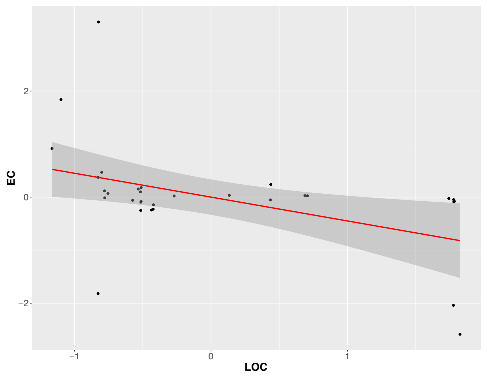
5.1.6 Models and Results for Dataset: Firefox

Table 5.16: Prediction performance of models on dataset Firefox.

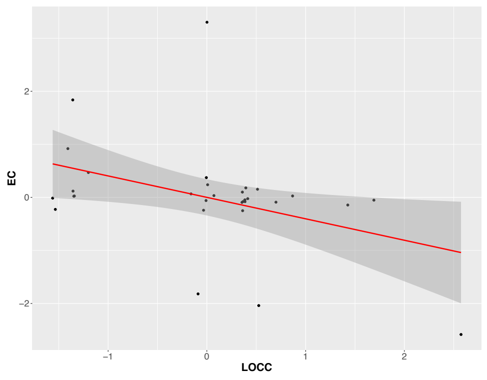
Model	Predictor Variables	p-value	R-squared	MAE	RMSE	PRED(25)
1	LOC	0.0112 *	0.343	0.626	0.932	0.530
2	LOCC	0.0241 *	0.340	0.622	1.110	0.555
3	CC	0.0143 *	0.316	0.659	0.921	0.553
4	MCC	0.0178 *	0.270	0.616	0.897	0.575

Significance: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

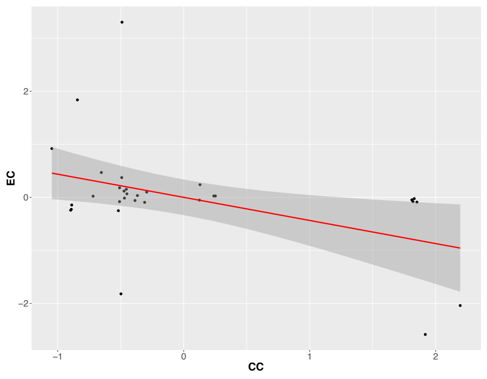
Table 5.16 presents Firefox dataset models and their prediction performance. From the correlation results (Table 4.27), significant correlations were obtained from LOC-LOCC, LOC-CC and LOC-MCC, LOCC-CC and LOCC-MCC pairs. Therefore, only one variable models were analized. All models show significance at the 0.05 level. However, their R^2 values were very low. Regarding prediction performance, they all had medium prediction performance with PRED(25) being 55% and low accuracy. The highest accuracy (RMSE and MAE) and highest prediction performance were obtained from Model#4 with predictor MCC, followed by Model#3 with predictor CC and Model#1 with predictor LOC. Figures 5.15 (a-d) show Firefox regression models graphs. They also support model performance results.



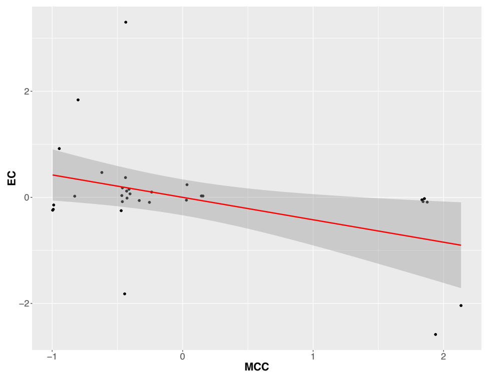
(a) Model#1



(b) Model#2



(c) Model#3



(d) Model#4

Figure 5.15: Regression model graphs for Firefox; the grey area show the 95% confidence intervals.

In summary, although Firefox dataset models had low accuracy and medium performance, model with MCC, CC or LOC may have been suggested and the model performances would be increase with more data.

Checking Regression Assumptions: Firefox

Assumption 1: The parameters of the linear regression model are numeric and linear (Table 4.26) all p-values are close to 0.

Assumption 2: The mean of the residuals are close to 0 (Model#1=7.6e-18, Model#2= 1.3e-18, Model#3= 1.6e-17, Model#4= 2.3e-17).

Assumption 3: There are no multi-variable model in Firefox dataset.

Assumption 4: The variance in the predictors must be larger than 0. All the variance in the predictors is much larger than 0, thus this assumption holds true for all the predictors (LOC=6.2e+10, LOCC=3.8e+10, CC=9.3e+8, MCC=8.6e+8).

Assumption 5: The residuals must be normally or approximately normally distributed. Figure 5.16 (a-d) show that Model#1, Model#3, and Model#4 residuals scatter along normality line. Therefore, it can be said that their residuals follow sufficiently normal distribution.

To support the graphical method of normality check, the Shapiro-Wilk test were applied to the Firefox dataset. Results are presented in Table 5.17.

Table 5.17: Firefox residuals Shapiro-Wilk test results.

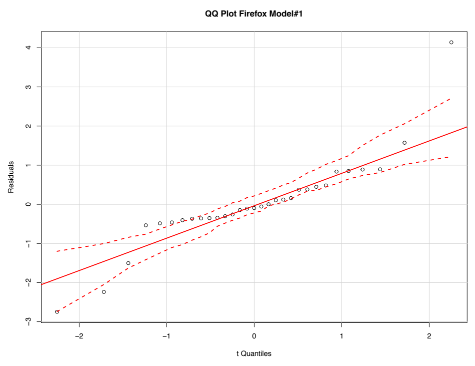
Model	**Predictor Variables**	**W**	**p-value**
1	LOC	0.901	0.005
2	LOCC	0.836	0.0002
3	CC	0.900	0.004
4	MCC	0.889	0.003

Although, all models residuals *p-value* are smaller than 0.05, W statistics of Model#1, Model#3, and Model#4 are closer to one.

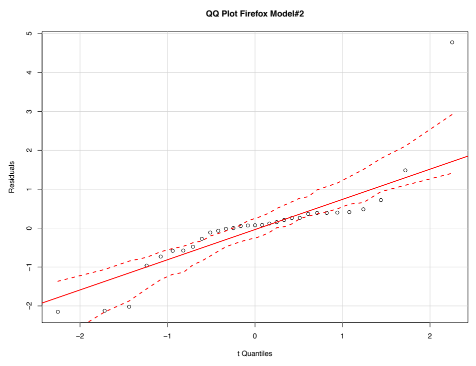
In summary, all Firefox one-variable models achieved low accuracy (high RMSE and MAE) with medium prediction performance. Although noise had been removed from the dataset, the models were not be successful in predicting EC on Firefox dataset.

5.1.7 Models and Results for Dataset: Vuze

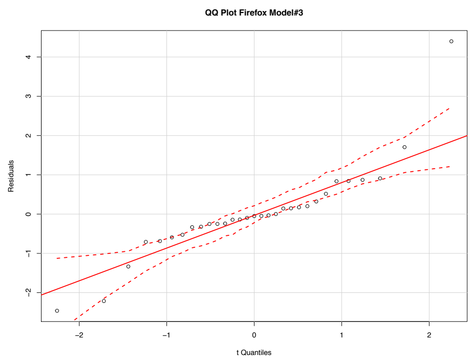
No model can be proposed with the Vuze dataset, because none of the metrics show correlation with EC (Table 4.28). This may be due to the fact that the Vuze dataset contains only



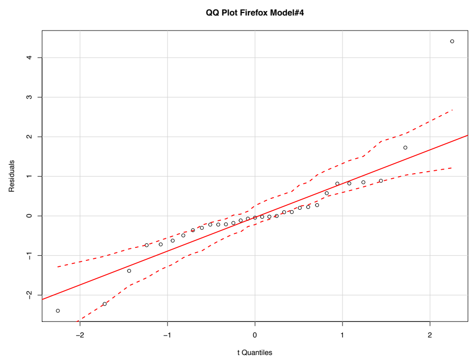
(a) Model#1



(b) Model#2



(c) Model#3



(d) Model#4

Figure 5.16: Residual plots for Firefox models; the dashed lines show the 95% confidence intervals.

subrevisions from which they could not capture any relationships between metrics and EC. If there had been more data with major revisions, relations would be captured.

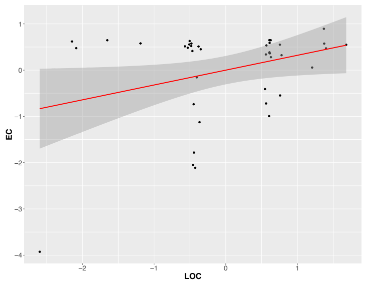
5.1.8 Models and Results for Dataset: rTorrent

Table 5.18 shows model assessments and their prediction performance on Dataset rTorrent. Results show that among the one-variable models (Model#1-#4), all the proposed models are significant; however, their R^2 values were very low when comparied with one-variable models on dataset DB2 and MYSQL. Model#1 had the highest R^2 followed by Model#3 and Model#4. Regarding prediction performance, all one-variable models had low prediction performance (PRED(25)) results. Regarding RMSE and MAE, Model#3 and Model#4 had medium accuracy, followed by Model#1. Model#5, as the only two-variable model, is also significant with medium accuracy (MAE being 0.49). However, this model had the lowest R^2 among the all

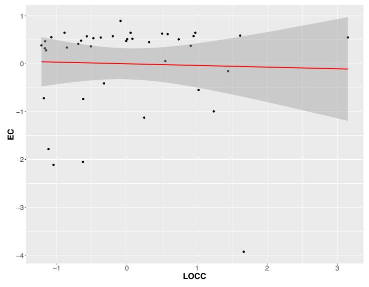
Table 5.18: Prediction performance of models on dataset rTorrent.

Model	Predictor Variables	p-value	R-squared	MAE	RMSE	PRED(25)
1	LOC	0.0075 **	0.390	0.509	0.775	0.402
2	LOCC	0.25	0.064	0.472	1.136	0.344
3	CC	0.0072**	0.312	0.481	0.577	0.364
4	MCC	0.0065 **	0.189	0.464	0.507	0.403
5	LOCC, CC	0.021*	0.170	0.489	0.879	0.349
Significance: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 0						

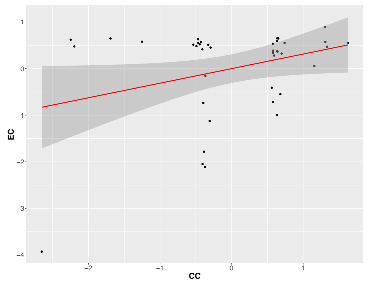
models. Similar to MYISAM-InnoDB results, the results suggest EC-MCC relation. Additionally, similar to DB2 and MyISAM, EC-LOC relation may also be suggested. Figures 5.17 (a-e) show rTorrent regression models graphs. They also support model performance results.



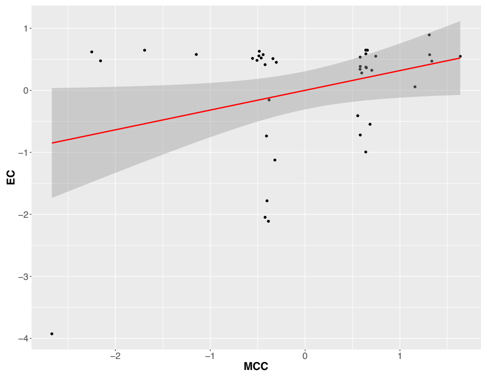
(a) Model#1



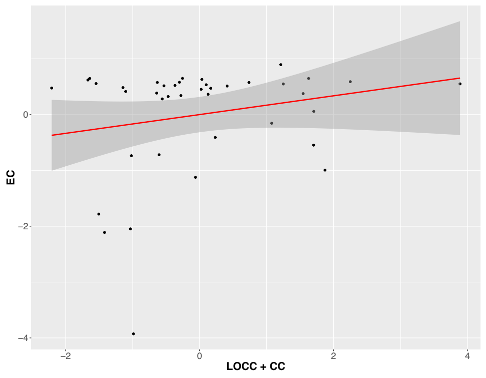
(b) Model#2



(c) Model#3



(d) Model#4



(e) Model#5

Figure 5.17: Regression model graphs for Firefox; the grey area show the 95% confidence intervals.

