Factors that Impact the Cloud Portability of Legacy Web Applications

Gabriel Costa Silva

Doctor of Philosophy

University of York

Computer Science

September 2016

Abstract

The technological dependency of products or services provided by a particular cloud platform or provider (i.e. cloud vendor lock-in) leaves cloud users unprotected against service failures and providers going out of business, and unable to modernise their software applications by exploiting new technologies and cheaper services from alternative clouds. High portability is key to ensure a smooth migration of software applications between clouds, reducing the risk of vendor lock-in. This research identifies and models key factors that impact the portability of legacy web applications in cloud computing. Unlike existing cloud portability studies, we use a combination of techniques from empirical software engineering, software quality and areas related to cloud, including service-oriented computing and distributed systems, to carry out a rigorous experimental study of four factors impacting on cloud application portability. In addition, we exploit established methods for software effort prediction to build regression models for predicting the effort required to increase cloud application portability. Our results show that software coupling, authentication technology, cloud platform and service are statistically significant and scientifically relevant factors for cloud application portability in the experiments undertaken. Furthermore, the experimental data enabled the development of fair (mean magnitude of relative error, MMRE, between 0.493 and 0.875), good (MMRE between 0.386 and 0.493) and excellent (MMRE not exceeding 0.368) regression models for predicting the effort of increasing the portability of legacy cloud applications. By providing empirical evidence of factors that impact cloud application portability and building effort prediction models, our research contributes to improving decision making when migrating legacy applications between clouds, and to mitigating the risks associated with cloud vendor lock-in.

Contents

List of Figures

List of Tables

To my beloved wife, Josiane, and my precious son, Miguel.

Acknowledgements

I am deeply grateful to:

"Tia Lu", for giving me this crazy dream of pursuing a PhD; Dr Yandre Maldonado e Gomes da Costa, for introducing me to the academic research 12 years ago; Prof. Itana Gimenes, for her advice and support before starting this journey; Mr Rafael Cassolato, for his technical support during endless discussions about software architecture, cloud computing and development frameworks; Prof. Claudete Werner, for her support with organising two of my experiments; Dr Reginaldo Ré, for his support with organising one of my experiments; Mr Munif Gebara, for his support with organising one of my experiments; Universidade Tecnológica Federal do Paraná, UNIPAR and Impact Information Technology, for granting access to their facilities for my experiments, and for allowing their students to take part in the experiments; Dr Simos Gerasimou, Dr Yasmin Rafiq, Dr Babajide Ogunyomi, Dr Thomas Richardson, Dr Colin Paterson, Dr Thanasis Zolotas and Mr Adolfo Sanchez-Barbudo Herrera, for their constant support throughout my PhD; Prof. Richard Paige, for his kind support throughout my PhD; Dr Louis Rose, Dr Radu Calinescu and Dr Fiona Polack, for their valuable comments and advice on this research; and Prof. Dana Petcu, for contributing to this research with valuable comments during my final assessment.

Declaration

I declare that contents of this thesis are the outcome of my own research that was conducted between January 2013 and September 2016 under the supervision of Dr Louis Rose, Dr Radu Calinescu and Dr Fiona Polack. This thesis has not been submitted for any other award at this or any other institution. Parts of this thesis have been previously published in the following:

Silva, G. C., Rose, L. M., & Calinescu, R. (2013). A Systematic Review of Cloud Lock-In Solutions. In 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (pp. 363–368). Bristol, UK: IEEE.

Silva, G. C., Rose, L. M., & Calinescu, R. (2013). Towards a Model-Driven Solution to the Vendor Lock-In Problem in Cloud Computing. In 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (pp. 711–716). Bristol, UK: IEEE.

Silva, G. C., Rose, L. M., & Calinescu, R. (2014). Cloud DSL: A Language for Supporting Cloud Portability by Describing Cloud Entities. In MD2P2 2014 - Model-Driven Development Processes and Practices (pp. 18–27). Valencia, Spain: ACM/IEEE.

Silva, G. C., Rose, L. M., & Calinescu, R. (2014). A Qualitative Study of Model Transformation Development Approaches: Supporting Novice Developers. In MD2P2 2014 – Model-Driven Development Processes and Practices (pp. 18–27). Valencia, Spain: ACM/IEEE.

The four publications presented above are result of my own research, written by me with the assistance of my supervisors.

Chapter 1

Introduction

1.1 Motivation

The National Institute of Standards and Technology (NIST) defines cloud computing as *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [155]. Cloud computing has long been regarded as a key technology, disruptive to both IT and business. For instance, cloud computing has featured among the top 10 strategic technologies listed by 2008-2013 Gartner reports[1]. Practitioners surveys suggest an ever increasing adoption of cloud computing in public and private sectors [47, 77, 169]. Academics have also shown much interest in cloud computing, which has been the subject of multiple systematic literature reviews [31, 79, 107, 144].

However, cloud computing still faces significant challenges, such as ensuring data confidentiality and privacy [252, 253, 264], other security aspects [109, 261] and high service availability [22, 63]. Very important among these challenges is overcoming the vendor lock-in [22, 44, 77, 109, 163, 253]. In cloud computing, vendor lock-in (or cloud lock-in) is the technological dependency of products or services provided by a particular cloud platform or provider [4, 59, 173, 202].

Although vendor lock-in is not a new issue in computer science [59, 253], it is critical in cloud computing because it renders cloud users unprotected against service failures [251] and providers going out of business [22], and unable to modernise their software

[1] `http://www.gartner.com/technology/research/top-10-technology-trends/`

systems [216, 238] and to reduce costs by exploiting new technologies [66] and cheaper services [202] from alternative clouds. High portability is key to ensuring a smooth migration of software applications between clouds [173, 185], thus reducing the risk of vendor lock-in. Portability is defined by the ISO/IEC 25010[1] as the *"degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another."*

The differences among the APIs [138, 197], semantics [147], and technologies [157, 235, 260] adopted by different cloud platforms and providers have been widely regarded as responsible for vendor lock-in, as they hinder the migration between clouds and reduce the portability of cloud *resources*. Cloud *resources* include, but are not limited to, data [35, 146, 191, 208], applications [35, 98, 146, 191, 208, 257], components [2, 114, 186], workloads [176], VMs [98, 148], configurations [176], and live deployments [176]. Overall, what is considered as a cloud resource varies according to the cloud service [211]. For instance, for Google Docs, documents (e.g., a spreadsheet) are considered as resources whereas for DropBox, files are considered as resources.

The portability of cloud resources, or simply cloud portability (Section 2.4), has been addressed by several solutions that aim to mitigating differences in APIs, semantics and technologies used by cloud platforms and providers [210]. However, these solutions do not provide the means to fully understand the cloud portability phenomenon. Whereas most studies for cloud portability concentrate on proposing technological and standardisation products (e.g., methods and tools), this thesis concentrates on (i) investigating factors that impact cloud *application* portability as a means to mitigate risks associated to vendor lock-in, and (ii) using these factors to support informed decision making on cloud *application* migration. To do so, we combine and exploit sound techniques from empirical software engineering, software quality and software effort prediction. In addition, we identify established strategies for dealing with portability issues in related areas including service-oriented computing and distributed systems, and we extend the applicability of these strategies to the cloud computing domain.

1.2 Research Overview

We conducted this research project using Wazlawick's framework for research in computer science [250]. We chose this research framework due to its comprehensive coverage of all aspects of a research project and its focus on computer science research. The

[1] `http://iso25000.com/index.php/en/iso-25000-standards/iso-25010`

terms *highlighted* in the following description of our research are taken from Wazlawick's framework, which is summarised in Appendix A.1.

The *issue* addressed in this research is the migration of legacy web applications between clouds. The migration addressed in this research does not focus on a particular *service model* (Section 2.1.2). This *issue* is part of the cloud portability *subject* in the cloud computing *area*. Through a *literature review* (Section 2.2), we identify that application migration in cloud is hindered by the vendor lock-in, which creates a technological dependence of cloud users on a particular cloud platform due to the different semantics, technologies and APIs adopted by these platforms. To enable the migration of resources (e.g., applications, data, virtual machines) between clouds, researchers, practitioners, standardisation bodies and European projects have proposed several technical solutions for new applications, mainly focusing on means that abstract cloud differences.

Our *literature review* identifies four major gaps in the existing cloud portability research (Sections 2.2.5 and 2.3). First, the existing cloud portability literature lacks empirical evidence. Second, existing research disregard existing solutions to similar problems from related areas (e.g., distributed computing) and socio-technical and business challenges. Next, solutions for cloud portability mainly focus on the transference of resources from one cloud to another although this is only one of several phases in the migration process. Finally, although real experiences on software migration show that the engineering effort to carry out a resource migration vary significantly (which impact on the total migration cost), the migration effort is often overlooked in cloud portability research.

Our *literature review* underpinned the definition of the *general objective* of this research: to investigate challenges associated with increasing the portability of legacy web applications in the cloud (i.e., the *cloud application portability*), focusing on the migration analysis phase.

By investigating the *literature* on software migration (Section 2.3), software quality (Section 2.4), software effort prediction (Appendix C) and the literature on cloud-related fields (i.e., distributed systems, service-oriented computing and outsourcing) (Section 2.1.3), we defined the following *hypothesis*:

> *"Design properties and technologies adopted by an application, along with environmental aspects, impact cloud application portability and can be used as indicators for predicting the effort of improving the cloud application portability."*

Starting from this *hypothesis*, we refined our *general objective* into *specific objectives* as follows:

1. To identify factors that impact on cloud application portability; and
2. To devise prediction models for the effort of increasing the cloud portability of legacy web applications.

As any research, our hypothesis has a well-defined scope (*limitations*). Firstly, the only design property investigated is software coupling. Secondly, the technologies investigated consist of two security authentication mechanisms. Thirdly, cloud platform and service are the two factors investigated as environmental aspects. Due to the limited time to conduct this doctoral research, the selection of the four factors investigated in this research was driven by convenience – they were the first four factors suggesting some impact on cloud portability that we could identify in the literature.

Our research scope also includes the size of legacy web applications covered in the research – medium-sized (900 to 40,000 lines of code according to Feigenspan et al. [82]) legacy web information systems. Next, four activities for cloud application migration are considered: (i) message queuing adoption; (ii) single sign-on adoption; (iii) cloud service configuration; and (iv) application deployment. The rationale for these activities are detailed in Chapters 3, 4 and 5. The approach used for building effort prediction models is point prediction linear regression. Finally, the migration effort is measured in terms of engineering time.

To achieve the *specific objective* 1, we carried out four rigorous experiments. It is important to note that each experiment was carried out within a limited scope due to research interests and time and budget limitations.

To test the first part of our *hypothesis* and achieve the *specific objective* 1, we adopted the Research Onion *methodology*[1]. We summarise this methodology in Appendix A.2. The Research Onion methodology defines several layers representing different methods and stances for scientific research. Our research adopts positivism as its philosophical stance. According to Easterbrook et al. [71], positivism advocates *"logical inference from a set of basic observable facts,"* and is often associated with controlled experiments. This is the dominant stance in computer science [203].

Our investigation uses a deductive approach. According to Sjøberg et al. [213], in a deductive approach a theory is devised in the theoretical realm, and then oper-

[1] `http://onion.derby.ac.uk`

ationalised in a practical environment, e.g. through experimentation. As we try to identify causal relationships between cloud application portability and a set of factors, we use experimentation as our research strategy. According to Wohlin et al. [256], experimentation is an empirical method *"that manipulates one factor or variable of the studied setting."* Appendix B provides details about the experimentation framework used in this research.

Our research is monomethod, as it uses only quantitative methods. According to Sjøberg, Dybå and Jørgensen [212], quantitative methods *"collect numerical data and analyse it using statistical methods."* From a time horizon viewpoint, our work is cross-sectional research, which means that our experiments considered participant groups at a single point in time [126].

Finally, our techniques and procedures for data collection and analysis vary according to the experiment carried out, but we mainly used forms for data collection and non-parametric statistics for data analysis. The rationale for the data analysis choice is that (i) non-parametric statistics are less sensitive to outliers and (ii) non-parametric tests make no assumption about the data distribution. Foundation for statistical tests used in this research can be found in [84, 113, 256].

The *methodology* for tackling the second part of our *hypothesis* and achieving the *specific objective* 2 is driven by established approaches to software effort prediction that consist of using a data set, a prediction approach, and an evaluation method. These approaches are presented in Appendix C.

This research contributes to (*expected outcomes*): (i) supporting informed decisions regarding the cloud migration of software applications, and (ii) mitigating the risks associated to vendor lock-in. The two major contributions of this research are the empirical identification of four factors that impact cloud application portability and a set of regression models for predicting the effort of increasing cloud application portability. These contributions are important because they increase our understanding of cloud portability and provide an essential tool to analyse the technical and financial feasibility of a software migration in the cloud.

1.3 Ethics and Ethical Consent

To conduct the experiments presented in Chapters 3, 4 and 5, we followed the University of York guidance on ethics in research [1]. In particular, we (i) gave participants autonomy to make decisions regarding the experiment, including the option not to participate; (ii) informed participants about any known risk of harm from their participation; (iii)

made clear the purpose and expected benefits of our experiments; and (iv) ensured the confidentiality of the experimental data and personal information. Appendix D presents the template form used for collecting participant's consent to participate in experiments.

1.4 Thesis Structure

The remainder of the thesis is structured as follows. Chapter 2 reviews cloud computing, vendor lock-in, software migration, and cloud portability concepts and terminology that are key to understanding this thesis.

Chapter 3 empirically investigates the impact of software coupling on cloud application portability in the context of re-engineering legacy three-tier web applications to decouple application components by adopting a message queuing mechanism. We exploit the results of this investigation by building prediction models to support decision makers in their analysis of the effort required to increase the cloud portability of their applications. Additionally, this chapter complements our literature review by motivating the use and explaining the main concepts of software coupling and message queuing.

Chapter 4 empirically investigates the impact of the choice of security systems on cloud application portability. This work is carried out in the context of modifying the security system of legacy three-tier web applications to enable the use of single sign-on by adopting implementations of the OpenID protocol. We exploit the results of this investigation by building prediction models to support decision makers in their analysis of the effort required to increase the cloud portability of their applications.

Chapter 5 empirically investigates the impact of the cloud platforms and services on cloud application portability, in the context of configuring cloud services and deploying cloud applications. Unlike previous chapters, this investigation required two experiments. We use the results of both investigations to devise prediction models supporting decision makers in their analysis of the engineering effort required to configure cloud services and deploy cloud applications.

Chapter 6 concludes the thesis by summarising the main contributions of this research and outlining directions for future work.

Chapter 2

Background

This chapter reviews key concepts and terminology needed for understanding this thesis. Firstly, we present aspects of cloud computing that contribute to vendor lock-in and briefly introduce existing results from cloud-related areas of computer science that contributes to understanding and tackling vendor lock-in (Section 2.1). Next, we explore the literature on vendor lock-in in cloud computing to identify its consequences, causes and current solutions (Section 2.2). Then, we investigate reports of real experiences on migration to understand the migration process (Section 2.3). Finally, we narrow down to our perspective on cloud portability, which takes the use of sound techniques from software quality and cloud-related literature into consideration to improve the portability of legacy cloud applications in hybrid environments (Section 2.4).

2.1 Cloud Computing

This section surveys concepts of cloud computing and their impact on the subject investigated in this research. Firstly, we present five essential cloud characteristics (Section 2.1.1). We highlight that although these characteristics lead to benefits for cloud adopters, legacy web applications need re-engineering to take full advantage of these benefits. Next, we show two classifications for cloud services - *service* and *deployment models* - that pose challenges for resource migration to or between clouds (Section 2.1.2). Finally, we summarise relevant results from three areas related to cloud computing - distributed systems, service-oriented computing and outsourcing. These areas share several characteristics with cloud computing, and therefore support the understanding of certain cloud aspects relevant for our research (Section 2.1.3).

2.1.1 Essential Cloud Characteristics

NIST [155] presents five essential characteristics of cloud computing. *On-demand resource provisioning* means that cloud resources are available to the cloud user whenever they are required. This characteristic enables cloud users to acquire and release computing resources like public utilities [44, 261]. For example, water is available just by opening a tap, and the amount of water varies as the tap is open. This characteristic leads to benefits such as reduced upfront investment [22, 125, 232], variable cost [63], and reduced time-to-market [154].

Broad network access is the cornerstone to access cloud resources [147, 193] since they may be distributed across different data centres [251, 261] and possibly geographical locations [45]. Two benefits of this characteristic are ubiquity and universal access through different devices [63, 261]. Ubiquity means that cloud resources are available anywhere since they are offered through the Internet [89, 261]. For example, a cloud user can access their resources from the closest datacentre. On the other hand, the heavy network dependency introduces challenges, such as bandwidth limits [253], cost to transfer data [109], and data security [264]. In addition, as we discuss in Section 2.4.5 and address in Chapters 3 and 4, distributing resources across the Internet requires some adaptations to enable legacy web applications to take full advantage of cloud benefits.

Resource pooling is an intrinsic characteristic of cloud. To provide resources to cloud users, the cloud provider maintains a very large infrastructure of hardware and virtualization technologies, resulting in a pool of resources [253]. These resources are shared among cloud users using a multi-tenant model [15]. Cloud platforms and underlying technologies, such as hypervisors, are responsible for managing and provisioning resources to cloud users [261]. This characteristic benefits cloud providers in three different ways. Firstly, it reduces IT spending [163], and maximise and optimise resources [66] by using idle resources [241]. Secondly, it enables cloud providers to take advantage of the economy of scale [89]. Finally, virtualization technologies facilitate the management and maintenance of physical resources [45, 261].

Rapid elasticity allows cloud resources to scale up (e.g., by providing a larger CPU for a VM) and out (e.g., by increasing the number of web containers serving user requests) according to the cloud user needs [22, 155]. Elasticity may also refer to costs. Since the pay-as-you-go model is widely adopted in cloud as the standard billing model [44], the bill increases with the resource usage. The pay-as-you-go billing model, along with on-demand provisioning, reduce upfront investments [261], turning fixed costs into variable costs [63]. Although rapid elasticity is particularly interesting for small and start-up

companies [186], legacy applications require re-engineering to exploit it [15, 52, 125].

Measured services is a characteristic that enables resource usage monitoring. A benefit of this characteristic is resource control and optimisation, such as monitoring unexpected consumption peaks [44, 124], and reducing cloud resource consumption [54, 114].

2.1.2 Service and Deployment Models

Cloud platforms offer a plethora of services, ranging from web deployment containers to ready-to-use software systems. These services are grouped into categories called *service models* [155, 261] based on the resource provided to the cloud user [240]. NIST defines three main service models [155]:

- *Infrastructure-as-a-Service* (IaaS) consists of fundamental computing resources, such as storage [63];
- *Platform-as-a-Service* (PaaS) consists of software platforms where cloud applications run on, such as application servers for web applications [240]; and
- *Software-as-a-Service* (SaaS) consists of on ready-to-use applications, such as email (e.g., Gmail[1]) or CRM (e.g., Sales Cloud[2]) [261].

Like a conventional computing system which comprises multiple interacting levels, cloud service models may also support each other [261]. For example, an email service (SaaS) might be deployed in an application server service (PaaS), which in turn is hosted by a VM (IaaS). The service stack may be in the same cloud platform or not.

Whereas cloud service models classify the type of cloud service, the *deployment model* distinguishes between different types of users that are the target of these cloud services [155]. The three most common deployment models are [66, 261]: (i) *public*, which targets the general public (e.g., Amazon[3]); (ii) *private*, which targets a single organisation; and (iii) *hybrid*, which consists of a composition of the previous two *deployment models*. Although hybrid deployments are a growing trend[4], cloud platforms might be heterogeneous [18, 86, 124, 221]. In this context, heterogeneity means that cloud platforms may differ, mainly, regarding their underpinning APIs, technologies

[1] `http://www.gmail.com`
[2] `http://www.salesforce.com/sales-cloud/overview/`
[3] `http://aws.amazon.com`
[4] `http://www.northbridge.com/2013-cloud-computing-survey`

and semantics. Cloud platform heterogeneity complicates the management of multiple clouds, and contributes to the vendor lock-in (Section 2.2).

It is important to note that different service and deployment models pose different challenges when moving applications to the cloud [15] or between clouds [184]. As we explain in Sections 2.3 and 2.4.5, migration requirements vary according to the source or target service and deployment model, and to the resource that is being migrated.

2.1.3 Cloud-Related Research Areas

Cloud computing inherits characteristics and uses techniques from different areas of computing. Distributed systems, service-oriented computing, and outsourcing are three cloud-related areas particularly relevant for this research.

According to Tanenbaum & van Steen [228], a distributed system is *"a collection of independent computers that appears to its users as a single coherent system."* Grid computing is a type of distributed system [227] that shares several characteristics with cloud [89, 252], such as large scale, geographical distribution, heterogeneity, and resource sharing. Even some challenges are similar, such as the need for migrating legacy systems [51], interoperability [259], security [88] and resource management [41, 76]. As in grid [42, 236, 247], cloud services and applications rely on middleware for their integration and management. In addition, the resource decentralisation requires specific security techniques [228]. The use of middleware and specific security techniques is summarised in Sections 2.4.5, 3.2 and 4.1.

Service-oriented computing is a paradigm whereby *services* represent the building blocks for composing complex applications [175, 177]. In this paradigm, a service is a self-contained, low coupled and language independent software unit that implements a business function [9, 175], such as placing an order in an e-commerce system. These service characteristics enable smooth and transparent interoperation of heterogeneous systems [178]. Although the composition of cloud services is expected to work like the service composition in service-oriented computing [251], the heterogeneous semantics [147], technologies [196] and interfaces [197] adopted by cloud platforms complicate this integration. Section 2.2.5 summarises a review of existing solutions for mitigating these differences in cloud.

Outsourcing is *"a situation in which a company employs another organisation to do some of its work, rather than using its own employees to do it."*[1] Clemons [59] describes cloud computing as an extreme form of IT outsourcing. According to Armbrust et al.

[1] `http://dictionary.cambridge.org/dictionary/english/outsourcing`

[22], in this form of IT outsourcing, maintenance and risks related to infrastructure and provided services are transferred to the cloud provider. On the one hand, outsourcing IT assets by adopting cloud computing provides cloud users with benefits such as [44, 63, 77, 154]: focus on core business, cost reduction, quick access to hardware and software resources, low barrier to innovation, service scalability, reduced upfront investment, and freedom to select their resource provider. On the other hand, as a type of outsourcing, cloud computing inherits a known outsourcing risk - the vendor lock-in [59].

2.2 Vendor Lock-in

One of the expected outcomes of this research is mitigating the risks of vendor lock-in by improving the portability of legacy cloud applications. This section delves into the vendor lock-in issue, explaining its definition and identifying its main characteristics (Section 2.2.1), identifying its consequences for the cloud field (Section 2.2.2), summarising its main causes (Section 2.2.3), and underlining its impact on cloud users and providers (Section 2.2.4). We conclude the section by summarising the findings of a systematic mapping carried out early in this research to establish the existing solutions for vendor lock-in in cloud computing (Section 2.2.5).