In summary, by reviewing R^2, accuracy and performance results, among the one-variable models, Model#1 and Model#4 achieved the highest performance on rTorrent dataset.

Checking Regression Assumptions: rTorrent

Assumption 1: The parameters of the linear regression model are numeric and linear (Table 4.31) all p-values are close to 0.

Assumption 2: The mean of the residuals are close to 0 (Model#1=2.9e-17, Model#2 =6.3e-17, Model#3= 3.4e-17, Model#4=9.7e-17, Model#5=1.4e-17).

Assumption 3: For two-variable models, multicollinearity must be sufficiently low between the predictors. If Variance Inflation Factor (VIF)>10, there is an indication for multicollinearity to be present. Model#5 VIF=1.03 smaller than 10, thus this assumption holds true for the Model#5.

Assumption 4: The variance in the predictors must be larger than 0. All the variance in the predictors is much larger than 0, thus this assumption holds true for all the predictors (LOC=2.2e+8, LOCC=4.4e+7, CC=6.7e+6, MCC=6.2e+6).

Assumption 5: The residuals must be normally or approximately normally distributed. Figure 5.18 (a-e) show that only Model#1 residuals follow sufficiently normal distribution.

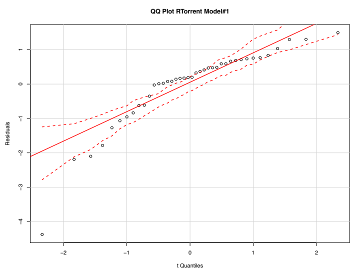
To support the graphical method of normality check, the Shapiro-Wilk test was applied to the rTorrent dataset. Results are presented in Table 5.17.

Table 5.19: rTorrent residuals Shapiro-Wilk test results.

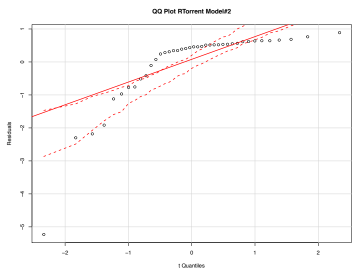
Model	Predictor Variables	W	p-value
1	LOC	0.882	0.001
2	LOCC	0.702	1.068e-07
3	CC	0.877	0.0004
4	MCC	0.880	0.0005
5	LOC, CC	0.864	0.0002

Table 5.19 shows that all models residuals *p-value* are smaller than 0.05, only *W* statistics of Model#1 is closer to one.

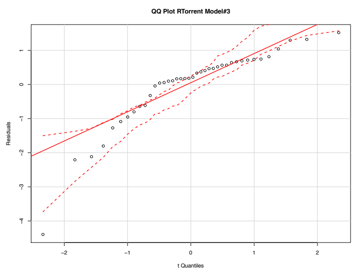
Table 5.18 shows that Model#3, Model#4 and Model#5 are significant (*p-value <0.05*). On the other hand, their residuals do not follow normal distribution. Additionally, these models have medium accuracy and low predictive performances (Table 5.18). In summary, except



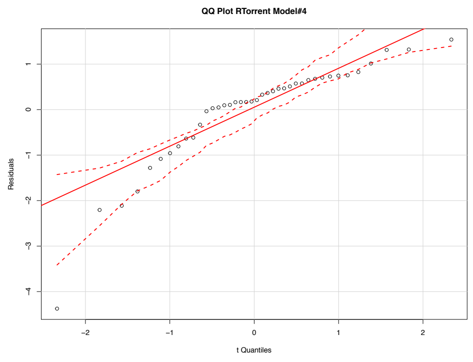
(a) Model#1



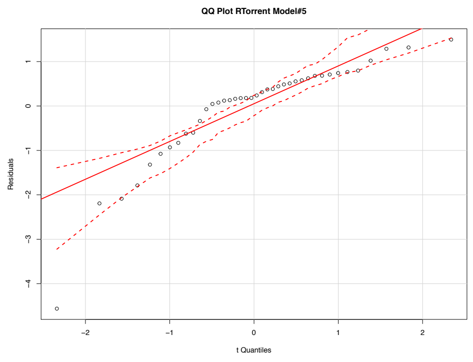
(b) Model#2



(c) Model#3



(d) Model#4



(e) Model#5

Figure 5.18: Residual plots for RTorrent models; the dashed lines show the 95% confidence intervals.

Model#1 ,with LOC as a predictor, the models are not successful for predicting EC on dataset rTorrent.

5.1.9 Regression Model Discussion

Dataset DB2 showed that both one-variable in which LOC is a predictor and two-variable models in which LOC, LOCC and CC are predictors, have high prediction performance with low error. Similar to DB2, in MYSQL (both MyISAM and InnoDB) both one-variable with LOC and two-variable with LOC and CC/MCC manage to have good prediction performance, but high error rates. Dataset Firefox results suggest only one-variable model using EC-LOC, EC-CC and EC-MCC relations. Similar to Firefox, in rTorrent dataset results suggest only the one-variable model with EC-LOC have an effective prediction performance. This study findings suggest that there is inconsistency in how predictive variables affect the energy consumption in different datasets.

5.2 Cross-Product Energy Consumption Prediction

As explained in Chapter 3, after assessing the models within products, they were evaluated in cross-product.

Intuitively, it could be argued that there is no issue in comparing within-project versus cross-project studies, because of the common belief that within-project data is always better than cross-project data. However, in the domain of effort estimation and defect prediction, it turns out that this intuition has contradictory results [112, 96]. Turhan [188] expressed one reason for the variance in the results reported in within-project and in cross-project studies is that there was an ambiguity of the effort estimation features [188] used in the studies. In order to clarify the ambiguity, Turhan et al [188, 189] adopted the most commonly used static code features and showed that models used on cross-company data performance were close to within-company data performance.

5.2.1 Cross-Product Energy Consumption Prediction (CPECP) Analysis

In the CPECP analysis, datasets from the five products given in Chapter 4 were used for all possible combinations to determine whether or not cross-product prediction works for EC prediction. Two additional datasets were built out of existing datasets. One dataset was formed by pooling MyISAM and InnoDB datasets and was named "MYSQL". The other dataset was formed by pooling all product datasets and was named "TOTAL". When the Total dataset was used as testing dataset in CPECP, within product dataset was not included in the pooling. For example, when we conducted CPECP on Firefox, the data from the remaining products (MyISAM + InnoDB + Vuze + rTorrent) was used to test a prediction model.Illustration and component of cross-product analysis is given in Chapter 3, Figure 3.3.

As explained previously, the modelling assessment steps for the CPECP analysis was adopted from Alpaydin [7] and Montgomery [132]. The same methodology was used in section 5.1.1.

The general steps are:

- *Aim:* to determine whether using cross-product data is beneficial for constructing energy consumption prediction and to identify the conditions under which cross-product data should be preferred to within-company data. Data collection and processing are explained in Chapter 4.
- *Selecting the response variable:* Response variable is energy consumption.
- *Choosing factors and levels:* The factors are input variables, standardization and different datasets with fixed learning algorithm. The input variables are software code metrics,

LOC, LOCC, CC and MCC. Standardization technique is z-score standardization and prediction is based on linear regression. More details are given in the following paragraphs.

- *Choosing design:* Test sets were built from Cross-Product (CP) data. Then, the model learned from Within-Product (WP) data. WP data is a single product dataset and CP data is all the other product datasets. Replication is also important and replication number depends on the dataset size. In this study 100 replications were performed.
- *Performing the prediction:* Assessment is conducted using the R version 3.2.4 (2016-03-10) tool. Linear model provided by The R Stats Package [146] was used.
- *Analyzing the results:* Model fit and performance assessments are done using a variety of measures which are given in Chapter 2.

The CPECP analysis follows the R script-3 given in Appendix D.

From the model analysis in section 5.1 results, the models that have significance in 0.05 level and high predictive performance among the models for the each dataset were used for the CP analysis.

5.2.2 Cross-Product Energy Consumption Prediction Results

Table 5.20 to Table 5.29 presents RMSE and PRED(25) results obtained from the CP analysis with standard deviations in brackets. Moreover, a one-variable linear regression model with LOC predictor fit plots for all datasets is given in Figure 5.19 to 5.23.

Table 5.20: DB2 dataset cross-product energy consumption prediction RMSE results.

Model	**Predictor**	**WP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**
#	**Variables**	**db2**	**myisam**	**innodb**	**mysql**	**firefox**	**vuze**	**rtorrent**	**total**
1	LOC	0.292	1.833	1.637	1.440	0.981	1.382	1.584	1.201
3	CC	0.316	1.533	1.769	1.400	0.985	1.382	1.581	1.185
4	MCC	0.318	1.553	1.780	1.398	0.986	1.383	1.585	1.174
5	LOC, LOCC	0.284	1.809	1.539	1.414	0.995	1.374	1.522	1.244
6	LOCC, CC	0.297	1.497	1.726	1.750	0.984	1.334	1.501	1.223
7	LOCC, MCC	0.298	1.510	1.697	1.330	0.987	1.365	1.503	1.213

Table 5.21: DB2 dataset cross-product energy consumption prediction PRED(25) results.

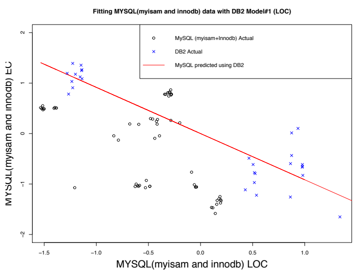
Model #	**Predictor Variables**	**WP DB2**	**CP myisam**	**CP innodb**	**CP mysql**	**CP firefox**	**CP vuze**	**CP rtorrent**	**CP total**
1	LOC	0.831	0.718	0.615	0.551	0.548	0.511	0.425	0.629
3	CC	0.792	0.717	0.627	0.518	0.534	0.509	0.415	0.620
4	MCC	0.766	0.718	0.628	0.520	0.530	0.501	0.406	0.618
5	LOC+LOCC	0.816	0.717	0.636	0.569	0.559	0.533	0.420	0.630
6	LOCC+CC	0.768	0.710	0.615	0.589	0.512	0.535	0.405	0.635
7	LOCC+MCC	0.818	0.716	0.628	0.600	0.515	0.540	0.414	0.640

Table 5.20 and 5.21 show DB2 models on CP analysis RMSE and PRED(25) results. RMSE results revealed that there is no model that performed close to WP DB2. All the CP RMSE results are approximately 5 times higher than WP DB2 RMSE results. Among all CP results, all models shows the lowest error performance on Firefox dataset followed by Total dataset, which contains all datasets except DB2. This may suggest that bigger datasets decrease the accuracy of the model.

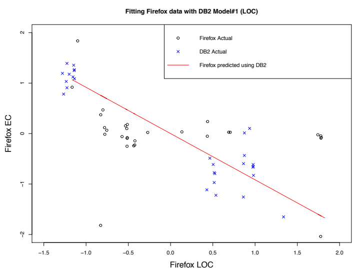
PRED(25) results show that (Table 5.21), in most of the datasets, Model#1 with LOC predictor shows higher prediction performance, followed by Model#6 with LOCC and CC. Prediction performance of MyISAM dataset has the closest performance to WP DB@ dataset performance, followed by Total dataset and InnoDB dataset. All the models show around 71% prediction performance on the MyISAM dataset, followed by Total with 63% and InnoDB with 62%. In summary, the analyzed models show around 30% lower prediction performance on CP than on WP DB2. Among the one-variable models, Model#1 has the highest prediction performance on WP with 83% and on all CP datasets with 57%. Model#6 has the highest performance among the two-variable models on WP with 77% and on all CP datasets with 56%. Model#1 fit on WP DB2, CP Firefox and CP Total dataset are also given in Figure 5.19 (a-c).

Considering CP prediction results, all CP datasets show low accuracy (high RMSE), but similar prediction performance, especially CP MyISAM, CP Total and CP InoDB. This may suggest that products in the same domain show similar prediction performance. Moreover, the success of cross-product energy consumption prediction is largely data-size driven.

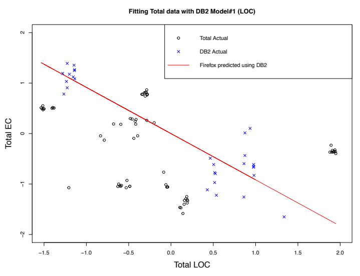
From Table 5.22, MyISAM models on CP RMSE results revealed that there is no model



(a) Fitting MYSQL data with DB2 Model#1



(b) Fitting Firefox data with DB2 Model#1



(c) Fitting Total data with DB2 Model#1

Figure 5.19: Fitting MYSQL, Firefox and TOTAL data with DB2 Model#1 (LOC).

Table 5.22: MyISAM dataset Cross-product energy consumption prediction RMSE results.

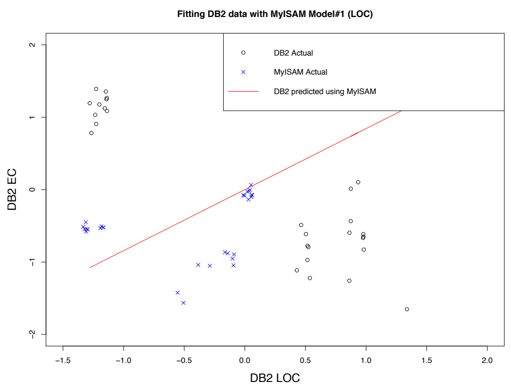
Model #	**Predictor Variables**	**WP myisam**	**CP db2**	**CP innodb**	**CP firefox**	**CP vuze**	**CP rtorrent**	**CP total**
1	LOC	0.560	1.771	0.933	1.414	1.285	1.060	1.373
5	LOC, CC	0.457	1.620	1.120	1.347	1.201	0.998	1.269
6	LOC, MCC	0.451	1.629	1.162	1.380	1.205	1.100	1.274

Table 5.23: MyISAM cross-product energy consumption prediction PRED(25) results.

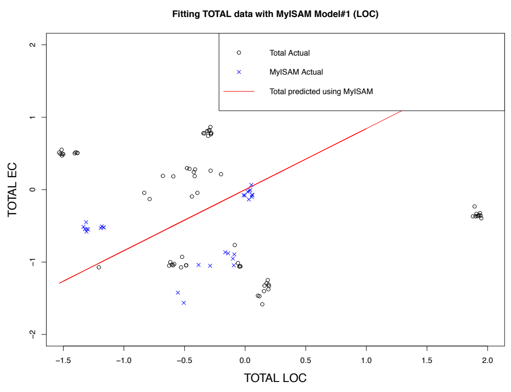
Model #	**Predictor Variables**	**WP myisam**	**CP db2**	**CP innodb**	**CP firefox**	**CP vuze**	**CP rtorrent**	**CP total**
1	LOC	0.817	0.528	0.632	0.548	0.576	0.305	0.791
5	LOC, CC	0.823	0.551	0.681	0.617	0.554	0.425	0.796
6	LOC, MCC	0.843	0.603	0.675	0.517	0.551	0.381	0.787

that performs closer to WP MyISAM. Among all CP results, all models behave differently. Model#5 and Model#6 have 5%-10% higher accuracy than Model#1. Regarding PRED(25), all the models have high performance on Total CP with 79% followed by InnoDB with 66% and Firefox 55%. When comparing all the models, Model#5 shows the highest prediction performance on CP Total with 80% and CP innodb with 65%. Only Total dataset showed close performance result to the WP. Summarizing the MyISAM CPECP results, it can be said that all CP show low accuracy and medium to low prediction performance. In contrast to DB2 results, the same domain models do not lead to accurate predictions.

As an example, Model#1 fit plots are given in Figure 5.20 (a-b).



(a) Fitting DB2 data with MyISAM Model#1



(b) Fitting Total data with MyISAM Model#1

Figure 5.20: Fitting DB2 and TOTAL data with MyISAM Model#1 (LOC).

Table 5.24: InnoDB dataset cross-product energy consumption prediction RMSE results.

Model #	Predictor Variables	WP innodb	CP db2	CP myisam	CP firefox	CP vuze	CP rtorrent	CP total
1	LOC	0.574	1.379	0.643	1.153	1.089	0.944	1.130
3	CC	0.659	1.677	1.011	1.446	1.236	1.033	1.309
4	MCC	0.622	1.680	1.013	1.456	1.234	1.027	1.312
5	LOC, CC	0.580	1.614	1.172	1.416	1.202	1.012	1.275
6	LOC, MCC	0.686	1.616	1.163	1.425	1.199	1.006	1.279

Table 5.24 and 5.25 present InnoDB models on CP analysis RMSE and PRED(25) results. From Table 5.24 similar to other CP results, there is no model on CP that performs closer to WP InnoDB. All CP results have 2 times lower accuracy than the WP results. Model #1 with LOC predictor has the highest accuracy across all CP results, followed by Model#5 and Model#6. Model #1 performed at 10%-23% higher accuracy than all the other models on CP.

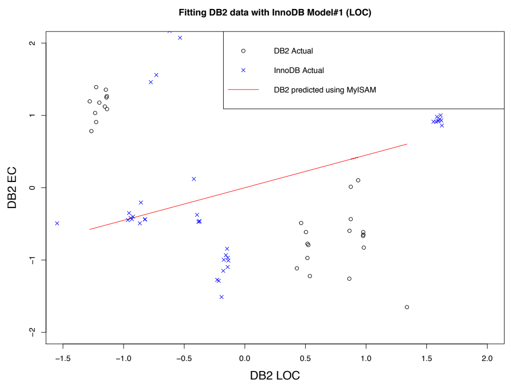
Regarding PRED(25), similar to MyISAM, all the InnoDB models performed well on CP Total, followed by CP MyISAM. Unlike the RMSE results, the PRED(25) results revealed that Model#6 has the highest performance among all CP. It showed 80% on WP InnoDB, followed

Table 5.25: InnoDB dataset cross-product energy consumption prediction PRED(25) results.

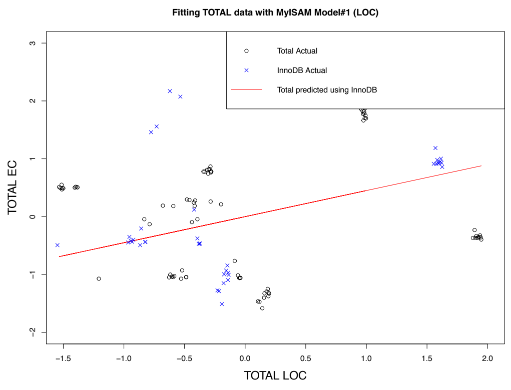
Model #	**Predictor Variables**	**WP innodb**	**CP db2**	**CP myisam**	**CP firefox**	**CP vuze**	**CP rtorrent**	**CP total**
1	LOC	0.747	0.532	0.718	0.547	0.511	0.555	0.756
3	CC	0.791	0.570	0.675	0.476	0.509	0.378	0.744
4	MCC	0.801	0.540	0.726	0.505	0.562	0.385	0.776
5	LOC, CC	0.850	0.541	0.716	0.516	0.544	0.327	0.783
6	LOC, MCC	0.852	0.593	0.728	0.605	0.604	0.432	0.800

by 80% on CP Total, and 73% on CP MyISAM. As a summary, similar to MyISAM CPECP results, all CP show low accuracy and medium to low prediction performance. Moreover, the same domain models do not lead to accurate predictions.

As a model fit example, Model#1 fit on WP InnoDB, CP DB2 and CP Total datasets are given in Figure 5.21 (a-c).



(a) Fitting DB2 data with InnoDB Model#1



(b) Fitting Total data with InnoDB Model#1

Figure 5.21: Fitting DB2 and TOTAL data with innoDB Model#1 (LOC).

From Table 5.26, WP Firefox models on CP analysis, except on CP DB2, all models show low accuracy on CP datasets. Models are on CP DB2 have 35% higher accuracy than on WP Firefox. Other CP results are 1.25 times higher than WP Firefox.

Regarding PRED(25) results, all models have 36% higher performance on CP MyISAM, 23% higher on CP InnoDB and 38% higher on MYSQL than on WP Firefox, followed by 10%

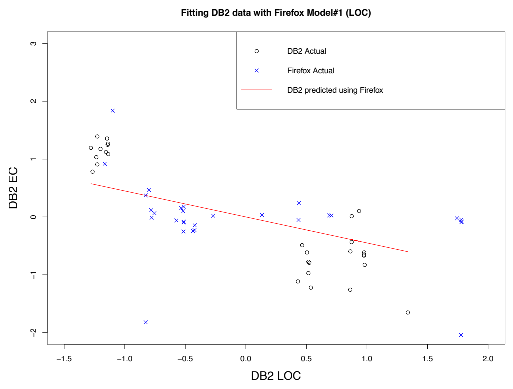
Table 5.26: Firefox dataset cross-product energy consumption prediction RMSE results.

Model #	**Predictor Variables**	**WP firefox**	**CP db2**	**CP myisam**	**CP innodb**	**CP mysql**	**CP vuze**	**CP rtorrent**	**CP total**
1	LOC	0.887	0.606	1.355	1.220	1.114	1.079	1.205	1.189
5	LOC, CC	0.921	0.604	1.346	1.266	1.105	1.081	1.207	1.192
6	LOC, MCC	0.897	0.602	1.371	1.255	1.111	1.079	1.206	1.189

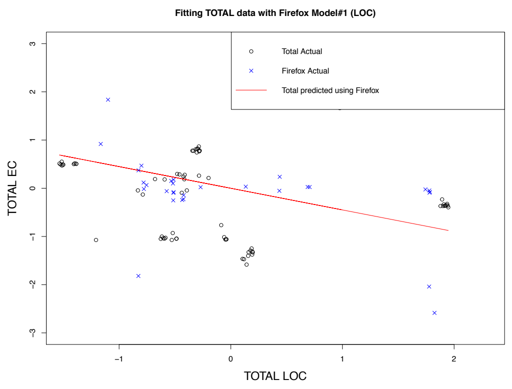
Table 5.27: Firefox cross-product energy consumption prediction PRED(25) results.

Model #	**Predictor Variables**	**WP firefox**	**CP db2**	**CP myisam**	**CP innodb**	**CP mysql**	**CP vuze**	**CP rtorrent**	**CP total**
1	LOC	0.530	0.553	0.734	0.612	0.784	0.530	0.306	0.586
5	LOC, CC	0.553	0.541	0.713	0.616	0.731	0.45	0.298	0.575
6	LOC, MCC	0.575	0.614	0.695	0.681	0.767	0.501	0.291	0.583

higher on CP Total and CP DB2. Among the models, Model#1 have higher performance than other models. As a model fit example, Model#1 fit on WP Firefox, on CP DB2 and on CP Total are given in Figure 5.22 (a-c). Figures also confirm that Model#1 performed well on DB2 and on Total datasets.



(a) Fitting DB2 data with Firefox Model#1



(b) Fitting Total data with Firefox Model#1

Figure 5.22: Fitting DB2 and TOTAL data with Firefox Model#1 (LOC).

Table 5.28: rTorrent cross-product energy consumption prediction RMSE results.