2.2.1 Vendor Lock-in Definition

Lewis [141] explains that *"vendor lock-in refers to a situation in which, once an organisation has selected a cloud provider, either it cannot move to another provider or it can change providers but only at great cost."* In this thesis, we use the term "vendor lock-in in cloud computing" to mean the technological dependence of cloud users on a particular cloud *platform* or *provider*.

Weiss [253] observes that vendor dependency was a *"significant consideration in the mainframe era."* Indeed, several studies of legacy systems migration indicate vendor dependency as a motivation to switch to another technology [23, 136, 216]. In summary, we could identify several recurring characteristics [6, 18, 22, 35, 48, 61, 66, 67, 86, 103, 105, 109, 141, 186, 191, 192, 202, 253, 257]:

- makes the cloud user dependent of their cloud platform or provider;
- hinders resource migration;
- requires some changes, or deep modifications on the when the migrate resource is an application;

- has a negative effect on the cloud user, usually related to high costs;
- may have a positive effect on the cloud provider.

We discuss these characteristics in more detail the next sections.

2.2.2 Consequences of Vendor Lock-in for Cloud Computing

Although vendor dependency is a recurrent problem in Computer Science [35], it is critical in cloud computing. Firstly, vendor lock-in has inhibited cloud adoption [98, 103, 141, 169, 180, 191, 202] as potential cloud users are concerned about (i) being unable to undertake countermeasures against service failures [251] and (ii) losing their data [22], for example. Secondly, vendor lock-in limits the cloud user freedom to exploit services from other platforms [67], which negatively affects resource optimisation [66] and impedes the use of new technologies [48]. Next, vendor lock-in creates cloud silos [176, 184]. In this context, a silo is a situation in which the resource (e.g., VM, application) is isolated, restraining its interaction with another application [109] or infrastructure [163], or preventing resource sharing [68].

Finally, vendor lock-in negatively impacts the overall cost of cloud services. Lewis [141] notes that the negotiation power of a cloud user is reduced due to the vendor lock-in. Furthermore, Hamdaqa, Livogiannis & Tahvildari [103] conclude that vendor lock-in is a challenge to cost reduction.

2.2.3 Causes of Vendor Lock-in

Several arguments have been made to explain the causes of vendor lock-in in cloud computing. Some authors argue that the lack of standards is responsible for cloud vendor lock-in [61, 186, 192]. Others attribute vendor lock-in to the heterogeneity of the cloud regarding service models [191], functions [83, 101, 176], environments [13, 192], billing strategies [6, 207], and architecture [189, 208]. However, as we explain in the next paragraphs, the use of different APIs, semantics, and technologies by cloud platforms and providers has been widely accepted as the main cause of cloud vendor lock-in.

APIs play an important role for the programmatic management of cloud services [77]. Therefore, they have a strong influence on vendor lock-in [138, 197]. Even though some cloud platforms offer APIs based on open standards [197], such as WSDL, or REST, they lack a single interface [61, 229]. The lack of a single interface is further complicated by property rights of APIs that prevent cloud providers to adopt homogeneous call formats [4, 59]. The use of different APIs prevents a smooth application migration to

another platform [73, 199, 260]. API incompatibility arises even between supposedly compatible platforms. Flores, Srirama & Paniagua [85] report incompatibilities in their experience on creating a mobile application to access storage services on both Amazon S3 and Eucalyptus, although these platforms claim API compatibility. Similarly, Souza et al. [221] demonstrated incompatibilities between the Amazon Web Services (AWS) and the OpenStack, Eucalyptus and OpenNebula APIs for computing service. Hill & Humphrey [108] observe that these differences in APIs are not only the result of syntax and semantics differences, but also a result of the different service models and services provided [167].

Loutas, Kamateri & Tarabanis [149] conclude that API compatibility is not enough to enable data migration, which also requires semantic interoperability. Semantics refers to the way in which a cloud platform describes characteristics and components of its services [147, 164] (e.g., security mechanism, resources, geographical location), mainly including the entity model [149] and entity names [152]. Analysing semantic differences among storage platforms, Kotecha, Bhise & Chaudhary [135] conclude that *"Migration of an application to another cloud requires an application to be re-written and database to be restructured according to the newly identified cloud service provider's datastore."* Loutas et al. [147] explain that this problem is not limited to data stores, but it is present in IaaS, PaaS, and SaaS. Hill & Humphrey [108] note that semantic differences force the cloud user to choose one specific platform. The same position is advocated by Kotecha, Bhise & Chaudhary [135], Nelson & Uma [164], and Fortis, Munteanu & Negru [86].

Finally, cloud providers use different technologies to internally manage their resources and support their services, including hypervisors and VMs, web containers, and cloud platforms. These technologies also significantly contribute to vendor lock-in [157, 235, 260]. Rochwerger et al. [196] observe that cloud technologies were not developed to be interoperable. Galán et al. [92] emphasise the same idea highlighting the differences between the Amazon EC2 and GoGrid VM image format. The use of platform-specific technology is a relevant aspect of vendor lock-in [18, 86] that is more evident in the packaging technology used by IaaS providers [73, 219]. Different technologies were also a critical issue to enable interoperability in grid computing [119, 249, 259], an area related to cloud. The differences in technologies also hinder interaction between different paradigms, such as grid and cloud [157].

Additionally, current development practices and technologies used for application development may also contribute to the vendor lock-in. Common hindering features are an application design based on proprietary technologies [191], and a monolithic

application architecture [106]. Of course, it would be impossible to develop an application without any commitment to a particular technology, such as a programming language. However, the major issue is the development without paying attention to important concerns, such as portability. Migration cases reported in [23, 216, 230] reveal the need for changes to adapt a legacy system to a new technology. In an extreme case, Lancia, Puccinelli & Lombardi [136] report the need for entirely re-developing a Cobol/mainframe-based system to take advantage of a new technology. Similar problems occur in grid computing since applications are hard-coded to use a specific grid middleware [157, 237, 249].

To illustrate this problem in application development, take into consideration the case of the PetStore application[1] – a reference application developed to demonstrate the features of JEE technology. In this particular case, the application is entirely developed to use a specific technology, limiting its deployment on some cloud platforms. However, there is a more critical aspect: although the application uses the traditional three-layered application architecture, the hard-coding of mechanisms used to ensure data persistence and other common functionality makes difficult and expensive any attempt to migrate it to a PaaS provider, for instance. The problem affects not only legacy applications, but also applications specifically developed for the cloud (cloud-native) [191] and applications migrated to the cloud (cloud-enabled) [15]. In [263], Zhou reports the need for re-designing the application architecture to comply with a specific cloud platform. A similar experience was reported in [232] and [52]. These experiences are evidence that even applications recently migrated to the cloud do not consider flexibility, portability, and interoperability.

On the other hand, van der Linden [238] emphasises the ease of migrating an application designed to be highly portable. Based on their experience with migrating an application to a cloud platform, Chauhan & Babar [52] conclude that applications based on stateless RESTful components are easier to migrate than applications based on other technologies. The authors also highlight the simplicity of migrating persistence components when the database is used only to store data instead of both data and business logic. Finally, the authors advocate, *"it is more convenient and cost effective to evolve an SOA based system to SaaS targeting IaaS clouds as compared to PaaS clouds for which components need to be re-factored according to the API provided by the PaaS provider."* In addition, Andrikopoulos et al. [15] observe that the migration cost is associated to the type of migrated components.

[1] `http://www.oracle.com/technetwork/articles/javaee/petstore-137013.html`

2.2.4 Impact of Vendor Lock-in for Cloud Users and Providers

Vendor lock-in affects both cloud user and cloud provider, although in different ways. For the cloud user, although cloud providers advertise service availabilities very close to 100% [83], they can suffer outages [202] – and the literature is full of cases[1]. The (temporary) unavailability of a cloud provider's services gives rise to a cascading effect, affecting cloud users and their customers (application users), and potentially resulting in business losses for cloud users[2,3,4].

Secondly, the cloud market is still maturing; in this process some cloud providers might go out of business [97]. Gonidis, Paraskakis & Kourtesis [97] note the case of Coghead, a company which offered cloud-based services between 2006 and 2009, and suddenly announced its closure in February 2009. In a similar case reported by Armbust et al. [22], the storage service The Linkup went out of business after losing 45% of their customer data, leaving its 20,000 users in a tricky situation. Finally, another risk to the cloud user is the violation of Service Level Agreements (SLA) [202], causing problems, such as data loss [22].

Another effect of vendor lock-in on the cloud user is the increase in the cost associated with resource migration [48]. Here, costs refer not only to money, but also to time and effort. From this perspective, migrating a resource between providers does not differ from resource migration in more established computing paradigms. Nfila, Dintwe & Rao [166] and Wilson [254] report two processes of migrating a library system to a new one (before the cloud era) that took one, and one and half years, respectively. These experiences can be compared with migrating between SaaS providers since the application is entirely replaced, and only the data is migrated [97]. Taking this long to complete a migration would be prohibitive in the case of a provider going out of business. Even when time is not a problem, the effort to migrate can exceed the expectations. Satzger et al. [202] analyse, *"once an application has been developed based on one particular provider's cloud services and using its specific API, that application is bound to that provider; deploying it on another cloud would usually require completely redesigning and rewriting it."* In addition, migration processes are not free from unexpected problems. Even with a well-structured process and an experienced team, Teppe [230]

[1] http://www.crn.com/slide-shows/cloud/240153188/6-devastating-cloud-outages-over-the-last-6-months.htm

[2] `http://www.computerworld.com/article/2495766/social-business/google-drive-hit-by-three-outages-this-week.html`

[3] `http://www.reuters.com/article/net-us-companies-netflix-idUSBRE8BO06H20121226`

[4] http://www.bloomberg.com/news/articles/2012-12-31/amazon-apologizes-for-christmas-eve-disruption-affecting-netflix

reports problems during code migration. Both time and effort impact on the financial cost [6, 103].

Finally, although the cloud user is still responsible for their resources, the IT control is partially transferred to the cloud provider [109], which introduces challenges [105], such as integration and resource optimisation [66]. Without the flexibility to switch provider, the cloud user must agree with business, technological, and socio-technical decisions taken by a third-party. Another critical aspect of such shared control refers to the data security, privacy and trust [91]. Furthermore, as the control is partially transferred to the cloud provider, responsibilities for the service reliability are also shared. Armbrust et al. [22] cite the case of The Linkup (cloud user) and Nirvanix (cloud provider), in which a failure in the cloud user service led to a dispute between the cloud user and the provider to determine the responsibility for the problem. Since new cloud services have been built on the already complex infrastructure of cloud providers (e.g., Dropbox, Netflix, and Foursquare on Amazon Web Services), control and division of responsibility tend to become critical concerns for large companies.

From an economic perspective, vendor lock-in may bring benefits to the cloud provider [22]. Firstly, vendor lock-in retains users [184, 192], ensuring a permanent customer base and enabling the cloud provider to predict both the usage of resources and the revenue. Secondly, the cloud provider has some flexibility to take decisions, such as raising prices[1]. Even though the flexibility to change the price is somewhat limited (otherwise cloud users could abandon the provider despite the migration costs), the cloud market is different from public utilities, in which governments can control, pressure, or, at least, question companies about their price increases. In the cloud market, big customers can pressure cloud providers [184]; however, the flexibility to decide, and any concessions that may be available for some of them lies in the cloud provider's hands.

Finally, exclusive services (i.e., heterogeneous and provider-specific) may be used to attract new customers [184]. For example, Amazon maintains a page dedicated to report success cases of customers that use their exclusive services[2]. Since the problems between user and provider are rarely disclosed, new customers might be attracted by the promised advantage of a service.

On the other hand, cloud providers can also suffer because of their heterogeneity – one of the causes of vendor lock-in. In [2], the authors highlight the case of hybrid clouds, in which heterogeneity is a problem. In a hybrid cloud, multiple clouds work together

[1] `http://bits.blogs.nytimes.com/2012/10/09/open-vs-closed-the-cloud-wars/?_r=0`
[2] `https://aws.amazon.com/solutions/case-studies/`

coordinated by a broker. To realise such an integration, the authors identified several requirements, such as common formats for VMs and APIs. Additionally, in [3], the authors present several scenarios in which seamless integration among cloud providers could be beneficial, such as bursting, and availability in case of disaster recovery. In a bursting scenario, a cloud provider can temporarily transfer resources to a cloud partner in order to deal with an unexpected workload increase [155]. Extra requests for resources from cloud users are redirected to the partner, balancing the workload and ensuring SLAs.

2.2.5 Existing Solutions for Cloud Lock-in

This section summarises a *systematic mapping* we carried out early in the project to identify and analyse the existing solutions for cloud lock-in (i.e., the vendor lock-in in cloud computing). A systematic mapping is a rigorous method for classifying research results published in a field of interest [188]. Previously published in [210], our review is based on a systematic literature mapping of 721 primary studies that describe the state-of-the-art in managing cloud lock-in, portability and interoperability.

Portability and interoperability are two characteristics of software applications that are often associated with cloud lock-in [173]. According to Petcu [184], *"interoperability is a property referring to the ability of diverse systems and organisations to work together (inter-operate). In computer world, this property has the concrete meaning of exchanging information and use of the information that has been exchanged between two or more systems or components."* We selected 78 of these primary studies for a thorough analysis of cloud standards, commercial products and academic work related to cloud lock-in.

This review shows that most solutions proposed so far are platforms, APIs or architectures addressing infrastructure-as-a-service (IaaS) interoperability. The focus on these solutions may be driven by the cloud market, which has shown interest in these topics [77]. The research presented in this thesis addresses three issues identified in this review: (i) exploiting established solutions from areas that are closely related to cloud computing; (ii) obtaining rigorous empirical evidence to raise the confidence in existing solutions; and (iii) addressing socio-technical and business challenges associated with cloud lock-in (Section 1.2).

2.2.5.1 Review Findings

Solution Identification, Analysis & Classification From a total of 721 research papers, white papers and project websites retrieved, 78 (11%) were identified as provid-

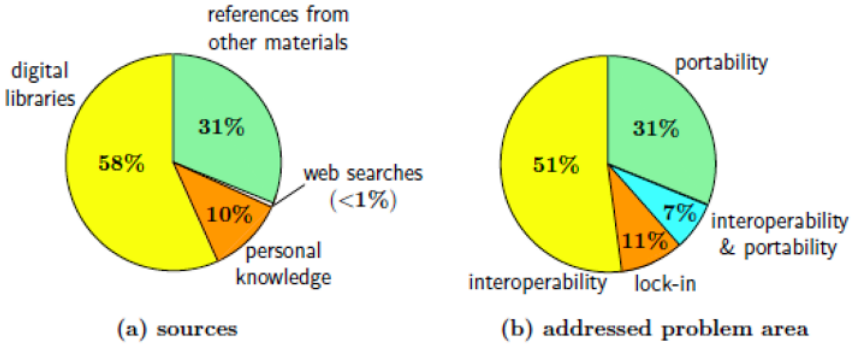


Figure 2.1: Sources and addressed problem areas for the 78 cloud lock-in solutions.

ing solutions to problems related to vendor lock-in, portability and/or interoperability in the cloud. From these solutions, 55 (71%) were published in peer-reviewed journals and conference proceedings, and 23 (29%) were obtained from alternative sources such as web sites and white papers. Figure 2.1 shows the partition of solutions across different types of sources, and the problem areas targeted by these solutions.

Each solution proposes a way to address one or more issues of vendor lock-in, portability or interoperability. Solutions were grouped according to their similarities, resulting in 26 different kinds of solutions (Figure 2.2). The most common type of solution are platforms, followed by APIs and architectures.

It is important to highlight that, unlike similar reviews [98, 148, 184], we focus on classifying solutions according to the type of artefact that they propose instead of the nature of the solution (standard, commercial, and so forth). Thus, although Figure 2.2 does not show an entry for standards, they are present, such as the interfaces SNIA CDMI[1] and UCI[2], the packaging format DMTF OVF[3], and the specification OCCI[4].

Similarly, relevant research projects were also present in our review, such as MODAClouds [18], mOSAIC [186], Cloud4SOA[5], TOSCA [33], RESERVOIR [196], and 4CaaSt[6]. We highlight relevant research projects and standards in our analysis when it is appropriate.

Some types of solution address a combination of portability, interoperability and vendor lock-in issues. For instance, the solution in [157] focuses on interoperability,

[1] http://www.snia.org/cdmi
[2] https://groups.google.com/forum/#!forum/unifiedcloud
[3] https://www.dmtf.org/standards/ovf
[4] http://occi-wg.org
[5] http://www.cloud4soa.eu
[6] http://www.4caast.eu

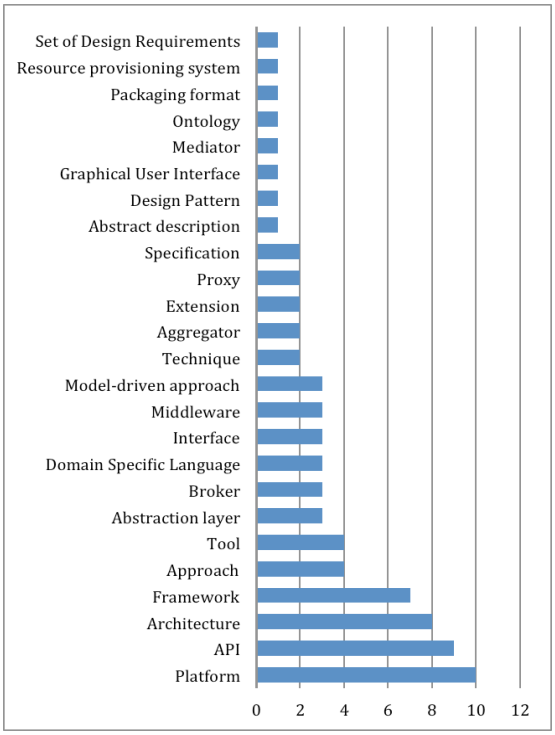


Figure 2.2: Types of solution for cloud lock-in, and their occurrences.

whereas the solution proposed in [186] focuses on portability, and Apache Nuvem[1] focuses on vendor lock-in, although all of them are APIs. Although some solutions were grouped because they propose the same type of solution, they can deal with different parts of a problem. For Architectures, for instance, the solutions in [86] and [45] are general architectures aiming to support interoperability, whereas the solutions in [219], [119], and [235] also propose architectures, but have more specific goals, such as devising an architecture for a broker that supports interoperability. In contrast to some solutions that can be directly understood in context, such as an API or a proxy, other solutions are unclear, such as a framework or an approach. This is the case with the RESERVOIR project [196], which is presented on its official website as a framework. Unless the report that presents the solution makes absolutely clear the type of solution proposed, we used the name proposed by its author.

Whereas 47 solutions (60%) present a developed artefact, 15 solutions (19%) do not present any artefact. Some kind of evaluation is carried out only in 16 solutions (21%). In general, solutions proposed by community, industry and consortiums offer functional versions of their product although they may be experimental versions.

[1] `https://wiki.apache.org/incubator/Nuvem`

Table 2.1: The common sources of primary studies of solutions for avoiding cloud lock-in.

Name	# Studies
IEEE International Conference on Cloud Computing Technology and Science	5
International Conference on Cloud Computing and Services Science	5
Lecture Notes in Computer Science (Journal)	4
IEEE Intl. Conference on Cloud Computing	3
Intl. Conf. on Recent Trends In IT	2
Future Generation Computer Systems (Journal)	2

Most evaluations analyse the performance of the solution, e.g. by measuring throughput and latency [186], inference and query time [164], schedule time [219], overhead in a translation process [73], and time to responds to user requests [257]. We also found solutions for which the cost to the cloud user is being evaluated, namely [182], [45] and [6]. The most common form of evaluation involves using prototypes, or simulations. Regarding industrial usage, 5 solutions (6%) are commercial, 11 (14%) have industrial partners or received support from industry, 8 (10%) have one or more authors from industry, and 5 (6%) reported having their product running in a company. The remaining 49 solutions (63%) do not clearly state their relation with industry.

Regarding the service model, 26 solutions (33%) are intended for IaaS, whereas 7 (9%) for PaaS and 1 (1%) for SaaS. There are also 12 solutions (15%) that target more than one of these three service models, and 8 solutions (10%) which target other service models, such as Data-as-a-Service. It is worth noting that 24 solutions (31%) do not specify a target service model. Some solutions, such as [85] and [229], show examples related to SaaS and IaaS (respectively), but the authors did not make clear if this is the target. Other solutions, such as [86] and [18], have some relation with well-known solutions. However, we cannot infer the intended service model from the description of the solution. One interpretation might be that these solutions aim to be generic approaches that target all service models.

Identification and Analysis of Research Area It is critical to any researcher in a new area to identify the most relevant sources of material. In addition, writing and publishing results are commons task in research. To support these activities, we show in the Table 2.1 the journals and conferences most targeted by researchers from this area. We also analysed the relationships between studies, by examining their citations. Our analysis allows us to reason about the impact of a solution. The OCCI project, for example,

Table 2.2: The most cited solutions for avoiding cloud lock-in in our systematic mapping.

Name	# Citations
Eucalyptus[1]	22
Open Virtualization Format[2]	18
Open Cloud Computing Interface[3]	17
OpenNebula[4]	15
Reservoir Project [196]	9
Nimbus[5]	8
Libcloud[6]	7
Delta-Cloud[7]	7
InterCloud [110]	7
OpenStack[8]	6

is cited as a related work [147], [73], as an example [186] and as a part of a solution [199]. Table 2.2 shows the most cited solutions. Building on well-established solutions is a potentially promising approach to devising a sound new solution, as evidenced by recent work [244].

Gap Analysis To assist in identifying gaps in existing work, the solutions were classified within a Computer Science area, according to the devised artefact, using the 10 of the 12 categories of the first level of the ACM Computing Classification System (Table 2.3). Two general categories have been disregarded: *General and reference* and *Proper nouns*. Existing solutions cover most of the ACM categories, except for areas such as Applied Computing and Theory of computation.

2.2.5.2 Discussion

By performing a thorough analysis of the solutions identified in our systematic review, we made three key observations. Firstly, regardless of the solution, a common approach is to create an abstraction layer that seeks to hide the differences between cloud providers. In turn, adapters are used to support the interaction between the abstraction layer and the target cloud.

Secondly, existing solutions provide alternative ways of tackling the same issue(s). This is the case for APIs and ontologies, for instance. A prerequisite for proposing an API is the identification of entities that the API will deal with [186]. Likewise, designing an architecture is a precondition to propose a platform in [179] and [257]. However, reusing or extending existing solutions is not common practice, although we have iden-

Table 2.3: Distribution of cloud lock-in solutions according to CS area.

Computer Science Area	(%)
Software and its engineering	72%
Information systems	18%
Computing methodologies	5%
Social and professional topics	3%
Human-centered computing	1%
Security and privacy	1%

tified a strong relation between some approaches. An exception is the use of standards. For example, the OCCI standard is used by other solutions (e.g., OpenNebula[1] and Eucalyptus[2]).