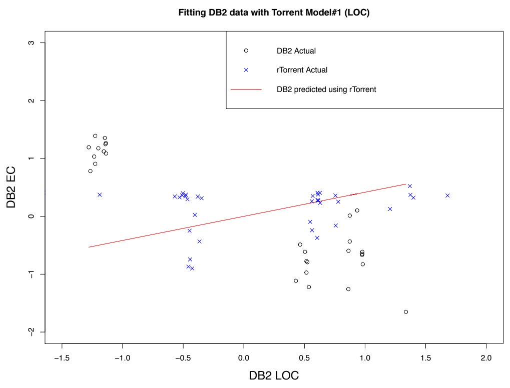
Model	**Predictor**	**WP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**
#	**Variables**	**torrent**	**db2**	**myisam**	**innodb**	**mysql**	**vuze**	**firefox**	**total**
1	LOC	0.835	1.277	0.739	0.890	1.024	1.042	1.160	1.147
3	CC	0.849	1.269	0.945	0.781	1.036	1.035	1.151	1.142
4	MCC	0.846	1.274	0.936	0.780	1.038	1.037	1.152	1.150
5	LOCC, CC	0.906	1.278	0.945	0.779	1.037	1.038	1.163	1.148

Table 5.29: rTorrent cross-product energy consumption prediction PRED(25) results.

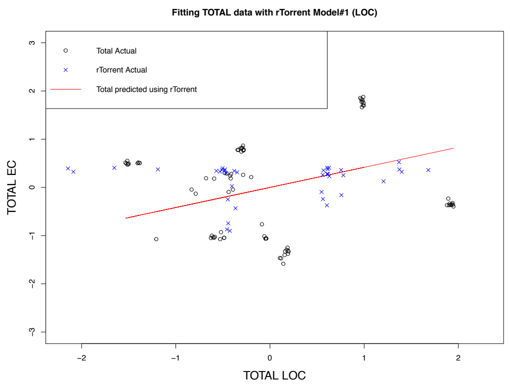
Model	**Predictor**	**WP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**
#	**Variables**	**torrent**	**db2**	**myisam**	**innodb**	**mysql**	**vuze**	**firefox**	**total**
1	LOC	0.402	0.594	0.719	0.641	0.475	0.535	0.527	0.551
3	CC	0.364	0.482	0.685	0.723	0.550	0.504	0.511	0.575
4	MCC	0.403	0.626	0.660	0.710	0.598	0.512	0.496	0.545
5	LOCC, CC	0.349	0.476	0.729	0.728	0.510	0.530	0.564	0.567

From Table 5.28 rTorrent models on CP analysis showed similar results to DB2, MyISAM and InnoDB CPECP analysis. All the models have 65% lower accuracy on CP DB2, 30% lower on CP Firefox and CP Vuze, and 47% lower on CP Total than on WP rTorrent. Model#1, Model#4 and Model#5 showed 15% higher accuracy on CP InnoDB than on WP rTottent. Similarly, Model#1 on CP MyISAM showed 12% lower accuracy than on WP rTorrent.

Regarding PRED(25) results from Table5.29, all models showed 75% higher prediction performance on CP MyISAM and CP InnoDB, 37% on CP DB2 and 33% on the rest of the CP datasets. On CP DB2, CP MyISAM and CP InnoDB, Model#3 and Model#1 have higher prediction performance than Model#4 and Model#5. In summary, similar to Firefox CPECP analysis results, rTorrent models have a better performance on CP than on WP. As a model fit example, Model#1 fit on WP rTorrent, CP DB2 and CP Total are given in Figure 5.23 (a-c). Figures also confirm that Model#1 performed better on CP Total than on CP DB2 datasets.



(a) Fitting DB2 data with rTorrent Model#1



(b) Fitting Total data with rTorrent Model#1

Figure 5.23: Fitting DB2 and TOTAL data with rTorrent Model#1 (LOC).

5.2.3 Cross-Product Energy Consumption Prediction Discussion

When considering CP prediction results, all CPs showed higher RMSEs and higher prediction performance (PRED(25)) than WPs. Only prediction performance (PRED(25)) results, which are close to WPs were obtained from CP Total datasets. This may indicate that the success of cross-product energy consumption prediction is largely data-size driven.

One interesting result obtained from cross-product analysis is that, models that come from the same domain had no effect on lowering error or increasing performance on CP. Having DB2 and MYSQL in the database domain, models on DB2 failed to predict MYSQL (MyISAM and InnoDB). The opposite direction also did not work out, both in terms of RMSE and PRED(25). Although DB2 and MYSQL are database products, they may not share the same components. DB2 is a "closed source" database product, whereas MYSQL is an "open-source" and their development organizations and their priorities are different. To summarize, DB2 and MYSQL share similar features and components because of their common domain; however, they are different in the process and tools used for their development. Same domain insights need further analysis and more data from different products from the same domain.

Unlike the same-domain cross-product results (DB2 and MYSQL), there were decreases in different-domain cross-product prediction performance (PRED(25)) and no improvement on accuracy results. Therefore, cross-product prediction does not work well. For example, DB2 models were able to predict Firefox. Similarly, MyISAM and InnoDB had better performance on Firefox than on other products. The reasons may be due to the product heterogeneity or domain source code characteristics. The results of cross-product prediction somewhat confirms the results of previous literature conclusions in the context of defect prediction. For example, Zimmermann et al. [213], Turhan et al. [189] and Canfora et al. [32] showed that cross-project prediction is not always successful.

When using all data as the Total dataset, the proposed models show better prediction performance than individual CP datasets. This may be explained with the increase in data size with different . When test data size increases cross-product prediction performance (PRED(25)) increases.

In summary, cross-product energy prediction is important for products with little or insufficient data to build prediction models. The results suggest that data size seemed to be crucial factors. However, on analyzed datasets there was no model that led to success in CPECP. In the case of different domains, it will not be possible to reuse the prediction model in same domain products. These results are limited to the datasets that were used in this study. In order to have more substantial outcome, the analysis will need to be replicated with more product datasets.

Chapter 6

Conclusions, Threats to Validity and Future Directions

This chapter presents a summary of the results and RQ evaluation followed by theoretical and practical contributions, and threats to validity. The chapter concludes with future directions.

6.1 Summary of Results

In applied sciences, study results should be beneficial to both academia and to industry. In the software engineering domain, study results (theoretical or experimental) should guide both software companies and researchers in academia. Therefore, while summarizing and generalizing the results, they need to be precisely presented.

The aim of this doctoral research is to predict the energy consumption of software products using a random effect based approach and classical linear-regression-based approach. Empirical findings on five different software products suggest that the effects of software code attributes and their relationships on software energy consumption are worthy of investigation. However, size, process and cyclomatic complexity metrics are not enough when designing software in order to effectively control its energy consumption. There are other factors that needs to be considered together with size and complexity. Although it seems that software releases are irrelevant, it has been found that product release have an effect on energy consumption. Performance of the mature product releases plays a role when comparing energy consumption among the product releases. In addition, cross-product analysis results suggest that cross-product data for energy consumption prediction may not feasible to use. More real world applications need to be analized with more product data. Similar to previous within-company cross-company predictive studies in software engineering (see Chapter 2), it is concluded that using products data in the same domain or different domain does not achieve accurate prediction models on energy consumption. Development process, data characteristics and domain need to be quantified, understood and evaluated before prediction models are built and used. Therefore, more studies with real world applications and with more products, different metrics, and more product characteristics are required.

6.1.1 Research Question Evaluation

This section summarizes the answers to the research question, according to the findings in Chapter 4 and Chapter 5.

6.1.2 RQ: How can we build a prediction model based on static code attributes to predict the energy consumption of software?

The aim of the study was to investigate the relationship between static code attributes and energy consumption of a software product. The findings of this study suggest relationships between energy consumption and size, energy consumption and complexity on analized datasets. Based on the correlation analysis, one- and two-variable random effect regression model and classic linear regression model are proposed.

Random-effect model is a successful modelling approach for grouped data structures accounting individual- and group-level variation in estimating group-level regression coefficients. This approach also has advantages regarding unobservable effects. Therefore, it is well suited to the DB2 dataset. Classic linear-regression is well suited to the problem since (i) it uses relationships between input and output variables and (ii) it can be built using continuous variables. Thus, both modelling approaches were analized on different datasets.

Two data collection approaches were adopted: observational data collection by measuring one software product and data extraction from software repositories. Results showed that the proposed random-effect one-variable model with LOC as a predictor had prediction performance upto 83% with less than 25% RMSE-MAE on the DB2 dataset. Simple linear regression model with LOC as a predictor had a performance around 75% with less than 50% RMSE-MAE on the MyISAM and InnoDB datasets. A proposed random effect two-variable model with LOC and LOCC had performance of 86% with less than 25% accuracy on DB2 dataset, while models with LOCC and CC/MCC had performances 81% with less than 25% accuracy on DB2 dataset. In MYSQL datasets, two-variable models with LOC and CC/MCC were able to predict 80-85% EC with 45% RMSE-MAE. One-variable models are able to predict only up to 57% EC with 60% RMSE in Firefox dataset. Similar to Firefox, in dataset rTorrent, one-variable models were only able to predict up to 40% with less than 50% accuracy. Finally, no model was able to be proposed on Vuze dataset.

In summary, the research findings suggest that cyclomatic complexity, size and code change metrics are not sufficient to consistently predict energy consumption.

After the prediction models built WP, model performances on CP data was investigated and the results were compared. The results of these experiments showed that WP models were able to predict EC with 80-85% (PRED(25)) performance, on the other hand, CP analysis showed 60-65% (PRED(25)) performance on predicting EC. While WP models were able to predict 80-85% of WP data, they were only able to predict only 50-60% of CP data. Moreover, WP models were able to predict 65-75% of a CP dataset, which is a pool of all the product‘s data. But, in all CP analysis, accuracy of the predictions were quite low. In summary, single CP data had no effect on increasing the prediction of EC, but pooled CP data showed close prediction performance to WP data. As stated in section 3.3.1, software reliability prediction guidelines suggest that a good model has a higher than 75% prediction performance. However, based on knowledge, there is no adopted guideline for energy consumption prediction performance. Therefore, only for the purpose of this study, higher than 70% would be adequate for performance.

6.2 Research Contributions

6.2.1 Theoretical and Methodological Contribution

The theoretical and methodological contributions of this dissertation are summarized as follows:

In this dissertation, a thorough analysis of a relationship of software metrics with energy consumption were conducted.

Recent research in software energy consumption have utilized correlations and only one of them used a linear model on one product [129]. Moreover, all of the research to date only investigated open source software products. This research contributes to the investigation with closed-source software product as well. Moreover, a methodology was suggested incorporating observational data collection, which includes energy consumption measurements and software code metrics extraction, with mining software repositories. Relationships were also investigated based on statistical tests strengthened with graphical analysis. Similar relationship analysis using code metrics were previously conducted in the literature [83]. In this dissertation, previous studies are improved upon considering complexity metrics and using various open source products and one closed source product.

A set of well-defined observational data collection and software repository mining followed by a regression analysis were performed during the model constructions. This allows other researchers to replicate the methodology. Reproducibility of empirical studies is a very important property in the software engineering domain. Unfortunately, there is a lack of publicly available datasets in software engineering, especially in the context of industrial energy data, as it is hard to collect. Therefore, constructing another dataset provides an important resource for software energy consumption studies. However, due to the confidentiality of IBM product, the dataset will not be available to public.

The study also investigated cross-product energy consumption prediction. Cross-product analysis is important for software projects with little or insufficient data to build prediction models. Although, there are a number of cross-project studies on software defect prediction and cost estimation, until now there has been no study in building energy consumption prediction models using cross-product data. To the best of our knowledge, this study is the first to use within-product and cross-product approaches in the software energy consumption prediction context.

Practical Implications

The aforementioned contributions also have practical implications, which are discussed as follows:

Energy efficiency and related measurements and tools are gaining increasing attention in software engineering practices. Monitoring and modelling energy consumption of software products and software intensive systems are yet to become mainstream in practice. Thus, it is necessary to investigate how to assess software energy consumption for stimulating the awareness of energy efficiency beginning in the early phases of the software development life cycle.

As discussed in Chapter 5, energy measurements were taken by an external device. Monitoring system consumption does not alter system state, hence, if one would like to conduct performance and measurements simultaneously, an external hardware device is safer to use. As explained, the process of incorporating an external measurement device is relatively straightforward: the energy measuring device has to be attached to the power of the system being tested and take the measurements before and after the test to compute the consumption for a given test. Note that the measurement process can be automated, as the measuring device provides programmatic access to the data. Although it is argued that measuring energy consumption directly from the incorporated external device is not an easy choice, it requires significant investments in terms of resources, specialized knowledge [104], and benchmark data. Predicting energy consumption from static code attributes would enable the software teams to re-factor the code before execution. The empirical study showed that energy consumption prediction is complicated. The findings suggest there are relationships between energy consumption and code attributes. However, size, process and complexity attributes are not strong predictors of energy consumption. It is important to note that, size and complexity metrics findings may be helpful in highlighting performance departures from design principles. In the explanatory analysis, version effect were identified and evaluated with the random effect model approach. Other factors which may effect on energy consumption must be quantified, understood and evaluated before prediction models are built and used.

It was observed that a product such as DB2 may be optimized by only changing one variable and hence the energy efficiency has nothing to do with the size or complexity of the product. In that case, static code attributes can not give any information and therefore the software company should look for collecting runtime metrics in order to predict the energy consumption.

As seen in the literature of predictive models in software engineering, there is no specific set of metrics that should be used in any prediction. It would be useful to extract a common set of metrics, which proved to be significant indicators of product energy consumption. Although, suggested relationships between static code attributes and energy consumption are promising, runtime metrics need to be collected to build a model with high predictive performance. For example, extract significant events and/or operational metrics (i.e. the number of queries executed by a system, the time spent in a particular function, etc.) at runtime. However, difficulty of collecting run time metrics, easily accessible static code attributes provide support for future investigation.

In order to assess proposed model performances, three different measures are presented: MAE, RMSE, and PRED(25). In practice; however, companies may use only a single measure. PRED(k) measure would be the most practical to use, since PRED(k) would evaluate the overall success of a model in terms of variance of its predictions. In order to use this measure, practitioners need to set their own threshold (k) for a model. For example, if they set their "k" as being 25 that means their model should predict at least 75% of data with 25% error or less.

Considering the CPECP analysis, the results showed that, overall, the CP prediction for single-product were significantly worse than the WP predictions for single-product. None of the models had higher accuracy than the WP. Thus, development companies need their own dataset (a single-product dataset) to build more accurate energy consumption prediction.

Overall, the findings provide insights to both practitioners and researchers with a foundation of knowledge that static code attributes may give some insights, but they would not be the sole predictors of energy consumption of software products. Developers may benefits from finding when assessing the energy footprint and performance of their product with the help of software metrics which would support developing green software; researchers may use the findings for building foundation for models predicting energy consumption.

6.3 Threats to Validity

In this section, the possible threats to validity and their mitigation are presented. There are different types of validity when trying to develop a model to understand experiments in complex settings. The most commonly used and general types are *internal validity* and *external validity* [42]. Furthermore, *construct validity* is defined in order to consider the generalizability of experiments across different settings. Cook and Campbell [30, 31] suggest that, *statistical conclusion validity* should be defined as a special case of internal validity in order to consider sources of random error and the appropriate use of statistics and statistical tests. Therefore, in this research, possible threats are underlined to internal, external, construct and conclusion validities.

6.3.1 Construct Validity

Threats to construct validity affect the degree to which the measures capture the concepts of interest in the experiment [163, 45]. The following construct validity threats are identified and their mitigations are explained.

- *Metric selection.* One limitation is the selection of the metrics for the the study. To mitigate this threat, metrics were chosen from the set of metrics that can be calculated from direct measurements and extractions. Moreover, the metrics represent the most commonly used software static code attributes, which are size, complexity and code change for estimation and prediction studies. Static code attributes have very good reasons to be used in estimation and prediction studies: they are easy to collect and widely used [123].

- *Model selection.* It is stated that relationships between metrics are a major part of the research and these relationships should be identified in order to construct an effective model. Regression models suggest relations between metrics representing size, complexity and processes. Moreover, all the variables in this study are continuous variables. Regression is simple and commonly used in previous studies. As for grouped data structure, fixed/random effect model approach is commonly used approach. Therefore, classic regression and fixed/random effect regression are the most well-known choices for modelling energy consumption on this study datasets.

- *Performance measure selection.* Another limitation is selection of performance measures for accessing the models. To overcome threats of selecting the right performance measures, various, most commonly used, performance measures are chosen. These measures have been used in effort estimation studies (e.g., [95, 14, 152]) and in cross-project effort prediction studies (e.g.,[127]). Therefore, MAE, RMSE and PRED(k) values were reported.

6.3.2 Conclusion Validity

Conclusion validity concerns the relation between the treatment and the outcome [31]. The following threats to conclusion validity are identified:

- *Data heterogeneity.* In the literature it is stated that the level of heterogeneity of the data may vary and influence WP models' performance (e.g., [119, 127]). To mitigate the data heterogeneity threat, z-score standardization was performed. A similar approach was applied by Minku and Yao [127] and Canfora et al. [32].

- *Statistical tests.* Statistical significance of relationships of metrics were checked using Pearson's correlation test, which is relevant for this study‘s variables. Significance test and scatter plots were are also used due to independence of a relation between two variables from distributions of data. Moreover, considering linear regression assumptions, all the assumptions were checked using various tests and residual plots. Thus, selecting appropriate tests were considered during the experiments in order to validate assumptions implied by several tests.

- *Accuracy and consistency of measurements.* In order to maintain accuracy of results and reduce measurement error, all the observational data collection tests were repeated 10 times. To ensure consistency of measurement and workload between two databased products, the same standard benchmark TPC-H workloads were run in data collection of DB2.

6.3.3 Internal Validity

Threats to internal validity affect the interpretation of the results. Internal validity is concerned with the uncontrolled factors that might affect the results of the experiments [97, 163]. The following threats to internal validity were identified:

- *Data collection.* Data collection is a challenging task in software engineering studies. In order to avoid bias during data collection, data collection guidelines for empirical studies were followed. For extraction required code metrics, the most commonly used code extraction tools were utilized. The results of the experiments and source code metrics were stored in a local computer and results and tests were automatically generated using R scripts. Furthermore, possible outliers in datasets, which may occur due to measurement error or miscalculation of metrics were investigated and removed.

- *Instrumentation.* Changes in measuring instrumentation or changes in observers used may produce changes in obtained measurements. To mitigate this threat, all the measurements regarding DB2 were conducted in the same test setup, as explained in Chapter 4. Regarding metric extractions, for all DB2 and MYSQL releases the same metric extraction tools were used. Similarly, for Firefox, Vuze and Torrent releases the same metric extraction tools were used.

- *EC measurement reliability.* Using open source EC measurements data may bring another treat to the internal validity. However, EC data of four products and various analysis that were done using these data have been published in peer-reviewed conferences and journals. Therefore, this threat were addressed.

- *Omitted variable bias.* Unobservable factors that simultaneously affect the outcome of the regression may create an omitted variable bias. All the variables that have measurements in this study were used. There may be other factors that may affect the results, however, there were no measurements of such factors. Fixed/random effect and/or instrumental variables can mitigate or eliminate the bias [49]. Therefore, the used fixed/random modelling approach for clustered data mitigates this potential threat as it introduces fixed/random effects.

- *Grouped data structure.* Grouped (clustered) structure of DB2 dataset was a major internal threat. Individual groups may affect the outcome of the prediction. Thus, each DB2 version group was analyzed individually. However. individual group analysis reduced the effective sample size, which, in turn affected the model results. In order to analyzed group effect on whole DB2 dataset without splitting the version groups, fixed/random effect modelling approach were used.

6.3.4 External Validity

External validity refers to the extent to which the results of the study can be disseminated to the outside world and be used more generally, rather than being limited to the scope of the study [97, 163].

- *Generalizability of results.* As described by Basili et al. [13] and Wieringa & Daneva [200] ware engineering studies suffer from the variability of the real world, and the generalization problem cannot be solved completely. As they indicate, to build a theory we need to generalize to a theoretical population and have adequate knowledge of the architectural similarity relation that defines the theoretical population.

Each product has unique characteristics, in terms of software processes, the product and the organization it belongs to. Thus, major results that influence software engineering practice rely on accumulation of evidence from many different products. Each individual research provides incremental knowledge, and collections of related research and reports provide both confirming and cumulative evidence [89].

Two database products, two Torrent trackers, and one web-browser were studied. The generalization to other database products, Torrent trackers, or other software products is, obviously, not possible. In this study, we do not aim build a theory, rather we would like to have a deeper understating of relationship between energy consumption and code attributes.

6.4 Future Directions

Software energy efficiency has gained importance in the software engineering community, but many challenges remain. The findings of this study provide basis for future research agenda on conducting more observational studies with different products and new sets of software metrics. Consequently, future aim is to build robust predictive models and integrate them into daily software development activities, so that software practitioners and decision makers may use such models in their decision making processes to define policies and contribute to environmental sustainability by decreasing their product footprint.

The next effort in energy prediction will be to develop benchmarks and benchmark data suitable for different domains (e.g., browser testing and streaming software testing) and to develop automated prediction and to design recommendations for software products. This will provide more experimentation and maturity to the area of energy efficient software design. Another future direction is to extract run time metrics and predict run time software energy consumption behavior.

Appendix A

Python Script for downloading source code from mercurial repository

Python Script-1

```
import csv
import requests, zipfile, io
import os
from urllib2 import urlopen, URLError, HTTPError
def dlfile(url):
# Open the url
try:
f = urlopen(url)
print "downloading " + url
# Open our local file for writing
with open(os.path.basename(url), "wb") as local file:
local file.write(f.read())
#handle errors
except HTTPError, e:
print "HTTP Error:", e.code, url
except URLError, e:
print "URL Error:", e.reason, url
reader = csv.reader(open("mozilla directories.csv", "rb"))
for row in reader:
row = ".join(row)
url = 'https://hg.mozilla.org/releases/mozilla-1.9.1/archive/' + row +'.↩
   tar.bz2'
print url
dlfile(url)
```

Appendix B

R Script for constructing linear regression model and performance evaluations

R Script-1

```r
db2<- read.csv("DB2_all_model.csv") #reading db2 data
# Now Let's write standardization function Z-score
standard_sd <- function (x) {return((x-mean(x))/(sd(x)))}
db2<- as.data.frame(lapply(db2, standard_sd)) #compute standardization
#Now lets run one-variable linear regression model
model_1<- lm(EC~LOC, db2)
summary(model1) #summary of model for p-value and R-squared
#Now doing Machine Learning and Calculating MAE, RMSE and Pred (25)
# 100 random replication for MAE
rep<-replicate(100,
  {
    db2_1 <- sample(nrow(db2), floor(nrow(db2)*0.7)) # train-test (70-30)
    train <- db2[db2_1,]
    test  <- db2[-db2_1,]
    model1 <- lm(EC ~ LOC, data=train)  # build the model with LOC
    predict<- predict(model1, test) # predict EC on test dataset
    errors <- predict - test$EC
    mae<-mean(abs(errors))
    mae
  })
mean(rep) #calculating MAE mean of 100 replication
# 100 random replication for RMSE
rep<-replicate(100,
  {
    db2_1 <- sample(nrow(db2), floor(nrow(db2)*0.7)) # train-test (70-30)
    train <- db2[db2_1,]
    test  <- db2[-db2_1,]
    model1 <- lm(EC ~ LOC, data=train)  # build the model with LOC
    predict<- predict(model1, test) # predict EC on test dataset
    RMSE<- RMSE(predict, test$EC) # calculating RMSE on test dataset
    RMSE})
mean(rep) #calculating RMSE mean of 100 replication
# 100 random replication for Pred(25)
rep<-replicate(100,
  {
    db2_1 <- sample(nrow(db2), floor(nrow(db2)*0.7)) #train-test (70-30)
    train <- db2[db2_1,]
    test  <- db2[-db2_1,]
    model1 <- lm(EC ~ LOC, data=train)  # build the model with LOC
    predict<- predict(model1, test) # predict EC on test dataset
    errors <- predict - test$EC
    rel_change <- 1 - ((test$EC - abs(errors)) / test$EC)
    table(rel_change<0.25)["TRUE"] / nrow(test)
  })
mean(rep) #calculating Pred(25) mean of 100 replication
```

Appendix C

R Script for constructing random effect linear model.