Finally, solutions of the same type are very similar, with only slight differences. For example, analysing three platforms for cloud management (i.e., OpenNebula, Eucalyptus, and OpenStack[3]), we could not identify any particular difference which could recommend one over another. The main difference in APIs, used to programmatically access cloud services, is the technology supported: Java (jclouds[4]), Python (Libcloud[5]), and REST (Deltacloud[6]).

From a research point of view, existing solutions on cloud lock-in lack empirical methods to conduct their studies and empirical evaluation. Empirical evidence is critical to evidencing the problem as well as to supporting the solution. From the 55 solutions published in peer-reviewed journals and conferences, only one study [6] presents an explicit research question and no studies present the method for conducting the study. This statistic considers only research papers as such information is expected from this kind of document. In addition, less than one third of the studies included any form of evaluation. It is common that a solution is proposed based on hypothetical scenarios or requirements derived from a brief analysis of current cloud providers.

Cloud computing is built on established concepts, such as distributed computing and virtualization. However, we found that solutions do not take advantage of approaches previously devised in areas related to cloud computing. Analysing the issues and solutions for vendor lock-in, portability, and interoperability in areas related to

[1] `http://opennebula.org`
[2] `http://www8.hp.com/us/en/cloud/helion-eucalyptus.html`
[3] `https://www.openstack.org`
[4] `http://jclouds.apache.org`
[5] `http://libcloud.apache.org`
[6] `https://deltacloud.apache.org`

cloud computing might save time and effort, and also support the cloud solutions by providing a sound foundation for cloud lock-in solutions.

Considering that cloud providers, such as Amazon and Microsoft, are also concerned about portability and interoperability, much may be gained by increasing collaboration with industry. In our systematic review, we have identified that 37% of all proposed solutions have some relation with the industry. IBM and HP have demonstrated their concerns by recently announcing their support for OpenStack, whereas Amazon and Microsoft have developed their own strategies for enabling virtual machine portability.

Finally, a further challenge is that solutions should reflect – and be adaptable to – the social and technical contexts in which they are applied. When migrating applications (not necessarily to make use of cloud), companies conduct a series of activities, such as analysis, test, and training (Section 2.3). The process is cumbersome, usually taking a long time to be completed. We believe that the migration process is similar when migrating applications to the cloud, or migrating cloud applications to a different cloud platform. Thus, we argue that there is a need for solutions to portability, interoperability and vendor lock-in issues to also address socio-technical and business challenges.

2.3 Software Migration for the Cloud

Vendor lock-in affects the application migration in cloud [173]. Therefore, by improving the portability of cloud applications (i.e., the degree of effectiveness and efficiency of a migration) we can reduce the risks of cloud lock-in. This section clarifies some decisions made in this research by analysing reports of real experiences of application, infrastructure and technology migration in scenarios that involve migration related or not related to the cloud. In addition, our analysis considers recent studies that provide guidelines for migration to the cloud.

To select relevant literature, we searched digital libraries covered by the ACM, IEEE, and Web of Science search engines. From 132 papers retrieved, nine primary studies reported real software migration experiences. In addition, we analysed the references of these nine primary studies to select further material. Finally, other relevant studies identified during this research were also selected for analysis, resulting in a total of 21 primary studies, of which 15 report real software migration experiences and six offer some guidelines for migration. Our analysis considers five critical issues emphasised in the primary studies: (i) migration process (Section 2.3.1); (ii) motivation to migrate (Section 2.3.2); (iii) effort (Section 2.3.3); (iv) involved stakeholders (Section 2.3.4); and (v) the need for re-engineering (Section 2.3.5). Each of the following sections

presents our findings regarding non-cloud migration studies first, and then studies on the migration to the cloud.

2.3.1 Migration Process

We identified four main phases for the application migration process in the primary studies (Figure 2.3): (i) analysis; (ii) pre-migration; (iii) migration; and (iv) post-migration. The analysis phase is the only phase that is reported in all studies. This phase involves the analysis of the current application, its requirements, and the migration target. As different studies report different types of software migration, we use the terms *"source"* and *"target"* to refer to the application, infrastructure, platform or technology being migrated, prior and after the migration, respectively. In one migration, a committee was set up to carry out the analysis [166]. The requirement analysis comprises both functional [254] and non-functional [23, 230] requirements. Most studies [23, 166, 216, 230, 238, 254] emphasise the analysis of the migration target, which suggests that this is essential in this phase.

In the pre-migration phase (Figure 2.3, 2), organisations conduct activities such as contract establishment [254], migration planning [230], and responsibility division among the stakeholders [254]. Although only two studies report this phase, migration planning is cited as a success factor for migration [230]. In fact, Suvorov et al. [225] identified planning and stakeholder engagement as critical factors for success in the migration of building systems in open source software projects. In addition, developing an implementation plan is a common activity when migrating software systems to modern systems in libraries [102]. Hallmark & Garcia [102] point out that contracts play a key role in both supporting the relationship between the organisation and its software provider, and ensuring that the expectations of all parties will be met. These activities prepare the organisation to conduct the migration in the next phase.

During the migration phase, the application is transferred from its source to its target. The primary studies provide little information about activities carried out in this phase. However, three activities are noted (Figure 2.3, 3): (i) data conversion [166, 254]; (ii) modifications in the application or business process [166, 216]; and (iii) testing [23, 230, 238]. Data conversion is a common activity when migrating from one system to another, such as in experiences reported by Wilson [254] and Nfila, Dintwe & Rao [166], and in the migration of legacy systems to new platforms, such as in experiences described by Teppe [230] and Sneed & Erdoes [216]. Three experiences [166, 216, 254] report data conversion as a problematic issue, which suggests that it is a

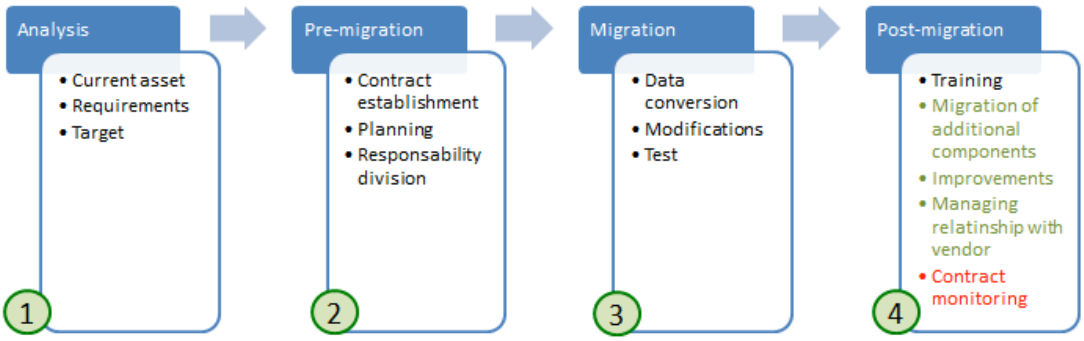


Figure 2.3: Non-Cloud general migration process.

critical activity. Modification refers to any changes necessary to perform the migration. Sneed & Erdoes [216] report their experience refactoring the code before starting a legacy system migration. However, modifications do not refer only to technical, but also to business aspects. Nfila, Dintwe & Rao [166] report a library system replacement, whereby some changes in the library process were required.

Tests are reported by three studies that involved: (i) a platform change [23]; (ii) a legacy system migration [230]; and (iii) an architecture change [238]. Suvorov et al. [225] underline several problems faced by the KDE[1] team, a community of free software developers, when adopting a new build system, suggesting that they failed due to the lack of testing before adopting a new technology.

Post-migration is the last phase identified. Although this phase typically comprises only one activity, staff training, studies point out several other potential post-migration activities (Figure 2.3, 4): the migration of related systems [136], improvements or adjustments of components [23], and relationship management with vendors [102]. In addition, in other areas, such as electronic contract [17], electronic negotiation [57] and service acquisition [218], contract monitoring is a common post-migration activity. Nfila, Dintwe & Rao [166] identify adequate training as a success factor for the migration. Indeed, training is central to supporting system-to-system migration [102], and legacy system migration [230]. Hallmark & Garcia [102] highlight the importance of training, advising its inclusion as a clause in the contract. Staff training is also important before the migration phase. For Aversano et al. [23], having experienced staff is a success factor for the migration.

The migration process to the cloud differs from non-cloud migration in that the former focuses on the entities involved in the migration whereas the latter concentrates

[1] `http://www.kde.org`

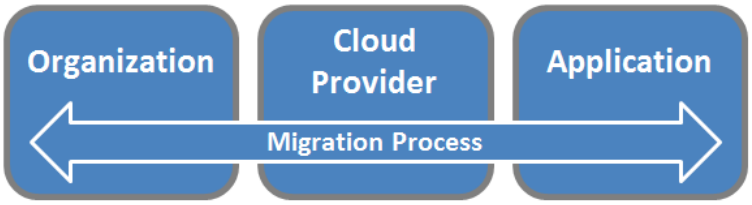


Figure 2.4: Entities in the process of migration to the cloud.

on the migration activities. Studies on the migration to the cloud highlight three main entities involved in the migration (Figure 2.4): organisation, cloud provider, and application.

Most studies on migration to the cloud focus on the analysis phase [16, 32, 53, 125, 232, 232, 263]. The emphasis of these studies on the analysis phase suggests that this is as critical for cloud migration as it is for the non-cloud. The pre-migration phase plays a key role for the migration to the cloud, especially for studies reported in [19] and [232]. The purpose of the activities in this phase is to adapt systems and prepare their users prior to the migration [19]. However, some activities are not mentioned, such as the contract establishment. This is surprising since the first step to use a cloud provider is signing a service contract. As Gagliardi & Muscella [91] point out, issues regarding the data deployed on a cloud must be addressed in a contract.

Beserra et al. [32], Zhou [263], and Chauhan & Babar [53] describe the migration phase as a single activity. In contrast, Tran et al. [232] present three activities: (i) code modification, which concentrates on making the required adjustments to the software to adapt it to the target cloud platform; (ii) migration, which involves system and data migration; and (iii) testing, which considers integration tests between the application and the database system, and application functionality tests. Like the non-cloud studies, little information is provided about this phase. Finally, the last phase, post-migration, is an evident gap left by the primary studies on migration to the cloud since no information is provided about any activity after the migration phase.

2.3.2 Motivation to Migrate

The non-cloud migration reports present a short motivation list. The need for modernisation is the most cited reason, which is typically due to the benefits of a modern target [216, 238], or to the deterioration of current technology [230]. The need for implementing new requirements or achieving more flexibility is the second most cited reason, whereas vendor independence and cost reduction are also often cited. Achieving vendor

independence to reduce costs is a common goal for old technologies (e.g., COBOL and mainframe based systems) [23, 136, 216].

In contrast to the non-cloud motivation list, the cloud literature lists multiple reasons for migrating an application to the cloud, including cost reduction, rapid scalability, and on-demand resource provisioning. Indeed, analysing the cloud literature, it would be easy to compile a long list of benefits to move to the cloud (see [47] and [169]). These benefits apply to multiple cloud stakeholders, including cloud users [52], cloud providers [19], and application users [263]. For example, Khajeh-Hosseini, Greenwood & Sommerville [123] calculated that their system infrastructure would have cost 37% less over a five-year period if it had been migrated to the cloud. In fact, cost reduction is constantly cited in the cloud literature as a benefit for both cloud users and cloud providers [44, 261].

2.3.3 Migration Effort

The effort to perform non-cloud software migration varies from six weeks [238] to two years [136]. System replacement (i.e., an old system is replaced by a new one) took slightly more than one year to complete [166, 254]. The longest migration process, a platform migration reported by Lancia, Puccinelli & Lombardi [136], took two years. Reports present different strategies to speed up the migration process, such as wrapping old components [23] and partial re-implementation [238]. We could identify a correlation between the number of activities (i.e., the process size) and the migration effort in the primary studies. In addition, unexpected issues during the migration also increase the effort [166, 254].

Apart from Tran et al. [232], studies on migration to the cloud do not report the migration effort. Tran et al. [232] took 22.5 hours to prepare a reference application for migration from an on-premises infrastructure to Windows Azure Cloud Services (PaaS). In addition, the authors underline that they took extra 36.5 hours to conduct the migration. However, their experience did not consider any activity in the analysis phase. Moreover, they migrated a relatively small application (PetShop, amounting to 118 adjusted function points).

2.3.4 Stakeholders Involved in a Software Migration

Non-cloud migration reports show that the migration involves more than one person [23, 166, 230, 254]. The organisation's staff play roles on strategy implementation [230, 254], training [166], and system re-engineering [23]. Staff engagement is an important success

factor [166, 225, 230]. Keeping staff well informed during the process is also important. In [166], a successful migration experience, the authors underline, *"It was important to inform staff at every stage of the process since they were going to be the implementers of the system."* Hallmark & Garcia [102] identify different strategies to keep the staff aware of the process status, including the use of newsletters, verbal communication and meetings.

In addition to company staff engagement, support and training from third-party experts also contribute to making the migration process more efficient [23, 166, 230, 254]. Software users were also involved in the process as they were affected by the changes [23, 254]. For example, Nfila, Dintwe & Rao [166] report the dissatisfaction of users with the old system (TINLIB). In addition, the authors report that the committee responsible for the migration analysis carried out interviews with system users, collecting opinions and suggestions. Finally, to manage people and tasks, companies established managers [230], a coordinator [23], or dedicated committees [166].

Reports on the migration to the cloud provide little or no information regarding external stakeholders. This may be due to the fact that the reported experiences have been carried out by researchers as early experiments [52, 263]. The process proposed in [32] suggests that migration involves developers, responsible for making technical decisions, and business and financial analysts, liable for defining and evaluating organisational constraints. Tran et al. [232] indicate that the capabilities of a team – including their existing knowledge of and experience with cloud providers and technologies – is a factor that influences the cost of migration. Finally, recent surveys [47, 169] have identified that decisions related to cloud adoption are not restricted to the IT department, but reaches other departments, with significant participation from business stakeholders.

2.3.5 Re-engineering to Migrate

Apart from Nfila, Dintwe & Rao [166], and Wilson [254], who report a system replacement rather than a system modification, re-engineering is a common activity for non-cloud migration experiences. Teppe [230] constantly highlights serious concerns about changing the functionality of software during migration, and assigns the project success to the decision of avoiding non-planned software changes.

Migration experiences revealed different re-engineering strategies, such as wrapping business components [23] and partial re-engineering [238]. Re-engineering activities were also undertaken in different stages, such as prior to and during the migration [216, 230]. Several problems are reported during the re-engineering, such as software

incompatibility [23] and strong component dependencies [216]. Finally, re-engineering a software architecture is reported as a problematic activity [106]. Re-implementation might be a better alternative when the migration target is a new technology [136].

Andrikopoulos et al. [15] conclude that migration to the cloud involves different re-engineering levels depending on the migration type. Tran et al. [232] describe modifications to adapt the application according to the operating system and back-end applications (e.g., database and programming language) offered by the cloud provider, whereas Zhou [263] changed the application to adapt to the cloud provider architecture. Chauhan & Babar [52] carried out two main modifications: the creation of a persistence layer, and a controller to manage multiple VM instances.

2.4 Cloud Portability

Unlike previous studies [98, 148, 184, 210], our research addresses cloud portability from the perspective of software quality (SQ). We consider portability as a quality attribute (QA) of cloud resources. We took this stance after analysing existing research on portability in general. We understood that portability concepts and challenges are similar regardless of the area where portability is required. Therefore, we hypothesised that our research objectives could be achieved by using accumulated knowledge and applying established SQ techniques for investigating QAs to cloud computing.

This section reviews concepts of SQ, portability and cloud portability. Firstly, we highlight the importance of non-functional requirements and explain their relationship with SQ (Section 2.4.1). Secondly, we present the role of quality models in defining, assessing and predicting QAs (Section 2.4.2). Next, we provide a definition for portability and underline the importance of portability as a QA (Section 2.4.3). Then, we present our definition of cloud applications and list critical activities for application migration in the cloud (Section 2.4.4). Finally, we present a motivation scenario used in this research and its requirements (Section 2.4.5).

2.4.1 Portability as a Software Quality Attribute

According to Sommerville [220], software quality management is concerned with *"ensuring that the required level of **quality** is achieved in a software product."* Sommerville also adds *"quality, simplistically, means that a product should meet its **specification**."* Still according to Sommerville, the software specification can refer to functional or non-functional (NFR) requirements. NFRs are *"constraints on the services or functions*

offered by the system" [220], such as performance and usability. Because NFRs affect the whole system [12], they are often considered more important than functional requirements [220]. This research focuses on a technical NFR of cloud applications – portability.

A survey of 12 organisations shows that performance, usability and security are considered the most important NFRs for software architects [12]. In this survey, portability is one of the less cited NFRs, along with scalability, modularity and monitoring. Surprisingly, these less cited NFRs are critical for cloud computing (Sections 2.1.1 and 2.2). The lack of portability, scalability, modularity and monitoring in legacy applications explains why legacy applications often require re-engineering to take advantage of cloud benefits [15, 52, 125, 186].

Dealing with NFRs is a challenge [12] because NFRs impact software architecture [29] and adopted technologies [11]. In addition, NFRs conflict not only with each other [29, 93], but also with functional requirements [220]. Furthermore, as Galster & Bucherer [93] explain, dealing with NFRs in highly distributed environments is even more challenging due to the influences of different services and stakeholders. For example, ensuring performance is complicated as the overall performance of a task might be compromised by an external service that suffers from unknown performance issues.

Note that although a NFR typically places a constraint on a QA (e.g., an NFR may read: "the response time (a QA) must not exceed 500ms"), NFRs and QAs are often used interchangeably [11, 12, 29, 93, 220]. This thesis adopts only the term QA from this point onward.

2.4.2 Quality Models

Quality Models (QMs) are used to (i) define, (ii) assess and (iii) predict QAs [246]. The most common definitional QM is the ISO/IEC 9126[1] [174], recently replaced by the ISO/IEC 25010[2]. Adopting a definitional QM is essential when working with QAs as they are often subjective [93] and different stakeholders may have different views for the same QA [12]. A prediction model is another type of QM [246] that is discussed in detail in Appendix C.

To evaluate a QA, an assessment model can be used. Evaluating QAs is critical to ensure that the application achieves the desired quality level [29]. Assessment models extend quality definition models (e.g., ISO 25010) by proposing metrics that evaluate

[1] http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749
[2] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

QAs [246]. However, a common approach to evaluate QAs is not by using quality definition models, but by testing causal or correlational relationships [40] (i) between the QA and software characteristics, such as software design properties [201] and development paradigm [145], or (ii) between the QA and its related process, such as a software development or maintenance process task (e.g., testing, fixing bugs) [90].

However, evaluating QAs is difficult because they depend on multiple environmental factors [62], which substantially increases the set of variables to be analysed. As an example, software performance cannot be measured without first quantifying the processor and memory attributes of the computer on which the experiments are carried out. One strategy to isolate variables and measure the causal relationship between the variable of interest and the QA is by using experimentation [40].

This strategy has enabled the study of QAs such as maintainability and changeability, improving the understanding of the software characteristic under study [21, 112], and allowing its measurement and prediction [129, 194]. Testing causal relationships by experimentation is extensively used in Chapters 3, 4 and 5 to identify factors that impact cloud application portability. For the interested reader, Appendix B summarises basic concepts of experimentation in software engineering.

2.4.3 Portability Overview

Portability is an important QA of software systems [29] defined by the ISO/IEC 25010 as the *"degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another."* In line with the real software migration experiences analysed in Section 2.3, this definition shows three elements that must be considered to ensure the success of a software migration[1]: the software to be migrated, and the source and target environments.

Focusing on code portability, Spinellis [223] points out four main benefits of high portability:

- Flexibility to select technologies based on quality and price rather than based on constraints imposed by vendors;
- Ability to modernise software projects by migrating to new technologies;
- Freedom to negotiate for a better price and service with different vendors;

[1]The term *migration* is broadly used in the literature of software, APIs and component migration. Therefore, we prefer this term rather than *transfer*.

- Reduced risks due to little dependency of proprietary technologies.

It is important to note that these benefits are key to prevent vendor lock-in (Section 2.2). According to Bass, Clements & Kasman [29], software portability can be achieved by *"minimising platform dependencies in the software, isolating dependencies to well-identified locations, and writing the software to run on a 'virtual machine' that encapsulates all the platform dependencies within."* In addition, the use of widely adopted standards can increase portability [36].

Despite its importance and benefits, portability is often overlooked by software architects [12]. Improving portability quite often requires re-engineering [46, 226]. According to Chikofsky & Cross [56], re-engineering is the *"examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form."* Re-engineering involves several risks, such as failing to attain quality goals and performance loss [217]. Furthermore, re-engineering is a difficult process [243] whose whereby the overall cost depends on the type of changes [220].

However, improving portability is not just a matter of re-engineering. Like most QAs, portability might conflict with other QAs [93]. In particular, portability often conflicts with performance [29]. For example, although adding layers is a common strategy to improve portability [29], a recent study identified a higher latency when using this type of solution for cloud portability [111]. Therefore, it is critical to analyse in advance the benefits and trade-offs of improving portability in order to ensure that its costs pay off.

Despite an extensive search of the research literature, we have found no empirical studies on portability. To the best of our knowledge, the empirical studies reported in Chapters 3, 4 and 5 are the first empirical work addressing portability. To support this research, we based our studies on portability-related QAs, including changeability [21] and maintainability [183, 194, 200]. In addition, we could find preliminary theoretical work on portability, such as the work of Dhama [65], which correlates portability with cohesion and coupling, and the work of Bass, Clements & Kasman [29], which investigates QAs, including portability, from the perspective of software architecture.

2.4.4 Cloud Application Portability

Cloud portability is a QA that measures the portability of cloud resources. As presented in Section 1.1, examples of cloud resources are data, applications, components, workloads, VMs, configurations, and live deployments. The management of this QA is critical in overcoming cloud vendor lock-in. To investigate cloud portability, one should

take into consideration [2, 15, 97, 184, 186]: (i) the migrated resource, (ii) migration goals, and (iii) the source and target environments. In this section we concentrate on defining cloud applications, presenting a high-level set of activities to migrate cloud applications and highlighting requirements for the scenario investigated in this research.

We analysed 86 research papers that explicitly deal with cloud applications in our reference database, but we could not find a clear definition for *cloud application*. Therefore, for the purpose of this research, and based on the study of these 86 research papers, we define a cloud application as a *"software system that uses cloud services by interacting with them directly (e.g., by using cloud APIs to retrieve some files from a storage service) or indirectly (e.g., by being deployed in a VM running within a computing service). These systems might be cloud-native or not, and recently developed or legacy."* Based on this definition, legacy web applications deployed in cloud services *are* cloud applications.

As presented in Section 2.3, migration activities might vary. We could identify the following critical activities for the migration of cloud applications [15, 53, 97, 134, 185, 242]:

- Application re-engineering;
- Re-creation of resources;
- Re-configuration of resources; and
- Re-deployment.

It is important to emphasise that some of these activities are not always required as they vary according to the migration goals.

2.4.5 Application Migration Scenarios and Their Requirements

Focusing on legacy cloud applications, this section presents two deployment scenarios and discusses requirements to reduce the cloud migration effort by improving cloud portability. Improving portability of legacy cloud applications when migrating an application between the two scenarios is the type of migration investigated in this research.

Figure 2.5 illustrates the two deployment scenarios (A and B), which are based on Amazon Web Service reference architectures[1] and Cloud Computing Use Cases [2]. According to Andrikopoulos et al. [15], scenario A represents a common deployment

[1] `http://aws.amazon.com/architecture/?nc1=f_cc`

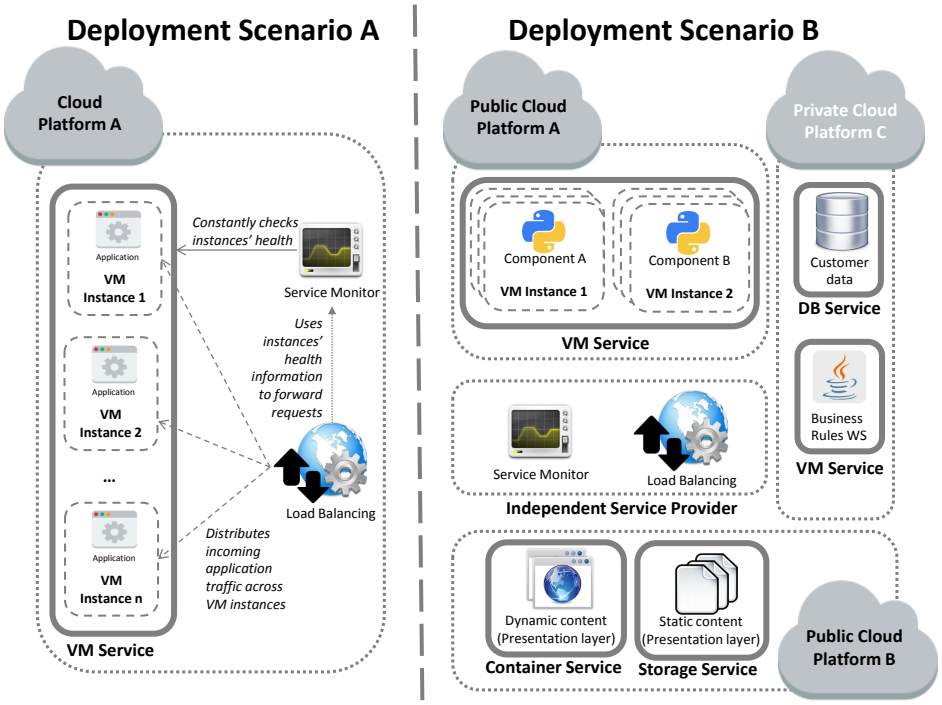


Figure 2.5: Cloud application deployment scenarios.

for cloud applications. Each VM instance consists of the entire application stack, which may or may not include the data tier. A cloud monitoring service is responsible for checking the health of VM instances, which is used by the load balancing service to redirect user requests to healthy instances.

Taking into account the architecture of scenario A as both source and target of a cloud application migration between clouds, the migration effort can be reduced by using solutions for *resource* [199] and *service configuration* [33] *portability* rather than *application portability*. A portable VM image enables the smooth migration of the entire stack application to another cloud platform. Analogously, a portable cloud service configuration enables the creation of similar cloud resources on the target cloud platform with minimal effort, but manual or automatic application re-deployment is required for this case. Some standards, like OVF[1] and TOSCA[2], can be used to increase portability (i.e., reduce the migration effort) of this migration [184]. Additionally, either cloud-specific [83, 199] or general-purpose [170, 222] (DevOps) tools can be used to automatically re-deploy the application stack on the target platform.

[1] `http://www.dmtf.org/standards/ovf`
[2] `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca`

The need for higher *application portability* arises when a cloud user wants to take advantage of specialised cloud services from multiple cloud platforms [186], e.g., when migrating an application from scenario A to B (Figure 2.5). In the deployment scenario B, application components are deployed in different services on public and private cloud platforms. Although the use of multiple cloud platforms (as in scenario B) is a current trend for cloud users [77], migrating a legacy three-tier web application from scenario A to B requires the decoupling of its components [101, 114, 161, 186, 242]. To carry out the migration from scenario A to B, the cloud user must be able to migrate a subset of their application components (rather than the entire application stack) from their current platform and service to another (possibly multiple) cloud platform and service, which might include different deployment and service models. Without decoupling application components, OVF, TOSCA, and most existing standards and technical tools cannot support this migration.

Unlike recent developed applications that use microservices to decompose applications into independent and collaborative components that ease portability in hybrid deployments [25, 80, 205], improving the cloud application portability of legacy three-tier web applications for a scenario with multiple cloud platforms and/or services requires the adoption of some middleware to decouple application components [156] and manage their interaction [245]. According to Alonso et al. [9], *"middleware facilitates the interaction between applications across heterogeneous computing platforms."*

Although cloud platforms offer their proprietary middleware (exposed by their cloud APIs) as the main mechanism to interact with their own cloud services [138], these cloud APIs are not suitable for integrating legacy web applications in the cloud (Section 3.2).

2.5 Summary

In this chapter, we reviewed concepts, terminologies and principles of cloud computing, vendor lock-in, software migration, and cloud portability that are key to understanding the work presented in the rest of this thesis. In summary:

- We described the characteristics, and the service and deployment models of cloud computing, and we highlighted their impact on legacy applications and on application migration. Thus, we can understand that legacy web applications migrated to the cloud make limited use of cloud features. We also briefly introduced three cloud-related areas from which the cloud inherits concepts and trade-offs. These areas provide useful techniques for supporting cloud portability.

- We discussed that the difference in the APIs, semantics, and technologies adopted by cloud platforms and providers have been widely accepted as the main causes of vendor lock-in, although current development practices and technologies also contribute to cloud vendor lock-in significantly. This discussion, along with real migration experiences supported the definition of our hypothesis.

- We analysed existing solutions for avoiding cloud lock-in and identified two gaps addressed in this research: (i) the existing cloud portability literature lacks empirical evidence; (ii) existing research disregard existing solutions to similar problems from related areas (e.g., distributed computing); and (iii) socio-technical and business challenges.

- We examined real migration experiences from which we learnt that the engineering effort to carry out a resource migration vary significantly (which impact on the total migration cost). Contrasting these experiences with cloud lock-in solutions, we identified two gaps addressed in this research: (i) solutions for cloud portability mainly focus on the transference of resources from one cloud to another although planning this transference is also important; and (ii) although real experiences on software migration show that the engineering effort to carry out a resource migration vary significantly (which impact on the total migration cost), the migration effort is often overlooked in cloud portability research.

- We presented concepts of software quality, important activities for migrating cloud applications and the migration scenario investigated in this research. These concepts, activities and the migration scenario supported the definition of our hypothesis and guided our work.

In the next three chapters, we present a set of experiments carried out to identify factors that impact on cloud application portability. We use data produced in these experiments to build prediction models for the effort to improve cloud application portability. Each of the three chapter starts with its own short background that clarifies additional concepts used only in the chapter.

Chapter 3

Investigating the Impact of Software Coupling on Cloud Application Portability

Most information systems developed to run on the web (before cloud) use a three-tier architecture [58, 95, 186, 228, 262]. Unlike recent developed applications that use microservices to decompose applications into independent and collaborative components that ease portability in hybrid deployments [25, 80, 205], legacy three-tier web applications were not designed for integration across the Internet [9]. Therefore, the portability of this ubiquitous class of legacy applications in hybrid deployments represents a major challenge [186]. In the software architecture [29, 43, 58] and distributed systems [9, 175] literature, portability is achieved by decoupling application components, which is facilitated by loosely coupled components. In addition, theoretical analyses suggest that coupling may affect portability in general [24, 65], and recent empirical studies have established causal relationships between coupling and portability-related software characteristics including changeability [21] and maintainability [183, 194, 200].

Tanenbaum & van Steen [228] observe that a middleware should be introduced to enable interapplication communication when decoupling application components of distributed information systems. Message-Oriented Middleware (MOM) is a type of middleware that has already been broadly used for integration in service-oriented computing [9, 175]. Furthermore, MOM is presented by Tanenbaum & van Steen as a solution for avoiding the tight coupling imposed by other types of middlewares [228]. Moreover,

message queuing, an abstraction of MOM, is advocated in the cloud literature as an optimal solution to decouple application components and ensure their communication in distributed environments [186].

However, replacing traditional message calls with message queuing in legacy three-tier web applications requires substantial architectural re-engineering [122]. Although re-engineering is a common process to add new quality attributes into software systems [46, 226], Sommerville [220] points out that distributing centralised systems is difficult from a technical perspective. Therefore, it is important to investigate whether the common assumption that low coupling facilitates portability [29, 43, 58] holds true for cloud application portability when replacing message calls with message queuing for decoupling application components.

This chapter empirically investigates the impact of coupling on cloud application portability in the context of re-engineering legacy three-tier web applications to decouple application components by adopting a message queuing mechanism (Section 3.3). We explore the outcomes of this investigation by building prediction models to support decision makers when analysing the required effort to increase the cloud portability of their applications (Section 3.4). Additionally, this chapter complements our literature review by motivating the use and explaining the main concepts of software coupling (Section 3.1) and message queuing (Section 3.2).

3.1 Review of Software Coupling

At the most general level, software coupling is defined as *"the degree of interdependence between parts of a design"*; *"two objects are coupled if and only if at least one of them acts upon the other"* [55]. Coupling has been shown to influence external QAs such as maintainability [183], changeability [21] and modifiability [38]. The use of low-coupled classes reduces the complexity and increases the efficiency of software [43].

To account for the different uses of software coupling, a plethora of coupling measures has been proposed over the past two decades, e.g. by [37, 39, 55, 65, 201]. Given the large number of coupling measures available, we chose a suitable coupling measure for our study based on the unified framework for coupling measurement introduced in [37]. This framework supports *"the comparison and selection of existing coupling measures with respect to a particular measurement goal"* using the following six criteria:

1. *Type of connection* considered by the coupling measure;
2. *Locus of impact*, i.e., whether class c_1 uses class c_2 (client or import coupling) or c_1 is used by c_2 (server or export coupling);
3. *Granularity of measure*, comprising the domain of the measure (i.e., attribute, method, class, set of classes or system) and the method used to count the connections from c_1 to c_2 (e.g., counting separately or only once similar connections such as the invocation of the same c_2 method from c_1);
4. *Stability of server*, which distinguishes between classes that are subject to modification (unstable) and classes whose development was completed (stable);
5. *Direct or indirect coupling*, i.e., whether the coupling measure counts only direct connections (e.g., a method m_2 of c_2 invoked by a method m_1 of c_1) or it also counts indirect connections (e.g., methods invoked by m_2);
6. *Inheritance*, i.e., whether coupling through inheritance, polymorphism, etc. are taken into consideration or not.

As recommended in [37], we use these six criteria to analyse existing coupling measures and to select a measure appropriate for our study. We present this analysis in Section 3.3.1.2. In the next section, we motivate the use and explain the main concepts of message queuing (Section 3.2), which is investigated in this chapter as a means to decouple application components of legacy three-tier web applications.

3.2 Review of Message Queuing

As presented in Section 2.4.5, the use of middleware is key to enabling the interaction of software applications in heterogeneous environments. Although cloud APIs have been widely used as a middleware for integrating *cloud services* [77], integrating *cloud application components* requires another type of middleware, such as a Message-Oriented Middleware (MOM) [156]. A MOM enables the asynchronous interaction of application components, which substantially increases the fault-tolerance since messages are not lost if a consumer/producer service is not available [9, 156]. Asynchronicity is a key characteristic for highly distributed and heterogeneous environments [9]. In addition, MOM has already been broadly used for integration in service-oriented computing [9, 175], which is an area from which cloud inherits key concepts (Section 2.1.3).

Message queuing (MQ) is the most common abstraction of MOM [9]. MQ systems enable that application components exchange data by sending and receiving messages from specific queues [228]. As Goel, Sharda & Taniar [95] explain, MQs consist of four main entities:

- *Queue* is a recipient that temporarily stores *messages* sent to one or more *consumers*;
- *Messages* encapsulate the user data;
- *Producers* originally own the data, create and send *messages* to a particular *queue*;
- *Consumers* represent the target component in an one-to-one or one-to-many component interaction. Consumers monitor one or more *queues*, receive their *messages* and use or forward the data encapsulated in them.

Producer and *consumer* are roles that can be played by the same component in a distributed environment [228]. Java Message Service (JMS) is a widely used standard for MQ that is implemented by JEE-compliant application servers [266], such as Glassfish[1] and JBoss[2]. However, there are several other implementations of MQs, such as Apache ActiveMQ[3], RabbitMQ[4] and more recently, cloud implementations such as Amazon Simple Queue Service[5] and stormmq[6].

3.3 Empirical Investigation

The goal of this experiment is to analyse coupling for the purpose of evaluation with respect to its impact on the effort to increase cloud application portability from the point of view of the software engineers in the context of undergraduate students reengineering legacy three-tier web applications by replacing method calls with message queuing.

We start this section by detailing the experiment protocol and its execution (Section 3.3.1). Next, we show the main results of our experiment (Section 3.3.2). Then, we analyse the impact of our results in our proposed hypothesis, and discuss the implications of our findings (Section 3.3.3). Finally, we examine major threats to the validity of our findings (Section 3.3.4).

[1] `http://glassfish.java.net/`
[2] `http://www.redhat.com/en/technologies/jboss-middleware/application-platform`
[3] `http://activemq.apache.org/`
[4] `http://www.rabbitmq.com/`
[5] `http://aws.amazon.com/sqs`
[6] `http://stormmq.com/`

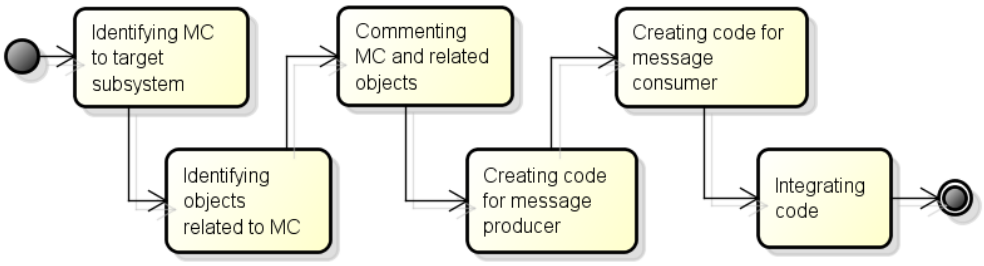


Figure 3.1: Tasks performed by participants to replace method calls (MC) with message queuing.

3.3.1 Experiment Plan and Execution

As explained in Section 1.2, we used guidelines provided in [256] to prepare and conduct this experiment. The next sections correspond to elements of the experimentation framework, detailing the protocol for the experiment, and its execution. Terminology associated with the experimentation framework is highlighted in the main text; this terminology is explained in Appendix B.

3.3.1.1 Task

Replacing method calls with message queuing requires a set of tasks for each Java class that is re-engineered (Figure 3.1). In the first and second steps, participants identified the method calls and objects related to these method calls. This can be achieved by investigating the source code. The programming environment offers mechanisms to support this task. In the third step, participants commented out the actual method calls and their related objects. We encouraged participants to comment out source code instead of deleting it (so that they could refer to the original code more easily).

Next, participants should create code related to the message producer and consumer. In a method call, classes can assume the role of a sender and/or a receiver. Message queuing mechanisms work in the same way, but the sender is called a producer, and the receiver is called a consumer. Both producer and consumer code might be created in different ways. For example, new classes can be created to accommodate the new code or auxiliary methods can be created in the original classes. During the training sessions, participants were taught these alternatives.

Finally, the new code should be integrated with the rest of the system. This involves replacing method calls with code related to producers and consumers, importing libraries and instantiating objects. Although the activity diagram in Figure 3.1 implies

a sequence, participants could perform tasks in any appropriate order. No automation tool was provided to participants other than the development IDE.

Participants were randomly assigned to work on one of two small commercial information systems (Section 3.3.1.3). Within each system, participants selected (at random) pairs of Java classes to re-engineer. Coupling varied over the pairs of Java classes for each of the two systems. Participants were asked to change several pairs of classes while recording the time taken to complete the activity. The time required to set up the environment and analyse the classes was not considered as part of the activity. All participants were able to complete the activity within the allotted time.

3.3.1.2 Selection of Variables

Dependent variable Effort is the *dependent variable* in this experiment. Effort is measured as the time taken T, in minutes, to replace method calls with message queuing for each Java class. This variable is measured on a *ratio* scale, whereby values vary from 0 to the total number of minutes taken to change a given Java class.

Independent variable Coupling is the *independent variable* in this experiment. There are several measures of coupling in the literature (Section 3.1). As there is no empirical work on linking any software design property with cloud portability, we investigated the set of 30 measures presented in [37], and measures adopted in studies that use coupling to evaluate changeability [21], modifiability [262] and maintainability [183].

As a single coupling measure is all that is needed for a preliminary demonstration that coupling can impact on cloud portability, this experiment focuses on a single coupling measure. To select an appropriate coupling measure, we adopted the unified framework for coupling measurement, presented in Section 3.1.

Table 3.1 lists the six criteria of the unified coupling framework, shows the values we selected for this experiment, and summarises reasons underpinning our selection. The Others Method-Method Import Coupling (OMMIC) metric [39] was the coupling measure used in this experiment because it is the only measure that addresses all values we selected in the unified coupling framework, in Table 3.1.

OMMIC measures the number of invocations (i.e., *ratio* scale) that a method m_c, from a class c, makes to a public method m_d from a class d, given that c and d only have visibility of each other's public interface (i.e., c and d are not the same class, are not part of the same inheritance hierarchy, and – in Java – are not in the same package). Thus, the coupling, represented by the OMMIC value, between a class c and a class d,

is the sum of all public method invocations that are made by methods in the class c to methods in the class d.

Table 3.1: Criteria for selecting a coupling measure defined in [37], values adopted for this experiment and reasons underpinning their selection.

Criterion	Value	Reason
Type of connection	Method	The activity addressed in this experiment is the replacement of method calls with message queuing.
Locus of impact	Import coupling	When replacing a method call, the change is made in the class acting as a client.
Granularity	Class level, count individual connections	Classes are the underlying element of subsystems or layers. Each connection is replaced with message queuing.
Stability of server	Unstable classes	We are only interested in classes in which changes are made.
Direct or indirect coupling	Direct	A method call involves two directly related classes.
Inheritance	Non-inheritance coupling	Our focus lies only on classes that are impacted by a change, i.e. the super class.

Although we adopt OMMIC as a coupling measure for this experiment, possible values for the *independent variable* are defined in a *categorical* scale for the hypothesis testing, varying between *low* and *high* coupled classes. This strategy is preferred because literature related to software development usually adopts a *categorical scale* when referring to coupling. For instance, literature on architectural patterns [43], object-oriented design [137] and modifiability [24] recommends keeping coupling as low as possible, without making any reference to what is considered low or high.

3.3.1.3 Instrumentation

This section details the instruments used to support the execution of the experiment. In particular, we present the two software systems that were changed, and the cloud service used to enable the message queuing implementation.

Software system Two commercial legacy web-based information systems were used in this experiment, namely Siget and GSCloud. As these are commercial systems, their code are not available to the general public[1]. Developed by students as their final

[1]Their documentation can be provided under request to: claudete@unipar.br

project to graduate as Bachelors in Information Systems, these systems are currently deployed in small companies[1]. The two systems differ in purpose and size. Whereas Siget enables sales management of tractor parts, GSCloud enables service management of a web service provider. Table 3.2 summarises both systems.

Table 3.2: Characteristics of software systems used in the experiment.

Size Metric	Siget	GSCloud
Classes	84	43
Methods	623	287
Method Invocations	952	315
Configuration Files	12	13
Web pages	60	40
Lines of Code	4,793	1,917

Although there are many open source systems that could be used in this experiment, they often adopt a set of technologies that are not well understood by participants. Therefore, they could complicate the experiment. Siget and GSCloud were implemented using JSP and Servlets only – technologies well-known by participants of the experiment. In addition, Siget and GSCloud are small and self-contained applications, rather than large legacy systems that would be difficult for our participants to modify.

The participants were not required to re-engineer the entire system, which would have required substantially more time than was available for the experiment. Instead, participants changed only classes within the *Control* subsystem (*source*) that called methods in (i.e., were coupled with) the *Business* subsystem (*target*). The rationale for this choice is that methods that implement associations between these two subsystems are small and have a well defined purpose, such as adding a new user to the system, thus minimising the complexity of modifications necessary to the application code. No refactoring was applied to any of the selected classes.

As shown in Table 3.3, participants had different options for class selection depending on the software system they were assigned.

Message queuing Adopting message queuing requires a commitment to a particular technology [245]. There is a variety of technologies providing the message queue mechanism, such as application servers and enterprise service buses [9]. In this experiment, we adopted a cloud-based technology. Specifically, we adopted the Amazon SQS[2] cloud ser-

[1] Six employees each, according to the software documentation.
[2] `http://aws.amazon.com/sqs/`

Table 3.3: Candidate Java classes for each software system.

System	OMMIC values	# of Java classes in the "Control" subsystem
Siget	3	11
	4	3
	5	1
	7	2
	8	1
GSCloud	3	2
	4	4
	5	3

vice, and its API that enables the programmatic management of messages and queues. Two reasons prompted this choice. Firstly, traditional message queuing technologies require installing and configuring a software stack, which would complicate this experiment. Secondly, some message queuing technologies are incompatible with some service models, such as Platform-as-a-Service (PaaS). SQS requires no prior configuration or software installation other than the SQS API, which is used for sending and receiving messages, and requires only a few lines of code to be implemented.

3.3.1.4 Participant Selection

A group of nine third-year undergraduate students in computer science took part in this experiment. They were enrolled at the Universidade Tecnológica Federal do Paraná - Campo Mourão, Brazil. The student selection was driven by *convenience*, according to the student interest in participating. Due to the characteristic of the two software systems used as *instruments* in this experiment (Section 3.3.1.3), knowledge of Java programming was the only mandatory criterion to select students. Students had the freedom to choose whether or not to participate, without any risk of penalty to their undergraduate course. Each participant received a gift worth £10 and a certificate of participation.

3.3.1.5 Definition of Hypotheses

To test whether low coupled classes require less effort to modify for the adoption of a message queuing mechanism, we compared the coupling of pairs of Java classes. We identified one pair with high coupled (HC) and another pair with low coupled (LC). Thus, HC and LC represents the two *treatments* considered in this experiment. To

formulate a formal hypothesis, we let T be the time, in minutes, taken to re-engineer the application. Our hypotheses (Table 3.4) compare the time taken to complete the migration tasks for HC ($\tilde{x}\,T_{\mathrm{HC}}$) and LC ($\tilde{x}\,T_{\mathrm{LC}}$) classes. The alternative hypothesis is based on the assumption that *high* coupling might lead to code complexity and modification inefficiency.