R Script-2

```r
install.packages("lme4") #linear Mixed Effect model package
install.packages("lmerTest") #Tests in Linear Mixed Effects Models
db2<- read.csv("DB2_all_model.csv") #reading db2 data
# Now Let's write standardization function Z-score
standard_sd <- function (x) {return((x-mean(x))/(sd(x)))}
db2<- as.data.frame(lapply(db2, standard_sd)) #compute standardization
#Now lets run one-variable random effect (version) linear regression model
model_1<- lmer(EC ~ LOC + (1+LOC|version), REML=FALSE, data=db2)
summary(model1) #summary of model
ranef(model1) #random effect coefficients
fixef(model1)  #fixed effect coefficients
#Now doing Machine Learning and Calculating MAE, RMSE and Pred (25)
# 100 random replication for MAE
rep<-replicate(100,
  {db2_1 <- sample(nrow(db2), floor(nrow(db2)*0.7)) # train-test (70-30)
    train <- db2[db2_1,]
    test  <- db2[-db2_1,]
    model_1<- lmer(EC ~ LOC + (1+LOC|version), REML=FALSE, data=train)
    predict<- predict(model1, test, re.form=NA) # predict EC on test ↩
       dataset
    errors <- predict - test$EC
    mae<-mean(abs(errors))
    mae})
mean(rep) #calculating MAE mean of 100 replication
# 100 random replication for RMSE
rep<-replicate(100,
  {db2_1 <- sample(nrow(db2), floor(nrow(db2)*0.7)) # train-test (70-30)
    train <- db2[db2_1,]
    test  <- db2[-db2_1,]
    model1 <- lm(EC ~ LOC, data=train)  # build the model with LOC
    predict<- predict(model1, test) # predict EC on test dataset
    RMSE<- RMSE(predict, test$EC) #calculating RMSE on test dataset
    RMSE})
mean(rep) #calculating RMSE mean of 100 replication
# 100 random replication for Pred(25)
rep<-replicate(100,
  {db2_1 <- sample(nrow(db2), floor(nrow(db2)*0.7)) #train-test (70-30)
    train <- db2[db2_1,]
    test  <- db2[-db2_1,]
    model_1<- lmer(EC ~ LOC + (1+LOC|version), REML=FALSE, data=train)
    predict<- predict(model1, test) # predict EC on test dataset
    errors <- predict - test$EC
    rel_change <- 1 - ((test$EC - abs(errors)) / test$EC)
    table(rel_change<0.25)["TRUE"] / nrow(test)})
mean(rep) #calculating Pred(25) mean of 100 replication
```

Appendix D

R Script for cross-product analysis.

R Script-3

```
db2<- read.csv("DB2_all_model.csv") #reading db2 data
mysql<-read.csv("MYSQL_all_model.csv") #reading mysql data
# Now Let's write standardization function Z-score
standard_sd <- function (x) {return((x-mean(x))/(sd(x)))}
db2<- as.data.frame(lapply(db2, standard_sd)) #compute standardization
mysql<- as.data.frame(lapply(mysql, standard_sd)) #compute standardization
#Now lets run one-variable random effect (version) linear regression model
model_1<- lmer(EC ~ LOC + (1+LOC|version), data=db2, REML = FALSE)
#Now doing Cross-product analysis and Calculating MAE, RMSE and Pred (25)
# 100 random replication for MAE
rep<-replicate(100,
  {
    train <- db2
    test  <- mysql
    model_1<- lmer(EC ~ LOC + (1+LOC|version), data=train, REML = FALSE) #↩
        model on train dataset
    predict<- predict(model1, test, re.form=NA) # predict EC on test ↩
       dataset
    errors <- predict - test$EC
    mae<-mean(abs(errors))
    mae})
mean(rep) #calculating MAE mean of 100 replication
# 100 random replication for RMSE
rep<-replicate(100,
  {
    train <- db2
    test  <- mysql
    model_1<- lmer(EC ~ LOC + (1+LOC|version), data=train, REML = FALSE)
    predict<- predict(model1, test, re.form=NA) # predict EC on test ↩
       dataset
    RMSE<- RMSE(predict, test$EC) #calculating RMSE on test dataset
    RMSE})
mean(rep) #calculating RMSE mean of 100 replication
# 100 random replication for Pred(25)
rep<-replicate(100,
  {
    train <- db2
    test  <- mysql
    model_1<- lmer(EC ~ LOC + (1+LOC|version), data=train, REML = FALSE)
    predict<- predict(model1, test, re.form=NA) # predict EC on test ↩
       dataset
    errors <- predict - test$EC
    rel_change <- 1 - ((test$EC - abs(errors)) / test$EC)
    table(rel_change<0.25)["TRUE"] / nrow(test)})
mean(rep) #calculating Pred(25) mean of 100 replication
```

References

[1] Marc Aerts, Geert Molenberghs, Louise M Ryan, and Helena Geys. *Topics in modelling of clustered data*. CRC Press, 2002.

[2] International Energy Agency. Appliances and equipment cross sectoral energy efficiency indicators, (accessed March, 2016). `http://iea.org/topics/energyefficiency/`.

[3] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[4] Alan Agresti, James G Booth, James P Hobert, and Brian Caffo. 2. random-effects modeling of categorical response data. *Sociological Methodology*, 30(1):27–80, 2000.

[5] Zainab Al-Zanbouri. Database engines: Evolution of greeness. Master's thesis, Ryerson University, 2015.

[6] Allan J. Albrecht and John E Gaffney. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE transactions on software engineering*, 1(6):639–648, 1983.

[7] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[8] Nadine Amsel and Bill Tomlinson. Green tracker: a tool for estimating the energy consumption of software. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 3337–3342. ACM, 2010.

[9] Inc. Azureus Software. Vuze project, (accessed September, 2016). `http://www.vuze.com`.

[10] Victor R Basili. Quantitative evaluation of software methodology. Technical report, DTIC Document, 1985.

[11] Victor R Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 1–12. Springer, 1993.

[12] Victor R Basili. The role of experimentation in software engineering: past, current, and future. In *Proceedings of the 18th international conference on Software engineering*, pages 442–449. IEEE Computer Society, 1996.

[13] Victor R Basili, Forrest Shull, and Filippo Lanubile. Building knowledge through families of experiments. *Software Engineering, IEEE Transactions on*, 25(4):456–473, 1999.

[14] Bilge Baskeles, Burak Turhan, and Ayse Bener. Software effort estimation using machine learning methods. In *Computer and information sciences, 2007. iscis 2007. 22nd international symposium on*, pages 1–6. IEEE, 2007.

[15] Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker. Fitting linear mixed-effects models using lme4. *arXiv preprint arXiv:1406.5823*, 2014.

[16] Ayse Bener, A Misirli, Bora Caglayan, Ekrem Kocaguneli, and Gul Calikli. Lessons learned from software analytics in practice. *The art and science of analyzing software data, 1st edn. Elsevier, Waltham*, pages 453–489, 2015.

[17] Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems. *Computer*, 11:68–74, 2004.

[18] Serdar Biçer, Ayşe Başar Bener, and Bora Çağlayan. Defect prediction using social network analysis on issue repositories. In *Proceedings of the 2011 International Conference on Software and Systems Process*, pages 63–71. ACM, 2011.

[19] Christian Bird, Tim Menzies, and Thomas Zimmermann. *The Art and Science of Analyzing Software Data*. Elsevier, 2015.

[20] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[21] Barry W Boehm, Ray Madachy, Bert Steece, et al. *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR, 2000.

[22] Barry W Boehm and Ricardo Valerdi. Achievements and challenges in cocomo-based software resource estimation. *IEEE software*, 25(5), 2008.

[23] Peter Boncz, Thomas Neumann, and Orri Erling. Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 61–76. Springer, 2013.

[24] Grady Booch. Software engineering in practice keynote: The future of software engineering. In *ICSE15: Proc. of the 37th Intl. Conf. on Software Engineering*, 2015. Available at: https://www.youtube.com/watch?v=h1TGJJ-F-fE.

[25] David Brooks, Vivek Tiwari, and Margaret Martonosi. *Wattch: a framework for architectural-level power analysis and optimizations*, volume 28–2. ACM, 2000.

[26] Göran Broström and Henrik Holmberg. Generalized linear models with clustered data: Fixed and random effects models. *Computational Statistics & Data Analysis*, 55(12):3123–3134, 2011.

[27] Christian Bunse, Zur Schwedenschanze, and Sebastian Stiemer. On the energy consumption of design patterns. In *Proceedings of the 2nd Workshop EASED@ BUIS Energy Aware Software-Engineering and Development*, pages 7–8. Citeseer, 2013.

[28] Bora Caglayan, Ayse Bener, and Stefan Koch. Merits of using repository metrics in defect prediction for open source projects. In *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 31–36. IEEE Computer Society, 2009.

[29] A Colin Cameron and Douglas L Miller. A practitioners guide to cluster-robust inference. *Journal of Human Resources*, 50(2):317–372, 2015.

[30] Donald T Campbell and Julian C Stanley. *Experimental and Quasi-Exprimental Designs for Research*, volume 4. Rand McNally, 1971.

[31] Donald T Campbell and Julian C Stanley. *Experimental and quasi-experimental designs for research.* Ravenio Books, 2015.

[32] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Multi-objective cross-project defect prediction. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 252–261. IEEE, 2013.

[33] Eugenio Capra, Giulia Formenti, Chiara Francalanci, and Stefano Gallazzi. The impact of mis software on it energy consumption. In *ECIS*, 2010.

[34] Eugenio Capra, Chiara Francalanci, and Sandra A Slaughter. Is software green? application development environments and energy efficiency in open source applications. *Information and Software Technology*, 54(1):60–71, 2012.

[35] VJ Carey and You-Gan Wang. Mixed-effects models in s and s-plus, 2001.

[36] IBM Knowledge Center. Db2 for linux, unix, and windows, db2 product editions and db2 offerings, (accessed September, 2015). `http://www.ibm.com/support/knowledgecenter/SSEPGG`.

[37] John M Chambers. *Graphical methods for data analysis*. Champman and Hall, 1983.

[38] Ruzanna Chitchyan, Christoph Becker, Stefanie Betz, Leticia Duboc, Birgit Penzenstadler, Norbert Seyff, and Colin C Venters. Sustainability design in requirements engineering: state of practice. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 533–542. ACM, 2016.

[39] Raul F Chong, Xiaomei Wang, Michael Dang, and Dwaine Snow. *Understanding DB2: Learning Visually with Examples*. Pearson Education, 2007.

[40] CLOC. Count lines of code tool version 1.72, (accessed December, 2016). `https://github.com/AlDanial/cloc`.

[41] Samuel Daniel Conte, Hubert E Dunsmore, and Vincent Y Shen. *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., 1986.

[42] Thomas D Cook, Donald Thomas Campbell, and Arles Day. *Quasi-experimentation: Design & analysis issues for field settings*, volume 351. Houghton Mifflin Boston, 1979.

[43] Vlad Coroama and Lorenz M Hilty. Energy consumed vs. energy saved by ict–a closer look. In *Environmental Informatics and Industrial Environmental Protection: Concepts, Methods and Tools, 23rd International Conference on Informatics for Environmental Protection, Berlin*, pages 353–361, 2009.

[44] Transaction Processing Performance Council. Tpc benchmark-h, decision support, standard specification, (accessed August, 2015). `http://www.tpc.org/tpch/spec/tpch2.14.4.pdf`.

[45] John W Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2013.

[46] Yves Croissant, Giovanni Millo, et al. Panel data econometrics in r: The plm package. *Journal of Statistical Software*, 27(2):1–43, 2008.

[47] Oxford Online Dictionary. A dictionary of physics, 2009. `http://www.oxfordreference.com/view/10.1093/acref/9780198714743.001.0001/acref-9780198714743`.

[48] Mian Dong and Lin Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 335–348. ACM, 2011.