Table 3.4: Formal definition of hypotheses.

Null hypothesis (H_0)	$\tilde{x}\,T_{\mathrm{HC}} = \tilde{x}\,T_{\mathrm{LC}}$
Alternative hypothesis (H_1)	$\tilde{x}\,T_{\mathrm{HC}} > \tilde{x}\,T_{\mathrm{LC}}$

3.3.1.6 Experiment Design

The experiment consists of one *factor* (coupling), and two *treatments* (HC and LC classes). To assign participants to the different treatments, we adopted a *completely randomised design*. The two software systems, adopted as instruments, were randomly assigned to participants. To perform the activity defined in Section 3.3.1.1, participants randomly selected classes from the software system assigned to them by choosing classes from a list. This random selection of classes led to an *unbalanced design*. Therefore, it was impossible to define, a priori, the exact number of HC and LC classes that would constitute the resulting sample (n).

3.3.1.7 Data Analysis

As explained in Section 1.2, we adopted only *non-parametric* statistics in this experiment, as summarised in Table 3.6 whereas cutoffs we used for inferential tests are summarised in Table 3.5.

Table 3.5: Cutoff values defined for evaluating the hypotheses, adapted from [70].

		Unknown true state of nature	
		H_0: $\tilde{x}\,T_{\mathrm{TC}} = \tilde{x}\,T_{\mathrm{LC}}$	H_1: $\tilde{x}\,T_{\mathrm{TC}} > \tilde{x}\,T_{\mathrm{LC}}$
Statistical Conclusion	Accept H_0	0.95 1 - α: Correct *(Confidence level)*	0.64 β: Type II error
	Reject H_0	0.05 α: Type I error *(Significance criterion)*	0.36 1 - β: Correct *(Statistical power)*

Table 3.6: Summary of statistics and statistical tests in the experiment.

Statistic Type	Statistic/Test	Definition/Config.	Purpose
Descriptive	Median and quartiles	25%, 50%, 75%, 100%	To identify the middle value and categorise occurrences into quartiles according to their values.
	Range	Maximum - Minimum	To quantify the data dispersion.
Inferential	Shapiro-Wilk	Two-tailed, $\alpha = 0.1$	To test the assumption that the data is normally distributed.
	Wilcoxon rank-sum	One-tailed	To test the statistical significance of differences between two medians.
	Jonckheere-Terpstra	One-tailed	To test the assumption that the higher coupling, the higher the time spent.
	Kruskal-Wallis	One-tailed	To test the statistical significance of differences between several medians.

3.3.1.8 Experiment Execution

The experimental design was refined after a pilot study involving an independent set of participants enrolled at a different university, UNIPAR - Paranavaí.

The experiment was carried out in two parts: training and experimentation. Each part was divided into two four-hour sessions. The first training session focused on cloud and message queuing concepts and technologies. The second training session prepared participants to perform the class modification required for this experiment. A prototype application was used during the training sessions. Both the training and experimentation sessions were carried out in the same classroom, using Linux desktop computers, Java EE and Netbeans IDE.

In the first meeting, participants were informed of the purpose of the experiment, the activities they were to perform, and the benefits of participating. We also required each participant to complete a consent form and the *participant characterisation form* in order for us to determine the participants' background (such as Java expertise). Data collected from participants was anonymised. Participants were asked to record the time taken to complete activities (not including any breaks).

During the experiment, participants were free to create new code and update existing code. No guidelines or reference materials were given to support participants in the reengineering activity. The experimenter and a monitor observed the experiment without interfering in tasks performed by participants. It was necessary further explanations to participants during the experiment session due to their lack of experience with cloud services and message queuing. Participants were strongly advised to not copy and paste any code, as doing so could significantly reduce the modification time.

Participants completed a *feedback form* at the end of the experiment, and received vouchers and participation certificates a few weeks later. For information on ethics in this experiment, refer to Section 1.3.

3.3.2 Results

In this section, we present the results of the experiment to test the hypothesis that coupling impacts on the effort to increase cloud portability.

3.3.2.1 Sample Distribution

Participants were able to change 63 classes (*sample size*) representing four out of five OMMIC values present in the classes changed. Table 3.7 summarises the number of classes (*data points*) for each OMMIC value. Classes were not homogeneously selected by participants. Some participants changed more classes than others. Classes with low OMMIC values were those that participants most selected. This preference might be due to the greater number of classes with low OMMIC values, and by the ease of modifying these classes when compared with classes with high OMMIC values. As each OMMIC value represents a treatment group in this experiment, we tested the data normality for each group by applying the *Shapiro-Wilk* normality test. Whereas for groups 1 and 2 the p-value indicates non-normal distributions (p-value<0.1), for groups 3 and 4 it indicates normal distributions. As noted in Section 3.3.1.7, we therefore chose to adopt only non-parametric measures for data analysis.

Figure 3.2 shows the data point distribution along axis of OMMIC value and effort. The chart shows some possible outliers. For example, a class with OMMIC value of 4 took nearly 85 minutes to change. Some extreme values might be explained by the order in which classes were selected. Analysing the data set, we note that the first classes changed took participants longer regardless of the OMMIC value. This is likely to be because participants take some time to find the optimal way to implement changes. As we had not planned a data reduction strategy, we opted to retain outlier data points.

Table 3.7: Normality test for the four treatment groups.

Group	OMMIC values	Data points	Normality test	
			W	p-value
1	3	28	0.8814	0.0043
2	4	20	0.6955	<0.001
3	5	11	0.9295	0.4058
4	7	04	0.8836	0.3541

Table 3.8: Descriptive statistics. Values represent the effort, in minutes.

Measures	OMMIC Group			
	3	4	5	7
Min	6.0	6.0	9.0	9.0
1st Quartile	10.8	13.8	25.5	28.5
Median	11.0	16.5	29.0	42.5
Mean	14.5	22.3	36.6	36.8
3nd Quartile	18.5	21.0	44.5	50.8
Max	32.0	84.0	80.0	53.0
Std. Deviation	7.1	17.9	21.2	20.1
Skewness	1.0	2.3	0.8	-0.7
Kurtosis	3.2	8.2	2.7	1.9

3.3.2.2 Descriptive Statistics

The boxplot in Figure 3.3 enables the analysis of the median time to change classes taking the four OMMIC values into consideration. Here, the high variation observed in Figure 3.2 is quantified. The boxplot also contains six outliers. We can observe an increasing trend in the medians across OMMIC values, suggesting a relationship between coupling and effort. Table 3.8 summarises the descriptive statistics illustrated by the boxplot.

The median time varies across the OMMIC values. An unexpected result is that the maximum effort for OMMIC groups 4 and 5 is far greater than the maximum effort for the other groups. Following further scrutiny of the data, this is probably caused by two participants who started the experiment by selecting their first classes with OMMIC values of 4 and 5. As noted before, the first classes that participants re-engineered took more time to be changed than later classes.

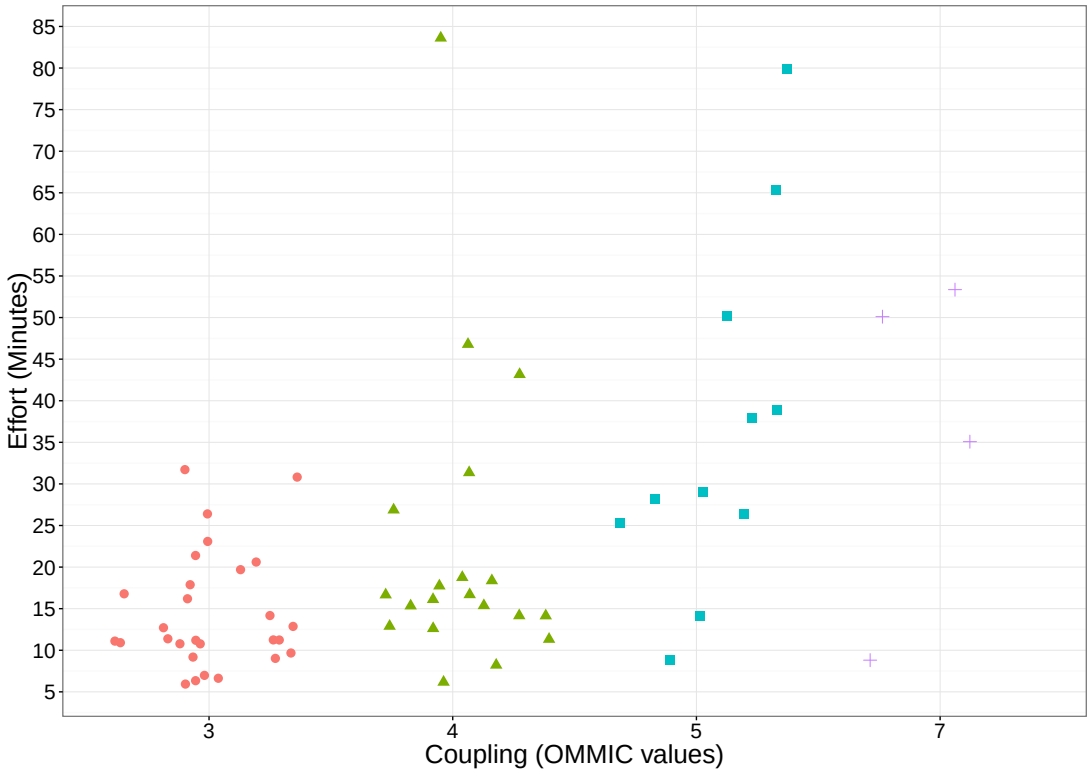


Figure 3.2: Data point distribution. Some possible outliers can be observed.

Table 3.9: Results of the *Kruskal-Wallis* test. *P*-value is less than the α, indicating a statistically significant difference between OMMIC groups.

Chi-squared	Degrees of Freedom	P-value
15.1799	3	0.001669

3.3.2.3 Inferential Statistics

We started our analysis by applying the *Kruskal-Wallis* test to identify whether there is a statistically significant difference of effort between OMMIC groups. Table 3.9 summarises the results. As the p-value is less than the significance criterion set up as the cutoff for this experiment ($\alpha = 0.05$), the test shows that there is a statistically significant difference between OMMIC groups. However, this test does not show whether there is a trend of increasing effort in line with increasing OMMIC values. Therefore, we applied the *Jonckheere-Terpstra* test. The result showed a JT = 971, and a p-value $<$0.001, which confirms the trend observed in Figure 3.3.

Although *Jonckheere-Terpstra* tests the increasing trend across OMMIC groups, it

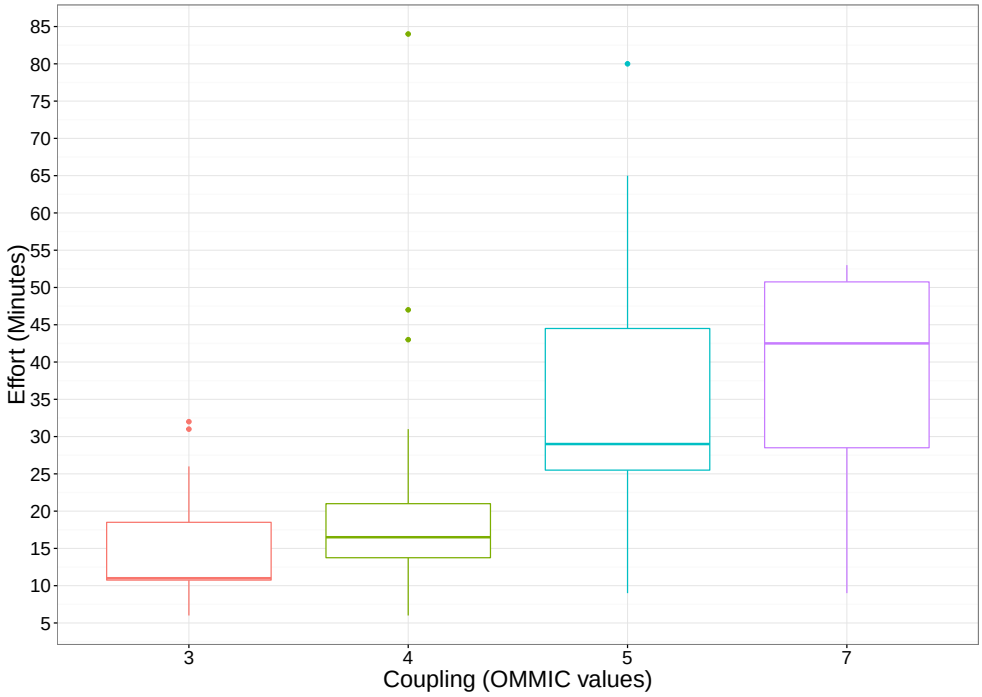


Figure 3.3: Boxplot shows an increasing trend in the medians across OMMIC values.

does not show independent group comparisons. Therefore, we applied *Wilcoxon* to test the statistical significance of differences between group pairs. The OMMIC group with value equals to 7 was excluded from this comparison as its sample size (i.e., 4) was below the cutoff defined in Section 3.3.1.7 for an acceptable statistical power. Table 3.10 presents the result of OMMIC group comparisons.

As discussed in Section 3.3.1.2, we grouped OMMIC values into groups of two, representing low and high coupled classes. This resulted in three comparisons. The sample size (n) corresponds to the sum of data points in each OMMIC group. The p-value calculated for each group is less than the adjusted α (see Section 3.3.1.7), supporting the rejection of the null hypothesis. The γ is higher than that defined in the *a priori* analysis for all groups, and the *post hoc* power is above the average for SE experiments[70] for all comparisons.

3.3.3 Discussion

This section discusses the major implications of the results presented in the previous section. We examine the hypotheses and the research question defined for this experiment. We also discuss a way in which these findings can contribute to building theories.

Table 3.10: Comparison of low and high coupled classes. The rejection of the null hypothesis is supported by p-values less than the adjusted α.

#	Coupling		N	Wilcoxon Test		Adjusted α	γ	Power
	Low	High		W	P-value			
1	3	4	48	185	0.0230	0.05	0.32	0.65
2	4	5	31	161	0.0175	0.03	0.42	0.52
3	3	5	39	265	0.0002	0.01	0.58	0.99

3.3.3.1 Hypothesis Testing

In Section 3.3.1.5, we defined the two hypotheses tested in this experiment. Whereas the *null hypothesis* states that the time to perform re-engineering to increase cloud portability does not change, regardless of coupling, the *alternative hypothesis* states that the higher the coupling, the higher the time to perform re-engineering. When different OMMIC values are grouped into low and high coupling, the *Wilcoxon* test confirms that there is a statistically significant difference between the median effort of groups (Group 1: p-value = 0.023, α = 0.05, one-sided; Group 2: p-value = 0.0175, α = 0.03, one-sided; Group 3: p-value <0.001, α = 0.01, one-sided). These tests confirm a statistical difference between median time (Table 3.8). The results support the **rejection** of the *null hypothesis* and the **acceptance** of the *alternative hypothesis*.

A common assumption in the software development literature is that lower coupling is better for portability. We empirically tested whether this assumption could be observed in cloud application portability. Table 3.8 shows that classes with lower OMMIC values took less time to modify than higher ones. The *Kruskal-Wallis* test confirms that there is a statistically significant difference between medians of effort for different OMMIC values (p-value = 0.001, α = 0.05) whereas *Jonckheere-Terpstra* test confirms an increasing trend of medians across increasing OMMIC values (p-value <0.001, α = 0.05, one-sided). For this experiment, the results confirm the assumption that low coupling is better for cloud application portability.

Finally, an important aspect of any statistical result is the effect size of statistical tests. As Kampenes, Dybå & Sjøberg [120] underline, the effect size can be interpreted by comparison to (i) similar experiments, (ii) results from the research field as a whole, or (iii) standard conventions. Due to the lack of effect size information in similar experiments [112, 129, 194, 200], we compare our results with experiments in SE and Cohen definitions as shown in [70] and [120], respectively.

Kampenes, Dybå & Sjøberg [120] calculated the effect size for 429 tests performed

in 92 SE experiments published between 1993 and 2002 in high impact conferences and journals. Based on their calculations, they created a classification similar to that proposed by Cohen. This classification supports the comparison of effect size in SE experiments by comparing the obtained values in a given experiment with the values that [120] calculated. The result of this comparison, is a classification into one of the three possible values: small, medium or large. Table 3.11 shows the effect size calculated for this experiment and its classification according to common values in the research field ([120]) and standard convention (Cohen).

Table 3.11: Classifying γ obtained in the experiment according to the software engineering research field and standard convention (Cohen's).

Coupling Pair	Obtained values of γ	Effect Size Classification	
		SE experiments	Std convention
1	0.32	Medium	Medium
2	0.42	Medium	Medium
3	0.58	Large	Large

Table 3.11 shows the three pairs of coupling, as previously defined in Table 3.10. The coupling pairs were defined by taking pairs of different OMMIC values. The second column shows γ values obtained in this experiment, and third and fourth columns show their classification when compared with the research field and standard convention. The γ calculated for coupling groups 1 and 2 are considered as *medium* for both SE experiments ($0.193 \leq \gamma < 0.456$) and standard convention ($0.30 \leq \gamma < 0.50$). Similarly, the γ calculated for coupling group 3 is considered large for both SE experiments ($0.456 \leq \gamma < 0.868$) and standard convention ($\gamma \geq 0.50$).

3.3.3.2 Research Question

The research question investigated in this experiment is whether software coupling impact cloud application portability. To answer this research question, this experiment took into consideration one out of several activities necessary to increase cloud portability. The rationale is that if the effort for performing the activity investigated in this experiment varies with the coupling, then coupling impacts on cloud portability.

As shown in Table 3.8, descriptive statistics show significant differences between different coupling values. In addition, inferential statistics confirm the statistical significance of coupling differences between low and high coupled classes (Table 3.10). Thus, we can conclude that *coupling does impact cloud application portability.* Hence,

coupling, represented by the OMMIC measure, is an indicator of cloud portability.

3.3.3.3 Devising Theories for Cloud Portability

Supporting the development of theories for cloud portability is one key contribution of this experiment. A theory explains how and why a phenomenon occurs and enables conceptual predictions of the phenomenon under study [104]. Understanding the phenomenon brings benefits to both academia and industry [213]. For example, the development of sound and effective solutions for application migration between clouds is a benefit of understanding the cloud portability phenomenon.

For example, we advocate that a benefit of understanding cloud portability is .

Shull & Feldmann [209] underline that even a single empirical study can be the first step towards building theories. Sjøberg et al. [213] present a conceptual framework for devising theories in SE. The framework consists of four elements:

- *Constructs* define basic elements, such as actor, technology, activity and software system;
- *Propositions* determine the interaction amongst constructs;
- *Explanations* provide the rationale for propositions; and
- *Scope* limits the universe in which the theory is applicable.

Although building a theory is beyond the scope of our objectives, this experiment provides all of the elements for building a theory for cloud portability. For example, coupling and time are two *constructs*. The design of this experiment provides the *proposition* whereas the result provides the *explanation*. Finally, the *scope* can be defined by the context of cloud computing, software systems used and activity performed.

3.3.4 Threats to Validity

This section presents some threats to the validity of the results of the experiment.

3.3.4.1 Internal

Internal threats impact on the cause-effect relationship. These threats might lead to an alternative cause for the effect (known as *confound*) [215]. In this section, we identify three internal threats.

Maturation validity threats refer to the reaction of participants as time passes. As our experiment was longer than many software engineering experiments [215], the participants might have experienced boredom [132] leading to the use of disallowed techniques such as copying and pasting code. To mitigate the effects of maturation, we used participants who showed high commitment to our study, and we carried out the experiment over two sessions organised on two consecutive days.

Participant selection threats are related to the use of participants with different background or level of experience. Although our use of undergraduate students with similar Java programming skills reduced the participant heterogeneity [256], we could not ensure that all participants had the same characteristics. However, the most important skills for the experiment were acquired during training sessions we organized before the experiment. During these sessions, we constantly checked that all participants understood and could achieve similar progress with the re-engineering tasks subsequently evaluated by the experiment.

Another threat is the use of the effort required to modify inter-component communication to message queuing as a measure of cloud application portability. Previous studies confirm that this architectural change: (a) is essential for exploiting the key capability of cloud platforms to support scalability; and (b) represents a major and costly re-engineering task. However, we could not rule out the possibility that higher software coupling eases other major re-engineering tasks that also improve the effectiveness and efficiency with which applications can be transferred across cloud platforms.

3.3.4.2 External

External threats to validity restrict the result generalisation beyond the scope of the study [256]. In this section, we identify three external threats.

First, our conclusions may be affected by using *time as a measure of effort* as in many other software engineering experiments [64, 168, 183] might affect the validity of our conclusions since re-engineering time depends on participant capabilities such as programming skills, ability to understand existing code and typing speed.

Second, the *selection of objects* for our study might impact its generality. Thus, we used only two small commercial information systems and a single cloud queuing mechanism in our experiment. To reduce the likelihood of this threat, we chose systems that share key characteristics with a wide range of applications that can benefit from the change investigated in the experiment, including multi-tier architecture, web-based front tier, and need for high availability. Also, we used the Amazon SQS service, which

is functionally representative of many cloud-based message queuing mechanisms.

Finally, the *use of students as participants* due to time, budget and partnership constrains—although similar to many software engineering experiments [215]—may make our results not generalisable to the target population for this experiment (i.e., software engineers).

Pickard, Kitchenham & Jones [190] emphasise that generalisation cannot be achieved by a single experiment, but by result replication and aggregation across multiple experiments. This is true for our results, which need to be replicated through experiments conducted with developers and software engineers in an industrial setting, and using additional real applications. Nevertheless, our study represents an important step towards understanding cloud portability and its underpinning factors.

3.4 Building Prediction Models

This section details the exploratory process of building regression models for predicting the effort of increasing cloud application portability by re-engineering legacy three-tier web applications to decouple application components by adopting a message queuing mechanism. By using concepts and techniques explained in Appendix C and results from the experiment reported in Section 3.3, we built a set of simple and multiple linear regression models using OLS and, alternatively, robust regression methods. Leave-one-out cross-validation (LOO-CV) was used to evaluate prediction models. The prediction model accuracy was assessed by using the MMRE summary measure. Additionally, we compared the accuracy of our models with maintainability prediction models reported in Section C.4. The rationale for techniques used in this section is detailed in Appendix C.

3.4.1 Simple OLS Linear Regression Model

The experiment reported in Section 3.3 suggests that OMMIC can be used as an indicator of cloud portability when replacing method calls with message queuing. Therefore, we built a simple linear regression model by using OMMIC as a single predictor. Figure 3.4 shows a jitter plot with all 63 observations and a model line representing the prediction model. As the plot shows, the data set consists of only a few observations for OMMIC = 5 and OMMIC = 7, which might compromise the model.

Table 3.13 summarises the accuracy results obtained with LOO-CV, and the goodness of fit for this simple model. According to the classification presented in Section C.4

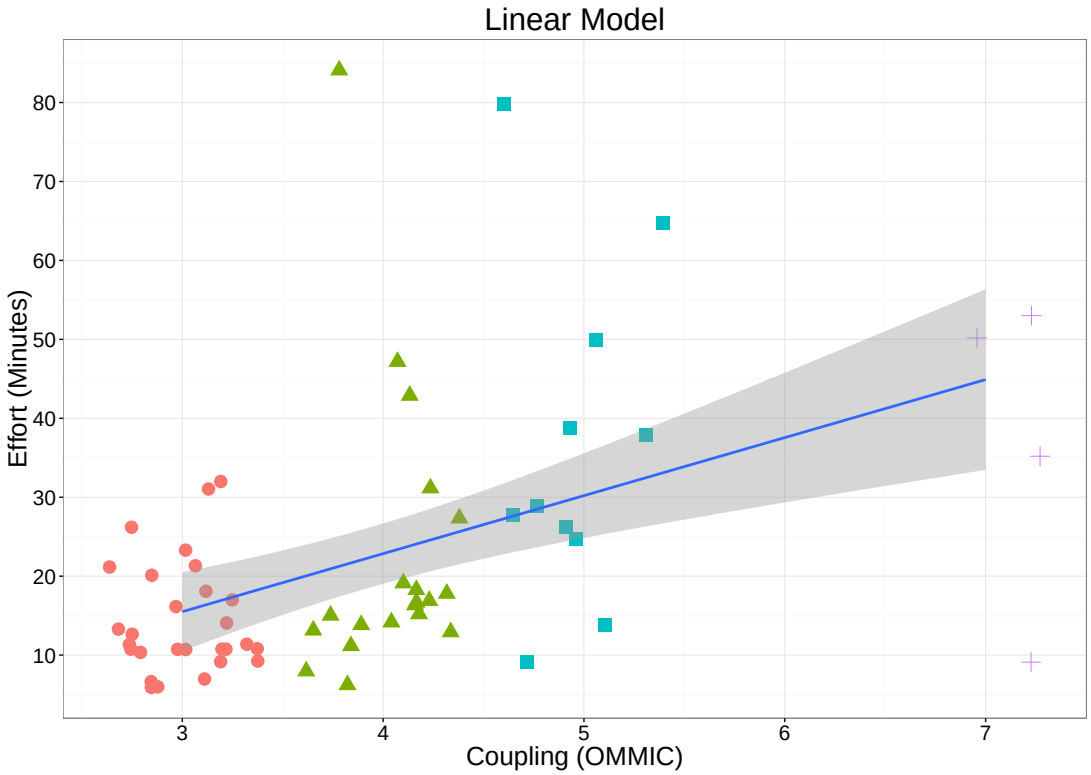


Figure 3.4: Simple effort prediction model line using OMMIC as a single predictor.

Table 3.12: Identification of possible outliers in the simple effort prediction model.

Variables	Observations			
	14	42	43	44
Participant	4	8	1	1
System	GsCloud	Siget	GsCloud	GsCloud
Class Size (LoC)	58	124	99	108
Class	CidadeControle	OsNovaControle	CompraControle	ContratoControle
OMMIC	4	7	5	5
Effort (Min)	84	9	80	65
Cook's Statistics	0.134	0.548	0.181	0.088
Residuals	4.075	-2.564	3.345	2.337

for maintainability prediction models, the simple model #1 achieved a *fair* accuracy ($0.49 < \text{MMRE} \leq 0.88$). Yet according to the accuracy classification, the Pred(0.37) shows that 41.3% of predictions using this model could be classified as *excellent*. The goodness of fit, as measured by the R^2, is relatively low ($R^2 = 0.224$), however.

3.4.1.1 Outlier analysis

As outliers influence the model, we tried to identify possible outliers and quantify their impact on the model. The plot in Figure 3.4 suggests three outlier candidates with Effort $\geq$ 65 min (Observations 14, 43 and 44). In addition, observation 42 presented an extremely low effort for OMMIC = 7. In addition to the jitter plot, we also used the Cook's distance to identify outlier candidates. As Figure 3.5 shows, only observation 42 was above the cutoff for Cook's distance (0.190) though observations 14, 43 and 44 are borderline.

We analysed these four observations to check whether they were influential cases for the model (Table 3.12). All four observations were heterogeneous when compared to similar observations in the data set (i.e., same participants, OMMIC values, and so on). Therefore, we concluded that they were real outliers, and we decided to remove these four observations. Figure 3.7 shows the resulting model line.

We can observe a lower intercept (dashed line) and a narrower confidence interval (green shade), suggesting some improvements in the model fit. Indeed, as Table 3.13 summarises, all accuracy and goodness of fit indicators improved considerably (model

Table 3.13: Accuracy and goodness of fit for effort prediction models built. Variables are identified according to Table 3.14

#	Prediction Model (Variables)	Accuracy Measures					Goodness of fit	
		MMRE	MAR	Pred(0.25)	Pred(0.30)	Pred(0.37)	R^2	RSE
1	Simple model (#1)	0.631	10.601	0.270	0.349	0.413	0.224	15.130
2	Simple model (#1) w/out outliers	0.477	7.470	0.424	0.492	0.593	0.405	9.396
3	Simple Robust model (#1)	0.412	7.227	0.390	0.458	0.576	0.162	6.388
4	Multiple model (#1+#2)	0.486	7.644	0.254	0.475	0.576	0.407	9.463
5	Multiple model (#1+#3)	0.477	7.611	0.322	0.356	0.576	0.418	9.377
6	Multiple model (#1+#4)	0.476	7.505	0.339	0.441	0.593	0.428	9.292
7	Multiple model (#1+#4+#2)	0.484	7.652	0.339	0.424	0.559	0.431	9.356
8	Multiple model (#1+#4+#3)	0.475	7.616	0.305	0.407	0.576	0.443	9.256
9	Multiple model (#1+#4+#3 + participant.experience.reeng)	0.469	7.611	0.339	0.441	0.610	0.447	9.304
10	Multiple model (#1+#4+#3 + system.id)	0.483	7.748	0.305	0.407	0.559	0.444	9.335
11	Multiple robust model (#1+#4+#3 + participant.experience.reeng)	0.451	7.841	0.390	0.475	0.542	0.824	8.388

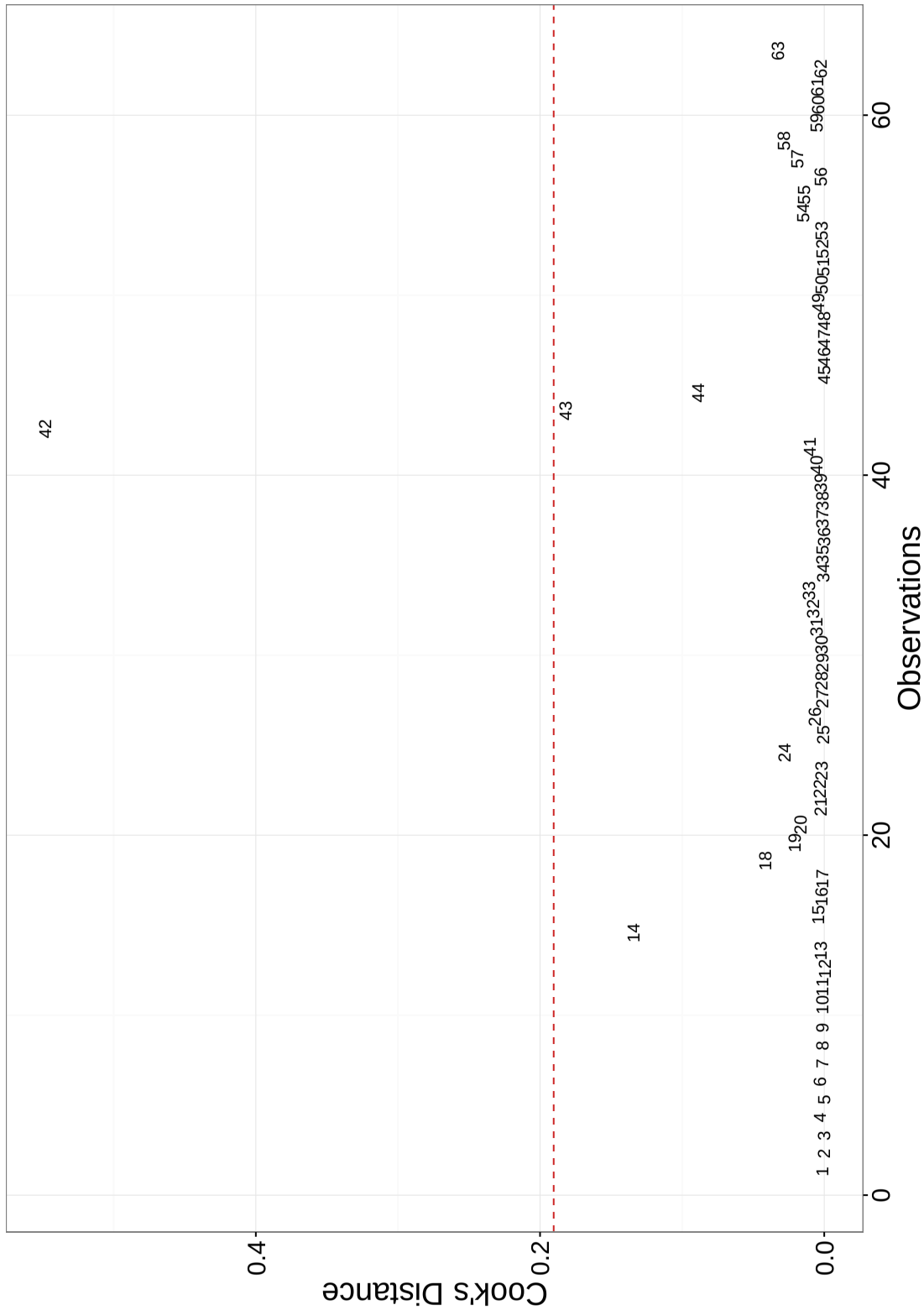


Figure 3.5: Cook's distance plot of data points. Only one observation is acknowledged as an outlier though other three are possible candidates.

#2). For instance, MMRE was reduced in 24.4% and RSE in 37.8% when compared to the previous model. This change makes model #2 be classified as *good*, according to the classification presented in Section C.4. Removing outliers also significantly increased the percentage of predictions classified as *excellent* (Pred(0.37) = 0.593).

3.4.1.2 Simple Model Diagnostics

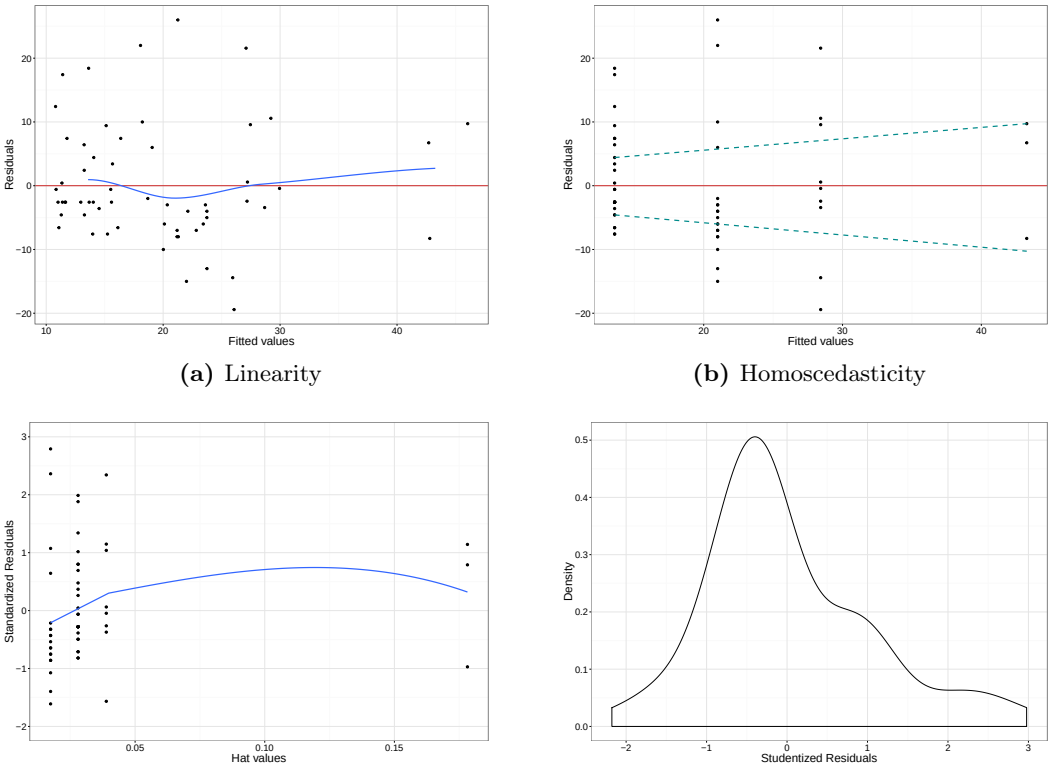
As explained in Section C.5, meeting some underlying assumptions is a pre-condition to accurately generalise a model beyond its data set. We tested the extent to which model #2 meets seven underlying assumptions for simple linear regression models based on the OLS method. Figure 3.6 shows four plots to visually support the diagnostics.

Sample size. In total, four observations were removed from the original sample, resulting in a reduced sample with 59 observations. As model #2 is based on a single predictor, the sample-predictor rate is within the acceptable range (i.e., ≥ 10).

Linearity. Figure 3.6 (a) shows the plot for residuals and fitted values of model #2. Data points are scattered along the red line (middle), suggesting a homogeneous distribution. The blue line, which supports the recognition of patterns, slightly deviates from the middle, but we cannot observe any significant pattern. The Rainbow test confirmed that the linearity assumption was met (Rain = 0.913, p = 0.596).

Independent residual terms (errors). We tested the lack of autocorrelation for residual terms by using Durbin-Watson test (DW = 1.821, p = 0.205). The DW statistics is close to 2, meaning that errors are independent. The p-value is not significant for $\alpha = 0.05$, confirming that model #2 meets this assumption.

Homoscedasticity. To meet this assumption, the variance of residual terms should be homogeneous. Figure 3.6 (b) helps to identify the homogeneity of the variance. The two dashed lines suggest a funnel pattern, which could mean a violation of homoscedasticity (i.e., the presence of heteroscedasticity). However, we should consider that the sample size is small for OMMIC = 5 and 7, which might contribute to this perspective. Furthermore, we can observe a homogeneous distribution of data points around the middle (red line). Finally, the Goldfeld-Quandt test confirmed that model #2 meets this assumption for $\alpha = 0.05$ (GQ = 2.494, p = 0.009).



(a) Linearity

(b) Homoscedasticity

(c) Low Leverage

(d) Normality

Figure 3.6: Diagnostics plots for effort prediction model #2

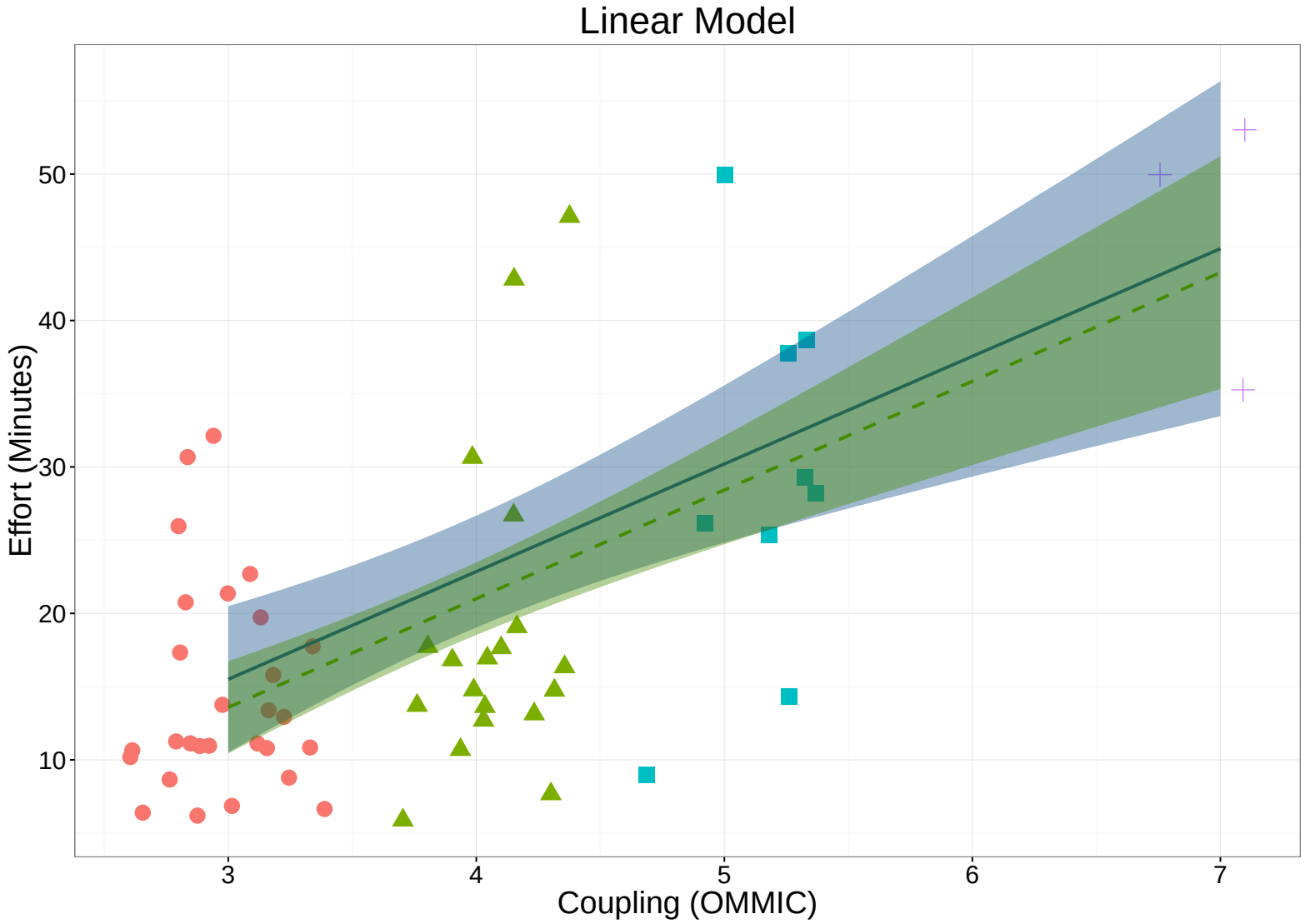


Figure 3.7: Simple model line for effort prediction model #1 (continuous line) and #2 (dashed line).

Low leverage. The plot in Figure 3.6 (c) suggests no major leverage issues as data points are centred around zero (y-axis). In addition, only three data points had Hat values greater than three times the leverage average (0.10). These three data points correspond to the OMMIC = 7.

No perfect multicollinearity. As Model #2 has only one predictor, there is no risk of multicollinearity.

Normality. The density plot in Figure 3.6 (d) shows a positive skew, which suggests deviation from normality. Indeed, as the Shapiro-Wilk test confirmed, model #2 violated the normality assumption (W = 0.936, p = 0.004).

3.4.1.3 Alternative Simple Robust Regression Model

Robust regression is a method less sensitive to outliers and make no assumption regarding the distribution type (Section C.5). This section uses this method to test whether it can improve the prediction accuracy of our regression model. Additionally, the robust regression is an alternative to overcome the lack of normality for the simple model #2.

Table 3.13 shows the accuracy and goodness of fit for the simple robust model #3. Compared to model #2, the robust regression improved the model in -13.6% (MMRE = 0.412) though all pred measures decreased. Regarding the goodness of fit, we can observe an improvement of -32% in the RSE measure (6.388), but a considerable decrease (60%) in the R^2 (0.162).

3.4.2 Multiple OLS Linear Regression Model

A multiple linear regression model is similar to a simple model, but it includes more than one predictor. We built a set of multiple linear models to investigate whether other variables in our data set could improve the prediction accuracy of our simple model. Firstly, we analysed possible predictor candidates. Secondly, we applied the forward stepwise method to identify the impact of predictor candidates in our model. Next, we analysed the existence of outliers. Then, we checked whether the best multiple linear model meets underlying assumptions for multiple linear regression models based on OLS. Finally, a multiple robust regression model was built to evaluate the impact of this method in the prediction accuracy.

Table 3.14: Ratio variables in our data set and their correlation with *effort*.

#	Ratio variables	rho	S	p
1	OMMIC	0.503	16979	<0.001
2	system.class.method.return	0.416	19969	0.001
3	system.class.loc	0.349	22244	0.006
4	participant.java.knowledge	-0.296	44356	0.022
5	system.class.method.void	0.117	30188	0.374

3.4.2.1 Analysis of Predictor Candidates

As the literature recommends [84, 206], we quantified the correlation between ratio variables in our data set and our dependent variable (effort) to identify predictor candidates. As Table 3.14 shows, *rho* values vary from 0.117 (lowest) to 0.503 (highest), disregarding the signal. To be considered as a predictor candidate, we set $rho = 0.193$ as the minimum cutoff. This value represents the minimum value for medium correlations in SE experiments [120]. Apart from variable #5 that did not meet our cutoff, all other variables were considered as possible predictors for the forward stepwise method.

3.4.2.2 Forward Stepwise Method

This method was used to define whether a predictor candidate should enter the multiple regression model. The entry condition was a MMRE <0.477, which was the best accuracy achieved by the simple OLS regression model. The stepwise method consists of several iterations. For each iteration, (i) one predictor candidate is added to the model, (ii) the model accuracy is re-evaluated by applying LOO-CV, and (iii) the entry criterion is applied to decide whether the predictor candidate should be kept in the model. As OMMIC is already in the model, we omitted this step and started with the second predictor candidate (variable #2, according to Table 3.14).

Iteration 1 In the first iteration, a second predictor was added to the simple model #2 as an attempt to improve its prediction accuracy. As Table 3.13 shows, model #4 (MMRE = 0.486) achieved a slightly worse (1.9%) prediction accuracy than model #2 (MMER = 0.477). Thus, variable #2 was removed from the model and a second iteration was performed.

Iteration 2 For the second iteration, the predictor candidate variable #3 was added to the simple model #2, resulting in the model #5 (Table 3.13). Although the prediction

accuracy did not change, we can observe some improvement in the goodness of fit.

Iteration 3 As variable #3 did not satisfy the criterion for inclusion in the model, it was removed and a new iteration was performed with the last predictor candidate in our list, variable #4. Model #6 (Table 3.13) resulted in a slightly better model (MMER = 0.476) when compared to the model #2 (MMER = 0.477). Although we can observe a significant improvement in the goodness of fit, some measures of accuracy decreased, such as Pred(0.25) and Pred(0.30). As the entry condition was met, this variable was kept in the model and a new iteration was performed.