[49] D Dranove. Practical regression: Introduction to endogeneity: Omitted variable bias, 2121.

[50] Marco DAmbros, Michele Lanza, and Romain Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577, 2012.

[51] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[52] Lorenz Erdmann, Lorenz Hilty, James Goodman, and Peter Arnfalk. *The future impact of ICTs on environmental sustainability.* European Commission, Joint Research Centre, 2004.

[53] Hakan Erdogmus. Essentials of software process. *IEEE software*, 25(4):4–7, 2008.

[54] EU2012directiveo. Directive 2012/19/eu of the european parliament and of the council of 4 july 2012 on waste electrical and electronic equipment (weee), (accessed January, 2016). `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32012L0019`.

[55] Yunsi Fei, Srivaths Ravi, Anand Raghunathan, and Niraj K Jha. Energy-optimizing source code transformations for operating system-driven embedded software. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(1):2, 2007.

[56] N Fenton, Martin Neil, and D Marquez. Using bayesian networks to predict software defects and reliability. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 222(4):701–712, 2008.

[57] Norman Fenton and James Bieman. *Software metrics: a rigorous and practical approach.* CRC Press, 2014.

[58] Steffen Fieuws, Geert Verbeke, and Geert Molenberghs. Random-effects models for multivariate repeated measures. *Statistical methods in medical research*, 16(5):387–397, 2007.

[59] Jason Flinn and M Satyanarayanan. Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review*, 34(2):13–14, 2000.

[60] Jason Flinn and Mahadev Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Transactions on Computer Systems (TOCS)*, 22(2):137–179, 2004.

[61] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

[62] Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevelhierarchical models*, volume 1. Cambridge University Press New York, NY, USA, 2007.

[63] Robert L Glass. The software-research crisis. *IEEE Software*, 11(6):42, 1994.

[64] Robert L. Glass, Iris Vessey, and Venkataraman Ramesh. Research in software engineering: an analysis of the literature. *Information and Software technology*, 44(8):491–506, 2002.

[65] Michael W Godfrey, Ahmed E Hassan, James Herbsleb, Gail C Murphy, Martin Robillard, Prem Devanbu, Audris Mockus, Dewayne E Perry, and David Notkin. Future of mining software archives: A roundtable. *IEEE Software*, 26(1):67–70, 2009.

[66] William H Greene. *Econometric analysis*. Pearson Education India, 2003.

[67] A Gupta, T Zimmermann, C Bird, N Naggapan, T Bhat, and S Emran. Energy consumption in windows phone. Technical report, Microsoft Research, Tech. Rep. MSR-TR-2011-106, 2011.

[68] Ashish Gupta, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, Thirumalesh Bhat, and Syed Emran. Detecting energy patterns in software development. *Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA*, 98052, 2011.

[69] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary Jane Irwin, Narayanan Vijaykrishnan, and Mahmut Kandemir. Using complete machine simulation for software power estimation: The softwatt approach. In *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pages 141–150. IEEE, 2002.

[70] HammerDB. Database load testing and benchmarking, (accessed September, 2016). `http://www.hammerdb.com/index.html`.

[71] Ahmed E Hassan. The road ahead for mining software repositories. In *Frontiers of Software Maintenance, 2008. FoSM 2008.*, pages 48–57. IEEE, 2008.

[72] Ahmed E Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.

[73] Ahmed E Hassan and Tao Xie. Software intelligence: the future of mining software engineering data. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 161–166. ACM, 2010.

[74] Anthony Hayter. *Probability and statistics for engineers and scientists.* Nelson Education, 2012.

[75] Donald Hedeker and Robert D Gibbons. A random-effects ordinal regression model for multilevel analysis. *Biometrics*, pages 933–944, 1994.

[76] Donald Hedeker, Robert D Gibbons, and Brian R Flay. Random-effects regression models for clustered data with an example from smoking prevention research. *Journal of consulting and clinical psychology*, 62(4):757, 1994.

[77] Israel Herraiz and Ahmed E Hassan. Beyond lines of code: Do we need more complexity metrics? *Making software: what really works, and why we believe it*, pages 125–141, 2010.

[78] Lorenz M Hilty. *Information Technology and Sustainability: Essays on the Relationship Between ICT and Sustainable Development.* BoD–Books on Demand, 2008.

[79] Abraham Hindle. Greenmining, (accessed September, 2016). `https://github.com/abramhindle/green-data-msr/tree/master/green-mining/data`.

[80] Abram Hindle. Green mining: A methodology of relating software change to power consumption. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 78–87. IEEE Press, 2012.

[81] Abram Hindle. Green mining: investigating power consumption across versions. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1301–1304. IEEE, 2012.

[82] Abram Hindle. Green mining: A methodology of relating software change and configuration to power consumption–web edition. *Web Edition*, 2014. `http://webdocs.cs.ualberta.ca/~hindle1/2014/green-emse-web-edition.pd`.

[83] Abram Hindle. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering*, 20(2):374–409, 2015.

[84] Dennis E Hinkle, William Wiersma, and Stephen G Jurs. Applied statistics for the behavioral sciences. *Journal of educational statistics*, 15(1):84–87, 2003.

[85] Joop J Hox, Mirjam Moerbeek, and Rens van de Schoot. *Multilevel analysis: Techniques and applications*. Routledge, 2010.

[86] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93. IEEE Computer Society, 2003.

[87] Natalia Juristo. Keynote: Use and misuse of the term experiment in msr research. In *PROMISE16: Proc.Empirical Software Engineering Conference*, 2016.

[88] Natalia Juristo and Ana M Moreno. *Basics of software engineering experimentation*. Springer Science & Business Media, 2013.

[89] Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice*, 19(2):77–131, 2007.

[90] Georgios Kalaitzoglou, Magiel Bruntink, and Joost Visser. A practical model for evaluating the energy efficiency of software applications. In *ICT for Sustainability 2014 (ICT4S-14)*, pages 77–86. Atlantis Press, 2014.

[91] Chris F Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, 1987.

[92] Barbara Kitchenham. Software development cost models. *Software Reliability Handbook, P. Rook (ed.), Elsevier Applied Science, NY*, pages 487–517, 1990.

[93] Barbara Kitchenham and Emilia Mendes. Software productivity measurement using multiple size measures. *IEEE Transactions on Software Engineering*, 30(12):1023–1035, 2004.

[94] Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger. Case studies for method and tool evaluation. *IEEE software*, 12(4):52, 1995.

[95] Barbara A Kitchenham and Emilia Mendes. A comparison of cross-company and within-company effort estimation models for web applications. In *Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering, Edinburgh, Scotland, UK*, pages 47–55, 2004.

[96] Barbara A Kitchenham, Emilia Mendes, and Guilherme H Travassos. Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering*, 33(5), 2007.

[97] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, 28(8):721–734, 2002.

[98] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering*, 28(8):721–734, 2002.

[99] Barbara A Kitchenham, Lesley M Pickard, Stephen G. MacDonell, and Martin J. Shepperd. What accuracy statistics really measure. *IEEE Proceedings-Software*, 148(3):81–85, 2001.

[100] Barbara A Kitchenham and NR Taylor. Software cost models. *ICL technical journal*, 4(1):73–102, 1984.

[101] Ekrem Kocaguneli, Ayse Tosun, Ayse Basar Bener, Burak Turhan, and Bora Caglayan. Prest: An intelligent software metrics extraction, analysis and defect prediction tool. In *SEKE*, pages 637–642, 2009.

[102] Sedef Akınlı Koçak, Andriy Miranskyy, Gülfem Işıklar Alptekin, Ayşe Başar Bener, and Enzo Cialini. The impact of improving software functionality on environmental sustainability. *on Information and Communication Technologies*, page 95, 2013.

[103] Patricia Lago. Software and sustainability, 2015, (accessed June, 2016). `http://dare.ubvu.vu.nl/bitstream/handle/1871/53978/Oratie_Lago.pdf?sequence=1`.

[104] Patricia Lago, Niklaus Meyer, Maurizio Morisio, Hausi A Müller, and Giuseppe Scanniello. Leveraging energy efficiency to software users: summary of the second greens workshop, at icse 2013. *ACM SIGSOFT Software Engineering Notes*, 39(1):36–38, 2014.

[105] Linda M Laird and M Carol Brennan. *Software measurement and estimation: a practical approach*, volume 2. John Wiley & Sons, 2006.

[106] John P Lamb. *The greening of IT: how companies can make a difference for the environment.* IBM Press/Pearson, 2009.

[107] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):160–171, 2003.

[108] Zhichao Li, Radu Grosu, Priya Sehgal, Scott A Smolka, Scott D Stoller, and Erez Zadok. On the energy consumption and performance of systems software. In *Proceedings of the 4th Annual International Conference on Systems and Storage*, page 8. ACM, 2011.

[109] Li C Liu and Donald Hedeker. A mixed-effects regression model for longitudinal multivariate ordinal data. *Biometrics*, 62(1):261–268, 2006.

[110] Jacob R Lorch and Alan Jay Smith. Software strategies for portable computer energy management. *Personal Communications, IEEE*, 5(3):60–73, 1998.

[111] Alan MacCormack, Chris F Kemerer, Michael Cusumano, and Bill Crandall. Trade-offs between productivity and quality in selecting software development practices. *Ieee Software*, 20(5):78–85, 2003.

[112] Stephen G MacDonell and Martin J Shepperd. Comparing local and global software effort estimation models–reflections on a systematic review. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 401–409. IEEE, 2007.

[113] CJ Mann. Observational research methods. research design ii: cohort, cross sectional, and case-control studies. *Emergency Medicine Journal*, 20(1):54–60, 2003.

[114] IBM DB2 manual. Db2 database product documentation, (accessed August, 2015). `www-01.ibm.com/support/docview.wss?uid=swg27009474`.

[115] Marius Marcu, Mircea Vladutiu, Horatiu Moldovan, and Mircea Popa. Thermal benchmark and power benchmark software. *arXiv preprint arXiv:0709.1834*, 2007.

[116] Thomas J McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, 4:308–320, 1976.

[117] William Mendenhall, Robert J Beaver, and Barbara M Beaver. *Introduction to probability and statistics*. Cengage Learning, 2012.

[118] Emilia Mendes and Barbara Kitchenham. Further comparison of cross-company and within-company effort estimation models for web applications. In *Software Metrics, 2004. Proceedings. 10th International Symposium on*, pages 348–357. IEEE, 2004.

[119] Tim Menzies, Andrew Butcher, David Cok, Andrian Marcus, Lucas Layman, Forrest Shull, Burak Turhan, and Thomas Zimmermann. Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on software engineering*, 39(6):822–834, 2013.

[120] Tim Menzies, Justin S Di Stefano, Mike Chapman, and Ken McGill. Metrics that matter. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 51–57. IEEE, 2002.

[121] Tim Menzies, Justin DiStefano, Andres Orrego, and R Chapman. Assessing predictors of software defects. In *Proc. Workshop Predictive Software Models*, 2004.

[122] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, 33(1):2–13, 2007.

[123] Tim Menzies, Ekrem Kocaguneli, Burak Turhan, Leandro Minku, and Fayola Peters. *Sharing data and models in software engineering.* Morgan Kaufmann, 2014.

[124] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang, and Ayşe Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17(4):375–407, 2010.

[125] Arthur Mickoleit et al. Greener and smarter: Icts, the environment and climate change. Technical report, OECD Publishing, 2010.

[126] Everald E Mills. Software metrics. Technical report, DTIC Document, 1988.

[127] Leandro L Minku and Xin Yao. How to make best use of cross-company data in software effort estimation? In *Proceedings of the 36th International Conference on Software Engineering*, pages 446–456. ACM, 2014.

[128] A Miransky, Sedef Akinli Koçak, Enzo Cialini, and Ayse Basar Bener. Save energy with the db2 10.1 for linux, unix, and windows data compression feature. *IBM DeveloperWoks, Technical Library*, 2013. `https://www.ibm.com/developerworks/data/library/techarticle/dm-1302db2compression/`.