Iteration 4 and 5 We tried to add a third variable to the model, resulting in models #7 and #8. As Table 3.13 shows, model #8 resulted in a better accuracy and goodness of fit. Therefore, variables #1, #4 and #3 were kept in the model.

Iteration 6 and 7 For these iterations, we forced the entry of two categorical variables. As Table 3.13 shows, the variable that measured whether participants had previous experience reengineering legacy applications (`participant.experience.reeng`), included in model #9, improved the model accuracy in -1.7% (MMRE = 0.469) when compared to model #2 (MMRE = 0.477). We can note that this variable also improved the Pred(0.37) measure. Therefore, this variable was kept in the model. The other categorical variable (`system.id`) that controlled the system used in the experiment did not result in a better model.

3.4.2.3 Outlier analysis

This section tries to identify possible outliers and quantify their impact on the multiple model #9. As there are multiple predictors in the model, it is no longer possible to infer outliers just by inspecting the jitter plot. Therefore, we used the Cook's distance to identify possible outliers.

Using default values for the cutoff of Cook's distance, we could not identify any outlier candidate (Figure 3.8). We investigated observations 18, 23 and 54 because they significantly differ from others, but we could not find any evidence that make them outlier candidates. For instance, observation 23 is one out of the three observations for the class `br.com.siget.controle.CompraControle`. The effort for this observation was 53 min, whereas the other two observations took 35 and 50 min. Thus, we concluded that model #9 was the best model we could find for this data set using OLS multiple linear regression.

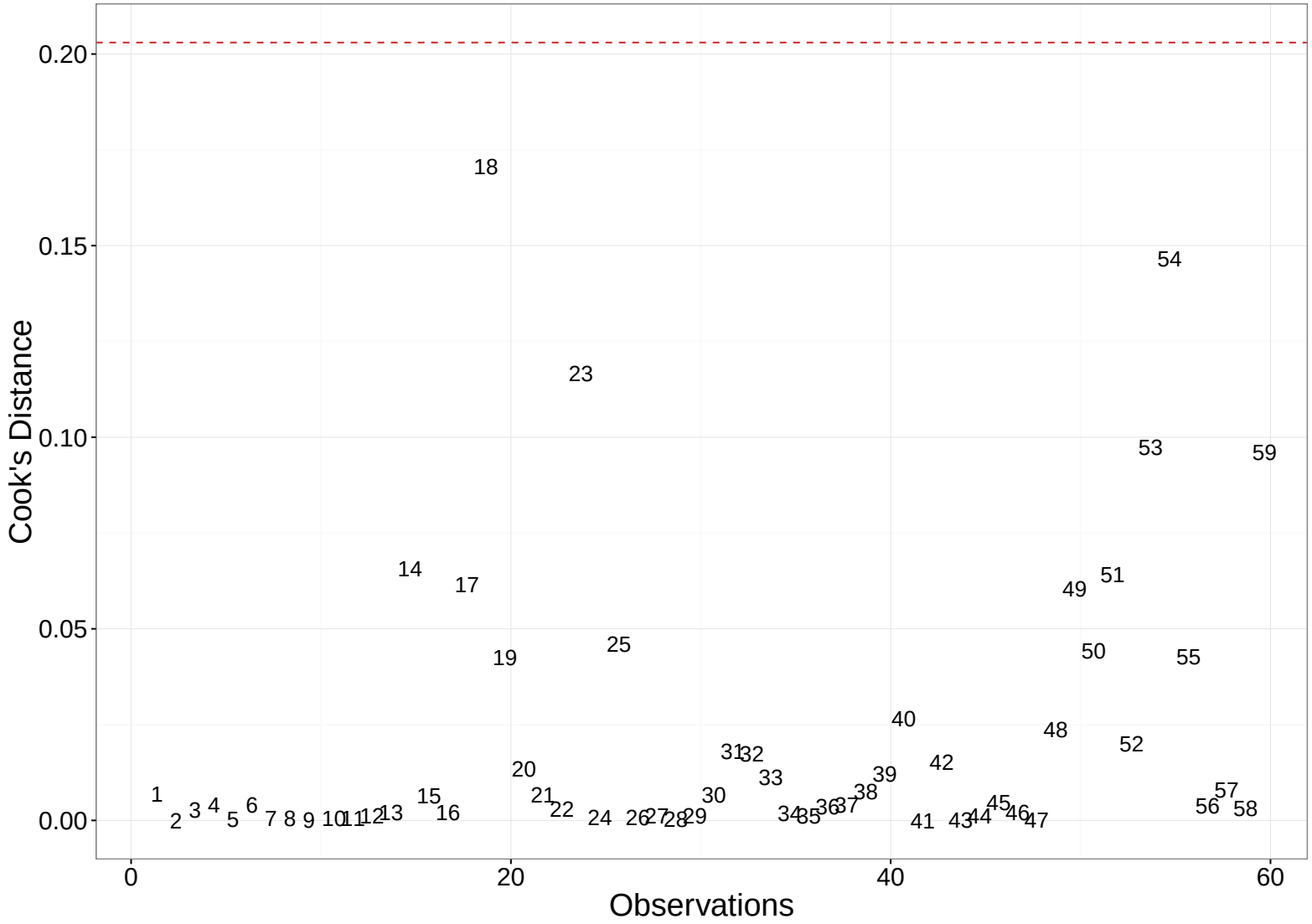


Figure 3.8: Cook's distance plot for the multiple effort prediction model #9.

3.4.2.4 Multiple Model Diagnostics

As for the simple model, we carried out a set of tests to identify the extent to which the best multiple model meets seven underlying assumptions for multiple linear regression models based on the OLS method. As a reference, we used model #9, as it obtained the best accuracy amongst all multiple models built (Table 3.13). Figure 3.9 shows four plots to visually support the diagnostics.

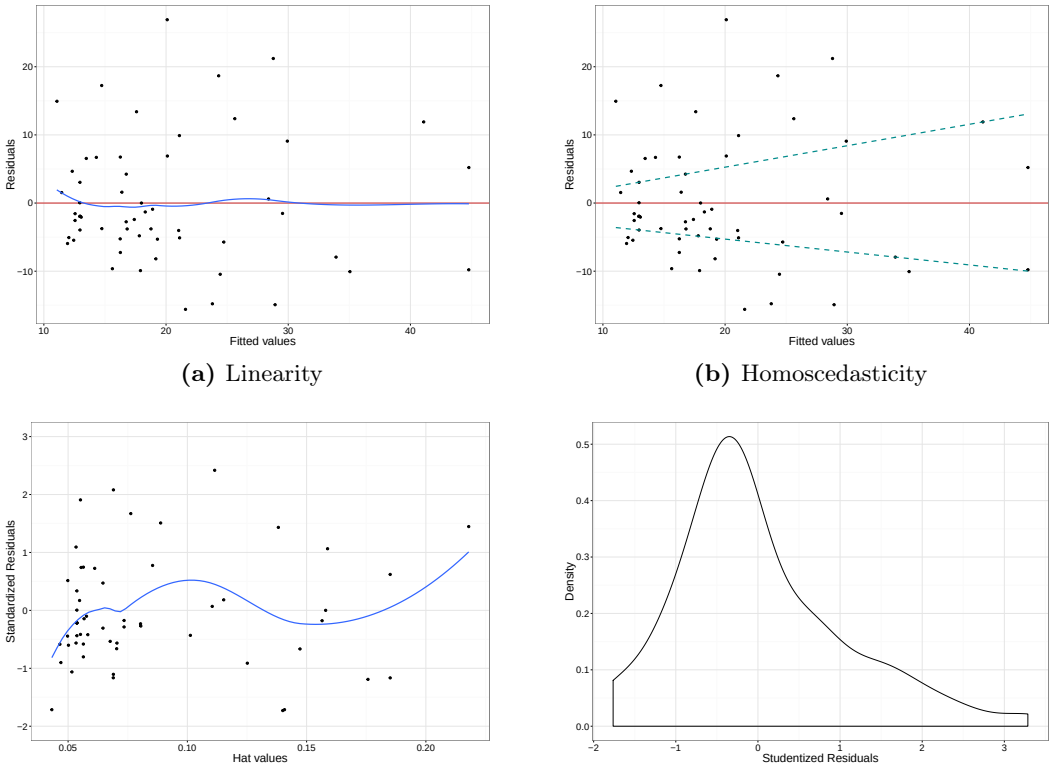
Sample size. Unlike the simple model, the multiple model #9 consists of four predictors. For our reduced sample with 59 observations the sample-predictor rate is 14.75, which is still greater than the minimum cutoff of 10. Therefore, model #9 meets this assumption.

Linearity. Figure 3.9 (a) shows the jitter plot for residuals and fitted values of multiple model #9. Data points are scattered across the plot around the red line (middle), and the blue line suggests no pattern at all. To test the assumption that the model has no linearity issues, we applied the Rainbow test that confirmed that the model met the linearity assumption (Rain = 1.503, p = 0.154).

Independent residual terms (errors). We tested the lack of autocorrelation for residual terms by using Durbin-Watson test (DW = 1.742, p = 0.097). Like in the simple model, the DW statistics is close to 2, meaning that errors are independent. The p-value is not significant for $\alpha = 0.05$, confirming that model #9 meets this assumption.

Homoscedasticity. To meet this assumption, the variance of residual terms should be homogeneous. Figure 3.9 (b) helps to identify the homogeneity of the variance. Like for the simple model (Figure 3.6 (b)) the two dashed lines suggest a funnel pattern, which could indicate the presence of heteroscedasticity. Although we can observe a homogeneous distribution of data points along the middle (red line), we applied the Goldfeld-Quandt to test whether the homoscedasticity assumption was met. The test result (GQ = 3.095, p = 0.003) is significant for $\alpha = 0.05$, which confirms that the assumption is met.

Low leverage. The plot in Figure 3.9 (c) suggests no major leverage issues as data points are centred around zero (y-axis). The average leverage for this model is 0.084. Therefore, the cutoff for analysis is three times this average (0.254). Examining the



(a) Linearity **(b)** Homoscedasticity

(c) Low Leverage **(d)** Normality

Figure 3.9: Diagnostics plots for effort prediction model #9

plot, we cannot observe any data point above the cutoff. Therefore, we considered that model #9 meets the low leverage assumption.

No perfect multicollinearity. Unlike simple model #2, multiple model #9 might violate the multicollinearity assumption as it has multiple predictors. To test this assumption, we checked the variance inflation factor (VIF) that indicates whether a predictor has strong relationship with other predictors. Table 3.15 shows that the VIF calculated for all four predictors does not represent major concerns as they are lower than 10, as recommended in the literature [84]. Furthermore, we also calculated the tolerance statistics. As Table 3.15 shows, the tolerance is greater than 0.2, which does not represent an issue. Thus, we considered that the model #9 meets this assumption.

Normality. As for the simple model #2, the density plot in Figure 3.9 (d) suggests a positive skew. As confirmed by the Shapiro-Wilk test, model #9 also deviates from normality ($W = 0.942$, $p = 0.007$).

Table 3.15: VIF and tolerance analysis.

#	Predictor	VIF	Tolerance
1	ommic	4.654	0.214
2	participant.java.knowledge	1.876	0.533
3	system.class.loc	4.610	0.216
4	participant.experience.reeng	1.877	0.532

3.4.2.5 Alternative Multiple Robust Regression Model

As Table 3.13 shows, the goodness of fit improved significantly for the multiple robust regression model #11 ($R^2 = 0.824$, RSE = 8.388) when compared to the best multiple model (#9). In addition, the accuracy also improved -3.8% (MMRE = 0.451). However, it is important to highlight that model #11 does not outperform the accuracy of simple robust regression model #3 (MMRE = 0.412).

3.4.3 Discussion

From 11 prediction models built in Sections 3.4.1 and 3.4.2, the simple robust model #3 achieved the best accuracy as measured by the MMRE summary measure (MMRE = 0.412). This model consists of one single predictor, the OMMIC variable (representing coupling). Although multiple models based on the OLS method presented an improved accuracy than the simple OLS model, this result did not hold for the robust method.

According to the accuracy classification presented in Section C.4, model #3 achieved a *good* accuracy. In addition, nearly 60% of predictions made by this model achieved an *excellent* accuracy (Pred(0.37) = 0.576). Finally, this model is robust to the lack of normality, presented by the simple OLS model, and meets all other assumptions for the generalisation of linear regression models. Therefore, we conclude that this model can be used to predict the effort of increasing cloud application portability within the context defined in Section 3.3

3.5 Summary

This chapter represents the first step towards addressing the objectives defined in Section 1.2. We empirically investigated a software design property, coupling, to identify whether it impacts cloud application portability. The result of the experiment in Section 3.3 confirmed our hypothesis - coupling does impact cloud application portability

within the context defined for the experiment. Moreover, the experiment achieved at least a medium effect size and a considerably high statistical power for all comparisons made, when taking into consideration SE experiments. Additionally, we used the data set produced in the experiment to build 11 prediction models by using two different prediction methods for linear regression. Our best model achieved a *good* accuracy when compared to other prediction models reported in the related literature (Section C.4). Moreover, nearly 60% of model predictions could be classified as *excellent*. Our best model meets seven common assumptions for linear regression models, which enables its generalisation beyond this data set. Together, these results are aligned with the outcomes envisaged for this research.

Chapter 4

Investigating the Impact of Security Systems on Cloud Application Portability

Unlike the previous chapter that investigated a design property, this chapter investigates the hypothesis that technologies adopted by cloud applications impact cloud application portability. This hypothesis was inspired by the literature on software migration, grid computing and migration to the cloud, which suggests that technologies adopted by software applications can also impact on portability (Section 2.2.3).

To select a particular technology to investigate, we took into account two aspects. Firstly, security has been as a top concern for cloud adoption according to both surveys with practitioners [47, 77, 169] and the cloud literature [31]. Thus, it makes sense to investigate technologies that implement security principles. Secondly, the migration scenario addressed in this research (Figure 2.5) requires re-engineering legacy three-tier web applications to enable them as a Distributed System (DS) (Sections 2.1.3 and 2.4.5). Ideally in a DS, security systems that implement Identity & Access (I&A) management (i.e., authentication and authorisation, respectively) should enable application users to authenticate only once and gain access to multiple application components without the need for re-typing their password (single sign-on) for each component accessed during a session [50]. Thus, we investigate whether the effort to enable single sign-on varies with the security system. Security system is a generic term used in this chapter to refer to a framework (e.g., Spring Security) or a software module (e.g., the container

security in Glassfish application server[1]) that implements security mechanisms, such as authentication and authorisation.

In particular, this chapter empirically investigates the impact of the choice of security systems on cloud application portability in the context of modifying the security system of legacy three-tier web applications to enable the use of single sign-on by adopting implementations of the OpenID protocol (Section 4.2). We explore the outcomes of this investigation by building prediction models to support decision makers when analysing the required effort to increase the cloud portability of their applications (Section 4.3). Before presenting these results, we start the chapter by introducing key concepts of authentication for DS in Section 4.1.

4.1 Review of Authentication in Distributed Systems

Identity & Access are principles of DSs that are mainly enforced by two mechanisms [228]:

- *Authentication* is the process of confirming that users are who they claim to be. This is the first step to grant a user access to protected resources; and
- *Authorisation* is the process of granting an authenticated user access only to resources over which they have access right.

Both authentication and authorisation are mechanisms implemented by security systems [228], such as security services provided by JEE containers[2] (Container-based security) or the Spring Security Framework[3]. Due to time and scope limitations, this research focus only on authentication.

One way to authenticate users in DSs is by adopting Federated Identity (FI) management systems [50, 139, 228], such as Kerberos[4] and Shibboleth[5]. As Chadwick [50] explains, a FI system enables users to use their *"credentials (authentication and authorisation) from one or more identity providers to gain access to other sites (service providers) within the federation."* A benefit of FI is the single sign-on (SSO) capability [50], which enables users to *"log-in once and gain access to multiple websites without the hassle of repeatedly typing their passwords"* [248].

[1] `http://glassfish.java.net`
[2] `http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm`
[3] `http://projects.spring.io/spring-security/`
[4] `http://web.mit.edu/kerberos/`
[5] `http://shibboleth.net/`

Although FI systems have been there for a long time, SSO has recently gained popularity thanks to cloud applications [172, 248]. With the proliferation of services provided through the internet, the website-specific password-based authentication used by traditional web applications became unfeasible [81, 139, 150]. SSO has not only become a requirement of service users [96], but it is also key to a seamless user experience in cloud [150, 172]. Therefore, adopting SSO legacy web applications are not only becoming more cloud portable, but they are also becoming more cloud-like [172].

OpenID Connect[1] is a SSO protocol that has been supported by large organisations, such as Google and Yahoo [30, 150]. The OpenID protocol consists of the following entities [30]:

- *End user* represents the entity (person or machine) that wants to access a cloud application or service;
- *Identity provider* is a third-party entity responsible for authenticating the *end user*;
- *Relying party* is the security system implemented by the cloud application or service that receives the authentication response from the *identity provider*;

The OpenID foundation maintains a list of certified identity providers[2], which includes Google and PayPal, for instance. These identity providers implement the OpenID Connect specification, and therefore can be seamlessly used to authenticate end users [150]. The OpenID Connect specification must be also implemented by the cloud application or service (relying party) to enable that end users are acknowledged as authenticated users [30]. To support relying parties to implement the OpenID Connect protocol, several libraries, services and tools are currently available[3]. One interested in using SSO can adopt one of these libraries, services and tools to enable OpenID Connect authentication.

4.2 Empirical Investigation

The goal of this experiment is to analyse two security systems commonly adopted by legacy web applications. We aim to evaluate these security systems with respect to their impact on the effort to increase cloud application portability from the point of

[1] `http://openid.net/connect/`
[2] `http://openid.net/certification/`
[3] `http://openid.net/developers/libraries/`

view of the software developers. This evaluation is performed in the context of software developers modifying configurations and code related to the implementation of security systems in legacy web applications. The modification aims to enable the use of SSO by adopting implementations of the OpenID Connect protocol.

We start this section by detailing the experiment protocol and its execution (Section 4.2.1). Secondly, we present characteristics of participants of this experiment (Section 4.2.2). Next, we show the main results of our experiment (Section 4.2.3). Then, we analyse the impact of these results in our proposed hypothesis, and discuss the implications of our findings (Section 4.2.4). Finally, we examine the main threats to the validity of our findings (Section 4.2.5).

4.2.1 Experiment Plan and Execution

As explained in Section 1.2, we used guidelines provided in [256] to prepare and conduct this experiment. In addition, this experiment followed recommendations put forward by Sjøberg et al. [214] to increase the realism of experiments in terms of participants, tasks and environment. Next subsections correspond to elements of the experimentation framework, detailing the protocol for the experiment, and its execution. Terminology associated with the experimentation framework is highlighted in the main text; this terminology is explained in Appendix B.

4.2.1.1 Participant Selection and Study Design

To increase both realism and external validity of this experiment [214], we recruited professional software developers to take part in this experiment. The recruitment and selection followed guidelines proposed in [131], and basically consist of:

1. *Setting up the population of interest.* Our focus is software developers that are responsible for maintaining the application code. In addition, we focused on small and medium Brazilian software development companies;

2. *Defining inclusion criteria.* The four inclusion criteria are: (i) more than one-year of professional experience, (ii) previous experience changing third-party code, (iii) Java programming skills equals to or more than level 2 in the self-evaluation test, and (iv) some knowledge on security mechanisms addressed in the experiment;

3. *Identifying sources for participant selection.* All participants work for a Brazilian company which we established partnership with for this experiment. Possible threats to the validity raised by this sample is discussed in Section 4.2.5;

4. *Defining compensation for participants.* We surveyed local companies to identify the current hourly rate paid for software developers. The hourly rate varies from R$ 12.00 to R$ 25.00[1], varying according to the developer experience. Therefore, we decided to pay the intermediary amount of R$ 18.00, which is equivalent to US$ 4.50. In addition to the payment, all participants received certificates for the concepts addressed in this experiment;

5. *Performing the recruitment and selection.* Once the partnership with the Brazilian company was established, we contacted the project manager, who took care of advertising the experiment to their developers. With the inclusion criteria in mind, the project manager pre-selected those interested in taking part in this experiment, which resulted in six participants.

Due to the limited number of participants, we decided for a *within-subject* design for this experiment. In practice, it means that all six participants worked on both *control* (CG) and *treatment* (TG) groups. Although we hired these six developers, we gave them the chance to decide whether or not to participate. Hence, they could resign from the experiment if they wanted to. No participant gave up (*drop-out = 0*).

4.2.1.2 Instrumentation

Software systems, APIs, identity provider, and experiment materials are instruments used in this experiment, as follows.

Software systems Three Java-based web applications are used in this experiment. All three applications implement Spring Security Framework. The web applications consist of one prototype and two commercial information systems. The prototype was developed for the pretest (Section 4.2.1.6) and consists of a web application that authenticates the user and grants access to a welcome page. The two commercial information systems (Siget and NisseiNet) were originally developed by students as their final project to graduate as Bachelors in Information Systems. As these are commercial systems, their code and documentation are not available to the general public. These

[1]Real (R$) is the Brazilian currency. For the time this document was written, one Real was equivalent to US$ 0.25.

Table 4.1: Measures of size for software applications used in the experiment.

Size Metric	Siget	NisseiNet
Classes	84	69
Methods	623	530
Configuration Files	12	12
Web pages	60	57
Lines of Code	4,793	3,237

systems are currently deployed in small Brazilian companies. The main reason to select these systems is that they are small and self-contained, as opposed to large legacy systems that would increase the complexity for our participants. Table 4.1 presents size measures for the two information systems. Possible threats to validity raised by this selection are discussed in Section 4.2.5.

To enable the effort comparison of Container-based security (Container) and Spring Security Framework (Spring) - the two security systems investigated in this experiment - the author of this thesis re-engineered the two applications to create versions of these applications that implement Container-based security. This re-engineering was necessary to ensure that *objects* manipulated in this experiment (i.e., applications) are alike, and therefore, the only difference is the security system implemented. Possible threats to validity raised by this approach are discussed in Section 4.2.5.

As a *within-subject* experiment, participants are randomly assigned to all *objects*. The exception is the prototype application. As the purpose of prototype is to be used as pretest, all participants received the prototype as their first application - though we randomised the version. Table 4.2 shows the order in which participants received their *objects*. This experiment had 24 observations[1] (*sample size*) since six participants modified two versions of the two commercial information systems (recall that the prototype was a pretest and therefore, it is not part of our hypothesis testing).

APIs The OpenID Connect protocol is implemented by several APIs[2]. We investigated some of these APIs to find one that could be applied for security systems adopted in this experiment, and that offered minimal complexity for learning and using. Two APIs were used: Scribe[3] for Container, and MITREid Connect[4] for Spring. Using

[1]This number disregards observations produced for the Prototype application.
[2]`http://openid.net/developers/libraries/`
[3]`http://github.com/scribejava/scribejava`
[4]`http://kit.mit.edu/projects/mitreid-connect`

Table 4.2: *Participants* were randomly assigned to *objects*, apart from the Prototype application (pretest).