[129] Andriy Miranskyy, Zainab Al-zanbouri, David Godwin, and Ayşe Başar Bener. Database engines: Evolution of greenness, (2017). Journal of Software Evaluation and Process, Accepted.

[130] Ayse Tosun Mısırlı. *Modelling Software Reliability Using Hybrid Bayesian Networks.* PhD thesis, Bogaziçi University, 2012.

[131] Ayse Tosun Mısırlı, Bora Çağlayan, Andriy V Miranskyy, Ayşe Bener, and Nuzio Ruffolo. Different strokes for different folks: A case study on software metrics for different defect categories. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, pages 45–51. ACM, 2011.

[132] Douglas C Montgomery. *Design and analysis of experiments*. John Wiley & Sons, 2008.

[133] MSR. International conference on mining software repositories, (accessed November, 2016). `http://2017.msrconf.org/#/home`.

[134] John C Munson and Sebastian G Elbaum. Code churn: A measure for estimating the impact of code change. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 24–31. IEEE, 1998.

[135] Glenford J Myers. An extension to the cyclomatic measure of program complexity. *ACM Sigplan Notices*, 12(10):61–64, 1977.

[136] MySQL. Mysql 5.7 reference manual: The innodb storage engine, (accessed October, 2016). `http://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html`.

[137] MySQL. Mysql 5.7 reference manual: The myisam storage engine, (accessed October, 2016). `http://dev.mysql.com/doc/refman/5.7/en/myisam-storage-engine.html`.

[138] Nachiappan Nagappan, E Michael Maximilien, Thirumalesh Bhat, and Laurie Williams. Realizing quality improvement through test driven development: results and experiences of four industrial teams. *Empirical Software Engineering*, 13(3):289–302, 2008.

[139] Nachiappan Nagappan, Laurie Williams, Mladen Vouk, and Jason Osborne. Using in-process testing metrics to estimate post-release field quality. In *The 18th IEEE International Symposium on Software Reliability (ISSRE'07)*, pages 209–214. IEEE, 2007.

[140] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 382–391. IEEE Press, 2013.

[141] Stefan Naumann, Markus Dick, Eva Kern, and Timo Johann. The greensoft model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294–304, 2011.

[142] Allen P Nikora and John C Munson. Developing fault predictors for evolving software systems. In *Software Metrics Symposium, 2003. Proceedings. Ninth International*, pages 338–350. IEEE, 2003.

[143] National Oceanic and Atmospheric Administration (NOAA). Noaas global greenhouse gas reference networ, (accessed March, 2017). `http://research.noaa.gov/Home.aspx`.

[144] Adam J Oliner, Anand P Iyer, Ion Stoica, Eemil Lagerspetz, and Sasu Tarkoma. Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 10. ACM, 2013.

[145] UN Framework Convention on Climate Change OP212015. Adaptation of the paris agreement, (accessed January, 2016). `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32012L0019`.

[146] The R Stats Package. Documentation for package stats version 3.4.0, (accessed November, 2016). `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/00Index.html`.

[147] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[148] Birgit Penzenstadler et al. Safety, security, now sustainability: The nonfunctional requirement for the 21st century. *IEEE Software*, 31(3):40–47, 2014.

[149] Dewayne E Perry, Adam A Porter, and Lawrence G Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 345–355. ACM, 2000.

[150] Shari Lawrence Pfleeger and Barbara A Kitchenham. Experimental design and analysis in software engineering part 15. *ACM SIGSOFT Software Engineering Notes*, 19, 20(4, 1, 2):16–20, 22–26, 13–15, 1994.

[151] PMCCABE. Mccabe-style function complexity and line counting for c and c++, (accessed December, 2016). `https://people.debian.org/~bame/pmccabe/overview.html`.

[152] Dan Port and Marcel Korte. Comparative studies of the model evaluation criterions mmre and pred in software cost estimation research. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 51–60. ACM, 2008.

[153] WattsUP? Pro. Wattplug load meters, (accessed January, 2016). `http://wattsupmeters.com/secure/products.php?pn=0`.

[154] Giuseppe Procaccianti. *Energy-Efficient Software*. PhD thesis, Politecnico di Torino-VU University Amsterdam (Paesi Bassi), 2015.

[155] Giuseppe Procaccianti, Patricia Lago, and Grace A Lewis. Green architectural tactics for the cloud. In *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, pages 41–44. IEEE, 2014.

[156] Peter Puschner and Ch Koza. Calculating the maximum execution time of real-time programs. *Real-Time Systems*, 1(2):159–176, 1989.

[157] Lawrence H Putnam and Ann Fitzsimmons. Estimating software costs. *Datamation*, 25(10):189, 1979.

[158] Stephen W Raudenbush and Anthony S Bryk. *Hierarchical linear models: Applications and data analysis methods*, volume 1. Sage, 2002.

[159] Nornadiah Mohd Razali, Yap Bee Wah, et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33, 2011.

[160] Paul R Rosenbaum. Observational studies. In *Observational Studies*, pages 1–17. Springer, 2002.

[161] Kaushik Roy and Mark C Johnson. Software design for low power. In *Low power design in deep submicron electronics*, pages 433–460. Springer, 1997.

[162] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[163] Neil J Salkind and Terese Rainwater. *Exploring research*. Prentice Hall Upper Saddle River, NJ, 2009.

[164] Alvin Saperstein. *Physics: energy in the environment*. Little-Brown, 1975.

[165] SciTools. Understand static code analysis tool, (accessed September, 2016). `https://scitools.com/features/#feature-category-metrics-reports`.

[166] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4):557–572, 1999.

[167] Jason WA Selby. *Unconventional applications of compiler analysis*. PhD thesis, University of Waterloo, 2011.

[168] Weiyi Shang, Zhen Ming Jiang, Bram Adams, Ahmed E Hassan, Michael W Godfrey, Mohamed Nasser, and Parminder Flora. An exploratory study of the evolution of communicated information about the execution of large software systems. *Journal of Software: Evolution and Process*, 26(1):3–26, 2014.

[169] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[170] Colin Shearer. The crisp-dm model: the new blueprint for data mining. *Journal of data warehousing*, 5(4):13–22, 2000.

[171] Martin Shepperd and Steve MacDonell. Evaluating prediction systems in software project estimation. *Information and Software Technology*, 54(8):820–827, 2012.

[172] Forrest Shull, Janice Singer, and Dag IK Sjøberg. *Guide to advanced empirical software engineering*, volume 93. Springer, 2008.

[173] Amit Sinha and Anantha P Chandrakasan. Jouletrack: a web based tool for software energy profiling. In *Proceedings of the 38th annual Design Automation Conference*, pages 220–225. ACM, 2001.

[174] David Sky. Db2 v10.1 query performance enhancements. *IBM DeveloperWoks, Technical Library*, 2012.

[175] Tom AB Snijders. Multilevel analysis. In *International Encyclopedia of Statistical Science*, pages 879–882. Springer, 2011.

[176] Jae W Song and Kevin C Chung. Observational studies: cohort and case-control studies. *Plastic and reconstructive surgery*, 126(6):2234, 2010.

[177] Bob Steigerwald, Rajshree Chabukswar, Karthik Krishnan, and JD Vega. Creating energy-efficient software. *Context*, 184(1), 2007.

[178] J Sundell. libtorrent and rtorrent project, (accessed September, 2016). `http://libtorrent.rakshasa.no/`.

[179] Juha Taina and Simo Mäkinen. Green software quality factors. In *Green in Software Engineering*, pages 129–154. Springer, 2015.

[180] Richard Taylor. Interpretation of the correlation coefficient: a basic review. *Journal of diagnostic medical sonography*, 6(1):35–39, 1990.

[181] Walter F Tichy, Nico Habermann, and Lutz Prechelt. Summary of the dagstuhl workshop on future directions in software engineering: February 17–21, 1992, schloß dagstuhl. *ACM SIGSOFT Software Engineering Notes*, 18(1):35–48, 1993.

[182] Walter F Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst A Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, 28(1):9–18, 1995.

[183] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):437–445, 1994.

[184] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. In *Technologies for wireless computing*, pages 139–154. Springer, 1996.

[185] Oscar Torres-Reyna. Getting started in fixed/random effects models using r. *Princeton University, Data and*, 15, 2010.

[186] Ayşe Tosun, Ayşe Bener, Burak Turhan, and Tim Menzies. Practical considerations in deploying statistical methods for defect prediction: A case study within the turkish telecommunications industry. *Information and Software Technology*, 52(11):1242–1257, 2010.

[187] Ayse Tosun, Burak Turhan, and Ayse Basar Bener. Feature weighting heuristics for analogy-based effort estimation models. *Expert Systems with Applications*, 36(7):10325–10333, 2009.

[188] Burak Turhan. *Improving the performance of software defect predictions with internal and external information sources*. PhD thesis, Bogaziçi University, 2008.

[189] Burak Turhan, Tim Menzies, Ayşe B Bener, and Justin Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5):540–578, 2009.

[190] Keith S Vallerio, Lin Zhong, and Niraj K Jha. Energy-efficient graphical user interface design. *IEEE Transactions on Mobile Computing*, 5(7):846–859, 2006.

[191] Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. Trends in worldwide ict electricity consumption from 2007 to 2012. *Computer Communications*, 50:64–76, 2014.

[192] Vasanth Venkatachalam and Michael Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys (CSUR)*, 37(3):195–237, 2005.

[193] Willem Vereecken, Ward Van Heddeghem, Didier Colle, Mario Pickavet, and Piet Demeester. Overall ict footprint and green communication technologies. In *4th International Symposium on Communications, Control and Signal Processing (ISCCSP 2010)*. IEEE, 2010.

[194] Stefan Wagner. *Software product quality control*. Springer, 2013.

[195] KS Wang. Linear and non-linear mixed models in longitudinal studies and complex survey data. *J Biom Biostat*, 7(290):2, 2016.

[196] WattsUP? Communication protocol, wattsup: Power analyzer, watt meter and electricity monitor, (accessed April, 2015). `https://www.wattsupmeters.com/secure/downloads/CommunicationsProtocol090824.pdf`.

[197] WattsUP? Operators manual, wattsup: Power analyzer, watt meter and electricity monitor, (accessed April, 2015). `https://www.wattsupmeters.com/secure/downloads/manual_rev_9_corded0812.pdf`.

[198] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[199] Isabella Wieczorek and Melanie Ruhe. How valuable is company-specific data compared to multi-company data for software cost estimation? In *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, pages 237–246. IEEE, 2002.

[200] Roel Wieringa and Maya Daneva. Six strategies for generalizing software engineering theories. *Science of computer programming*, 101:136–152, 2015.

[201] Bodo Winter. A very basic tutorial for performing linear mixed effects analyses: Tutorial 2, 2015.

[202] C Wohlin, P Runeson, M Host, MC Ohlsson, B Regnell, and A Wesslen. Experimentation in software engineering: an introduction. 2000, 2000.

[203] Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*, pages 7–23. Springer, 2003.

[204] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[205] Jeffrey M Wooldridge. *Introductory econometrics: A modern approach*. Nelson Education, 2015.

[206] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. A programming environment with runtime energy characterization for energy-aware applications. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 141–146. IEEE, 2007.

[207] Wanghong Yuan and Klara Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 149–163. ACM, 2003.

[208] Marvin V Zelkowitz and Dolores R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23–31, 1998.

[209] Chenlei Zhang and Abram Hindle. A green miner's dataset: mining the impact of software change on energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 400–403. ACM, 2014.

[210] Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. Towards building a universal defect prediction model with rank transformed predictors. *Empirical Software Engineering*, 21(5):2107–2145, 2016.

[211] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105–114. ACM, 2010.

[212] Zhiwu Zhang, Edward S Buckler, Terry M Casstevens, and Peter J Bradbury. Software engineering the mixed model for genome-wide association studies on large samples. *Briefings in bioinformatics*, 10(6):664–675, 2009.

[213] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain

vs. process. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 91–100. ACM, 2009.

[214] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005.

[215] Alain Zuur, Elena N Ieno, and Graham M Smith. *Analyzing ecological data*. Springer Science & Business Media, 2007.