Participant	Prototype		Siget		NisseiNet	
	Container	Spring	Container	Spring	Container	Spring
1	2	1	6	5	3	4
2	2	1	5	6	3	4
3	1	2	6	5	3	4
4	1	2	3	4	5	6
5	1	2	5	6	3	4
6	1	2	4	3	5	6

two different APIs might raise some concerns regarding the validity of this experiment. These concerns are discussed in Section 4.2.5.

Identity provider As explained in Section 4.1, the OpenID Connect protocol relies on an identity provider. The OpenID Connect website maintains a list of certified providers. For this experiment, Google was used as identity provider because it provides detailed documentation and examples on how to implement its authentication mechanism (Google Sign-in).

Experiment materials Training material, collection forms and guidelines for performing this experiment were carefully prepared by the experimenter and validated by two experienced researchers.

4.2.1.3 Task

The *task* performed in this experiment consists of modifying application code and configuration related to the security system to implement SSO by using OpenID Connect protocol implementations and Google as an OpenID identity provider. The procedure for modifying the application slightly differs between security systems, and can be performed in different ways even for the same system. Hence, this task can be considered as *non-deterministic*, as the randomness involved in how the task is performed might affect the result of effort, e.g. by using additional libraries.

In summary, the task consists of performing the following steps:

1. Importing the application into the IDE;
2. Adding/updating libraries;

3. Analysing application architecture;
4. Modifying application/security system to deal with the OpenID Connect authentication flow;
5. Modifying the login page to support Google Sign-in;
6. Testing the application and correcting possible problems.

Step 4 represents a major difference between Container and Spring because Container requires mainly code modification whereas Spring requires only configuration. This difference is due to the Spring implementation chosen for this experiment.

Spring provides two implementations[1]: XML-based (Security Namespace Configuration) and code-based (Java Configuration). We opted for using the XML-based implementation because the two commercial systems used in this experiment use the XML-based implementation of Spring. Possible threats for the validity of this experiment raised by this choice are discussed in Section 4.2.5.

We did not consider authentication configuration on the Google service as part of the task investigated in this experiment. The experimenter configured the Google service for each of the six applications used as objects in this experiment. This Google service configuration is necessary to make the Google service recognise the application as a relying party. This configuration was not included as part of the task to reduce the complexity and save time in the experiment execution. Moreover, this configuration is the same regardless of the security system adopted.

To control the time spent (*effort*) for changing each application, participants registered the start and end time on *data collection form*, along with the step they had to perform. We classified participants' steps into four categories (*subtasks*):

- *Environment preparation*, which includes importing the project into the IDE and adding libraries into the project;
- *Code analysis*, which consists of getting familiar with the application;
- *Change*, which represents the necessary changes to fulfil the required task; and
- *Test & corrections*, which consists of testing the application to check the effectiveness of changes and making any necessary correction.

This classification enabled us to investigate whether there is any difference across different *subtasks* (Section 4.2.3.3).

[1] `http://docs.spring.io/spring-security/site/docs/4.0.3.RELEASE/reference/htmlsingle/`

Table 4.3: Formal definition of hypotheses.

Null hypothesis(H_0)	$\tilde{x}\,T_{\text{Spring}} = \tilde{x}\,T_{\text{Container}}$
Alternative hypothesis(H_1)	$\tilde{x}\,T_{\text{Spring}} > \tilde{x}\,T_{\text{Container}}$

4.2.1.4 Selection of Variables and Definition of Hypotheses

Security systems represent the *independent* variable, or *factor*, evaluated in this experiment. Container and Spring are the two possible values (*treatments*), measured on a *categorical scale*. These two security systems have several versions. For the Container, we used the Servlet API version 3.0.1 whereas for the Spring we used the Spring Security Framework version 4.0.1.

The effort is the *dependent* variable, measured in the number of minutes[1] (*ratio scale*) to perform the task presented in Section 4.2.1.3.

Two hypotheses were defined for this experiment, as formalised in Table 4.3. T is the number of minutes spent to modify an application. Thus, the *null hypothesis* states that the median time is the same regardless of the security system whereas the *alternative hypothesis* states that Spring takes **more time than** Container. We chose this *alternative hypothesis* rather than **different time** because our preliminary analysis indicated that Spring takes more time than Container.

4.2.1.5 Data Analysis

As explained in Section 1.2, we adopted only *non-parametric* statistics in this experiment, as summarised in Table 4.4 whereas cutoffs we used for inferential tests are summarised in Table 4.5.

4.2.1.6 Experiment Execution

A pilot experiment was performed by the author of this document to check the task complexity, materials for training and experiment, and time required for performing the task. The result of this pilot prompted some changes in the experiment protocol. In addition, it provided preliminary guesses on the results. The experiment was undertaken in five steps: (i) participant selection; (ii) training; (iii) pretest; (iv) experimentation; and (v) feedback.

1. *Participant selection.* The process for participant selection was previously described in Section 4.2.1.1. After selection, participants received an *introduction*

[1] We use the terms *effort* and *time* interchangeably throughout this document.

Table 4.4: Summary of statistics and statistical tests in the experiment.

Statistic Type	Statistic/Test	Definition/Config.	Purpose
Descriptive	Median and quartiles	25%, 50%, 75%, 100%	To identify the middle value and categorise occurrences into quartiles according to their values.
	Range	Maximum - Minimum	To quantify the data dispersion.
Inferential	Shapiro-Wilk	Two-tailed, $\alpha = 0.1$	To test the assumption that the data is normally distributed.
	Wilcoxon rank-sum	One-tailed	To test the statistical significance of differences between two medians.
	Spearman's correlation	Two-tailed	To test the strength of a relationship between two numerical variables.
	Kruskal-Wallis	Two-tailed	To test the statistical significance of differences between several medians.

Table 4.5: Cutoff values for evaluating the hypotheses. Table adapted from [70].

Unknown true state of nature			
		H_0: $\tilde{x}T_{\text{Spring}} = \tilde{x}T_{\text{Container}}$	H_1: $\tilde{x}T_{\text{Spring}} > \tilde{x}T_{\text{Container}}$
Statistical Conclusion	Accept H_0	0.90 1 - α: Correct *(Confidence level)*	0.24 β: Type II error
	Reject H_0	0.10 α: Type I error *(Significance level)*	0.76 1 - β: Correct *(Statistical power)*

script. This script provided an overview for the experiment, highlighting its purpose, the task that participants should perform, and benefits of taking part in this experiment. Those who accepted taking part in the training session were asked to fill out the *participant characterisation form*. This form collected information regarding the participants' background, such as professional experience. Questions on this form were based on guidelines proposed in [82];

2. *Training*. Although selected participants fulfilled the basic selection criteria, they had limited knowledge on some topics addressed in this experiment (Section 4.2.2). The training session covered concepts related to cloud, security systems, and the OpenID Connect protocol. Moreover, it prepared participants to perform the task evaluated in this experiment. The total training time was two hours;

3. *Pretest*. The first application assigned to participants was a pretest (the prototype described in Section 4.2.1.2). This strategy aimed to reduce the *learning effect*. It is common in experiments that participants take more time to perform the first task as they are getting familiar with the experiment [131]. However, participants were not informed about the pretest to prevent that they behave differently as that application would not be analysed;

4. *Experimentation*. The participants were allowed to take short breaks, which were excluded from the time spent on the tasks. The experimenter observed the experiment execution without interfering in tasks performed by participants. Participants were strongly advised to not copy and paste any code as it could bias the result. The experiment was performed in two four-hours sessions; and

5. *Feedback*. When the experiment had finished, participants were asked to fill a *feedback form*, assessing their experience in this experiment. Participants received their participation certificates a few weeks after submitting the *feedback form*.

Both training and experiment sessions were performed in the company where the participants work, using their usual computers and tools. This is an important aspect to increase the realism of this experiment [214]. For information on ethics in this experiment, refer to Section 1.3.

4.2.2 Characterisation and Feedback of Participants

This section examines responses from the *participation characterisation form* (Section 4.2.1.6). We have classified responses into four groups related to participants' back-

Table 4.6: Summary of software development background of participants.

#	Question	Answers Years/# Languages/Grade	N	(%)
1	How long have you been programming?	1 - 2	1	17
		6 - 10	4	67
		>10	1	17
2	How many programming languages do you have experience with?	2	2	33
		3	1	17
		>3	3	50
3	How would you grade your current development skills in Java?	2	1	17
		3	3	50
		4	2	33
4	How would you grade your experience on Github or any other repository based on Git?	2	1	17
		3	2	33
		4	3	50
5	How would you grade your skills with Maven?	2	1	17
		3	3	50
		4	2	33

ground regarding: personal, software development (Table 4.6), professional (Table 4.7) and experiment-related skills (Table 4.8). Responses for self-evaluation questions vary from zero (lowest) to five (highest).

Regarding the group of personal questions, we identified that participants were 21 (n = 1), 25 (n = 1), 27 (n = 2), 33 (n = 1), 40 (n = 1) years old. All participants hold a graduate degree, and two of them are postgraduate. All participants are male.

Table 4.6 summarises responses for the group of software development questions, which aim to identify participants' development skills. In this group, participants were asked about their programming background and to evaluate their development skills and knowledge on common software development tools. Most participants program computers for more than 6 years (question 1), are skilled in more than one programming language (question 2), has medium to high Java skills (question 3), and has some experience in common development tools (questions 4 and 5). It is worth to note that questions 2, 3 and 5 are used to infer the overall professional level of our participants. For instance, Maven and Git are tools often used by software professionals [153]. Similarly, experienced professionals tend to know more programming languages due to the variety of projects they worked on [82].

Table 4.7: Summary of professional background of participants.

#	Question	Answers Size/ Years/ Grade	N	(%)
6	Usually, how large are the projects that you take part in?	Medium (900-40,000 LoC)	5	83
		Large (>40,000 LoC)	1	17
7	How long have you been developing software professionally?	1 - 2	1	17
		3 - 5	3	50
		6 - 10	1	17
		>10	1	17
8	How would you grade your current development skills when compared to the most experienced developer you have worked with?	1	1	17
		2	1	17
		3	1	17
		4	3	50
9	How would you grade your current development skills when compared to your colleagues?	2	1	17
		3	1	17
		4	3	50
		5	1	17

Table 4.7 summarises responses for the group of professional questions, which aim to identify important aspects of the professional background of participants. Regarding the size of projects they are involved in, most participants take part in medium projects (n = 5). When asked to compare their development skills with experienced developers (question 8) and their colleagues (question 9), we can observe a slight variation in their evaluation compared with their skills in Java (question 3).

For question 8, participants 1, 4 and 5 kept the same grade as in the question 3 whereas participants 3 and 6 decreased their grades in one point. Only participant 2 increased his grade from 3 to 4. For question 9, half group kept the same grade as question 3 (Participants 1, 3 and 6) whereas the other half increased one point. When compared with question 8, only participants 1 and 2 kept their grades for question 9, all others increased their grades in one point. This self-evaluation suggests that most of them consider themselves in a stable position when compared with their workmates.

Table 4.8 summarises responses for the group of experiment-related skills questions, which aim to identify participants' level on skills for this experiment. All participants had experience changing third-party applications (question 10). Although one participant did not have any experience implementing security systems in an application (question 11, Participant 6), all participants had some skills on both security systems

Table 4.8: Summary of experiment-related skills of participants.

#	Question	Answers		
		Grade/ Answer	N	(%)
10	Have you ever changed an application developed by a third party?	Yes	6	100
		No	0	0
11	Have you ever partially implemented any security system in an application?	Yes	5	83
		No	1	17
12	How would you grade your current skills on Spring security framework?	1	2	33
		2	3	50
		4	1	17
13	How would you grade your current skills on container-based security?	1	4	67
		4	2	33
14	How would you grade your current knowledge on OpenID?	0	3	50
		1	1	17
		3	2	33

investigated in this experiment (questions 12 and 13). Half of participants did not have any knowledge on OpenID (question 14, Participants 2, 3 and 6).

Responses for questions 11 to 14 show some gaps that could negatively impact this experiment. Therefore, a training session was performed (Section 4.2.1.6). We evaluated the effectiveness of training session by asking some questions with the *feedback form* (Section 4.2.1.6). Table 4.9 summarises how participants evaluated their skills/knowledge before and after the training session regarding the three main concepts addressed in this experiment. We can observe that the training was effective to provide participants the knowledge/skill necessary for the experiment. Moreover, another question asked participants to evaluate to what extent the training session provided what they needed for the experiment. Five out of six participants evaluated this question with 5, and one participant with 4.

Table 4.10 summarises further responses for questions asked in the *feedback form*. We did our best to increase the realism of this experiment by hiring real developers, performing the experiment in their usual environment, and giving participants a real task (Section 4.2.1). As question 15 shows, all participants evaluated the task reality from medium to high. However, we can note that the task easiness was perceived differently across participants (question 16). Participants 2 and 3 evaluated this question with 5 (very easy) whereas participants 1 and 4 evaluated with 2 (not too easy).

Finally, we asked participants to select which security system was easier to work

Table 4.9: Comparison of participant's skills before an after training session.

Participant	How would you grade your current (..)					
	(..) skills on Spring security framework?		(..) skills on container-based security?		(..) knowledge on OpenID?	
	Before	After	Before	After	Before	After
1	2	3	4	5	1	3
2	2	3	1	3	0	4
3	1	3	1	4	0	4
4	2	3	1	3	3	3
5	4	4	4	4	3	4
6	1	2	1	3	0	2

Table 4.10: Participant's responses for the feedback form.

#	Question	Answers		
		Grade/ Mechanism	N	(%)
15	How much do you agree that the activity performed in this experiment is realistic?	3	1	17
		4	4	66
		5	1	17
16	In your perspective, how easy was performing activities required by this experiment?	2	2	33
		3	1	17
		4	1	17
		5	2	33
17	In your perspective, which security system was easier to work with?	Container	4	67
		Spring	2	33
18	Which security system would you prefer adopting in your applications?	Container	1	17
		Spring	5	83

Table 4.11: Descriptive statistics for treatments. Spring-based applications require more effort for all statistics analysed.

Statistics	Container	Spring	Difference (%)
Min	10.00	17.00	70.0
1st Quartile	14.75	19.75	33.9
Median	23.00	32.50	41.3
Mean	26.33	46.08	75.0
3rd Quartile	35.00	55.75	59.3
Max	50.00	129.00	158.0
Range	40.00	112.00	180.0

with (question 17). Four out of six participants agreed that the Container system was easier than Spring. This result is in line with results presented in Section 4.2.3. The two disagreements came from participants 1 and 5 - the two experts. On the other hand, most participants would prefer Spring than Container in their applications (question 18). Participant 3 was the only one to select Container for this question.

4.2.3 Results

This section reports the experiment results. We start by examining the effort required by the two security systems. Then, we analyse whether these results hold for individual applications. Next, we investigate the distribution of effort for different subtasks. Finally, we examine results by skills of participants.

4.2.3.1 Security System Effort Analysis

Table 4.11 shows that changing Spring-based applications ($\tilde{x}$ = 32.5 min) took 41.3% longer than Container-based ones ($\tilde{x}$ = 23.0 min). The Wilcoxon test reveals that this difference is statistically significant at $\alpha = 0.10$ (p = 0.007, V = 70.5). In practice, it means that the probability that the difference between *treatments* was achieved only by chance is discarded for this experiment. Moreover, the large effect size ($\gamma = 0.54$) and high statistical power (Power = 0.89) suggest that the difference between treatments is significant. Implications of this finding are discussed in Section 4.2.4.

The difference between *treatments* is not limited to medians, but it spans across other descriptive statistics. For example, the minimum effort for changing Spring-based applications (17 min) is 70% greater than that for Container-based applications (10 min). Similarly, the range effort for Spring-based applications (112 min) is 180%

Table 4.12: Descriptive statistics for the three applications. Results suggest that the application also impacts the effort.

Statistics	NisseiNet	Prototype	Siget
$\tilde{x}$ Container	23.50	44.00	23.00
$\tilde{x}$ Spring	41.50	37.00	22.50
$\tilde{x}$ difference (%)	76.60	-15.90	-2.20
p-value	0.01	0.41	0.10
Wilcoxon V	21	12	17
Effect Size (γ)	0.68	-0.23	-0.46
Range - Container	36.00	38.00	38.00
Range - Spring	86.00	56.00	112.00

greater than that for Container-based applications (40 min). These differences support the hypothesis that Spring requires more effort than Container to modify.

4.2.3.2 Effort Analysis by Applications

Table 4.12 shows that the difference between medians of the two security systems differ across the three applications used in this experiment. Note that, for this analysis, we take the Prototype application (used for pretest) into consideration as we are interested in observing the extent to which the pretest differs from other applications. Median differences between Container and Spring vary from -15.9% to 76.6%. Only the Prototype difference was not found statistically significant, though Siget is borderline ($\alpha = 0.10$).

NisseiNet had the largest effect size ($\gamma = 0.68$), followed by Siget (γ = -0.46) and Prototype (γ = -0.23). Effect sizes for Siget and Prototype are negative due to the order of the comparison, i.e., Container >Spring effort. Regarding the range, effort keeps larger for Spring than Container regardless of the application. These results suggest that the application has a fundamental role in the effort. In fact, this was noted by one of our participants as a comment in the *data collection form*.

One could argue that these differences are due to the different application sizes (Table 4.1). However, the task investigated in this experiment does not require major architectural modifications that would be influenced by size. Perhaps, the application size could influence in the time to analyse the code but, as Section 4.2.3.3 shows, the median time for analysis does not present any difference regarding the two *treatments*. Therefore, we do not believe that the size alone is the major responsible for these differences. Investigating security rules in the configuration file, we found that Prototype had the smallest number of rules, followed by NisseiNet and Siget (largest). Perhaps,

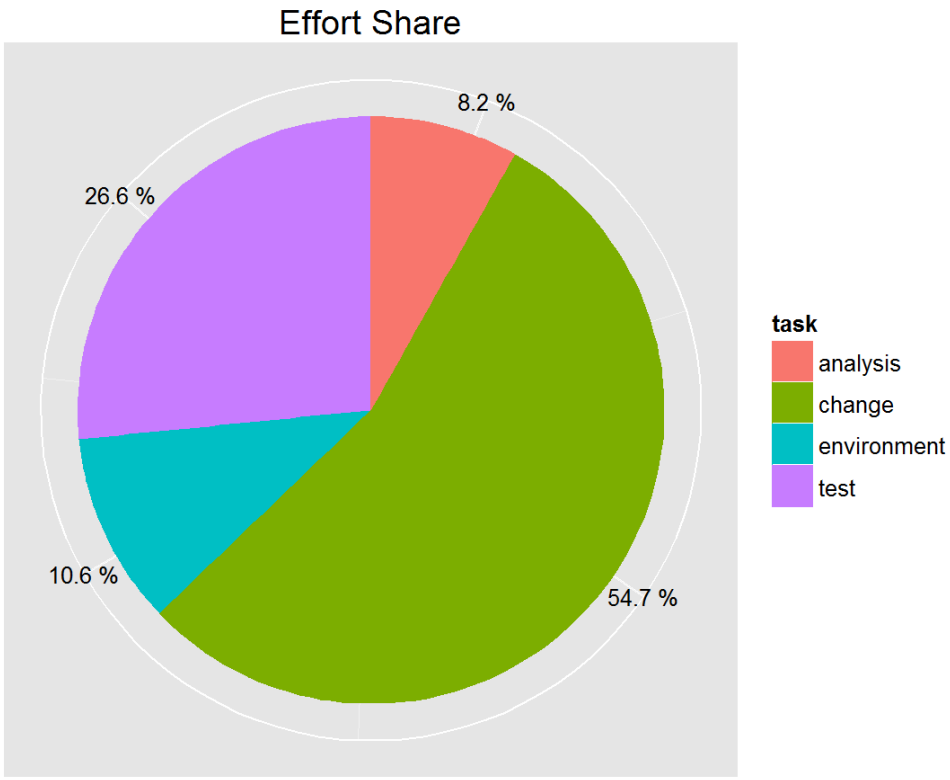


Figure 4.1: Division of effort amongst four subtasks for security system modification.

the difference is motivated by a combination of application and configuration size, but this is a question that needs further investigation.

4.2.3.3 Effort Analysis by Subtask

The effort share chart in Figure 4.1 shows that *Change* was by far the most time consuming subtask (54.7%) when compared with all others. *Test & corrections* also consumed a considerable time percentage in this experiment (26.6%). It is interesting to note that this *subtask* nearly had a rate of 1:2 compared with *Change*. Although this result does not differ from the maintenance literature [168], it might suggest that participants took a considerable amount of time correcting errors. The other two subtasks, *Code analysis* (8.2%) and *Environment preparation* (10.6%) took similar time share.

We also analysed the difference between the two *treatments* for each subtask (Table 4.13). For *Change* and *Environment preparation*, Table 4.13 shows difference in medi-

Table 4.13: Effort comparison by modification subtask.

Statistics	Environment Preparation	Code Analysis	Change	Test & Corrections
$\tilde{x}$ Container	3.00	2.00	12.50	3.50
$\tilde{x}$ Spring	4.00	2.00	21.00	3.50
$\tilde{x}$ difference (%)	33.33	0.00	68.00	0.00
Range - Container	9.00	10.00	17.00	20.00
Range - Spring	4.00	10.00	45.00	73.00

ans whereas for the other two substasks, medians are alike. For the first time in this experiment, we could observe a range (*Environment preparation*) for Container (9 min) greater than Spring (4 min). It suggests a major variance for preparing the environment for Container-based applications. This is a surprising result, as Spring required more libraries and preliminary configuration than Container.

For *Test & corrections* subtask, we can also observe a large range difference (265%) between Container (20 min) and Spring (73 min). This significant difference suggests that participants struggled more for testing and correcting errors in Spring-based applications than in Container-based. This might be result of the *learning effect* as Table 4.9 shows that the sum of skills (after training) for Container (22) is 22% greater than Spring (18). Possible implications of this result on the validity of this experiment are discussed in Section 4.2.5.

4.2.3.4 Effort Analysis by Participant

As we had few participants, we could easily analyse individual results (Figure 4.2). Possible implications of the small sample in the validity of this experiment are discussed in Section 4.2.5. Apart from participant 5 (Container = 31.5 min, Spring = 28.5 min, Difference = -9.5%), Spring took more time than Container to modify for all participants. Regarding participant 5, he is: an expert among participants, found Spring-based applications easier to change than Container-based ones, and evaluated his skills (after training) with a 4 for Spring, Container and OpenID. However, this participant noted in the *data collection form* that he found a bug while changing two Container-based applications. He noted that he unsuccessfully tried to solve this bug. Therefore, he did a workaround and kept working on the experiment. Unfortunately, we could not isolate this case as the participant did not identify on the *data collection form* the time he spent working on this bug. No other participant reported something alike.

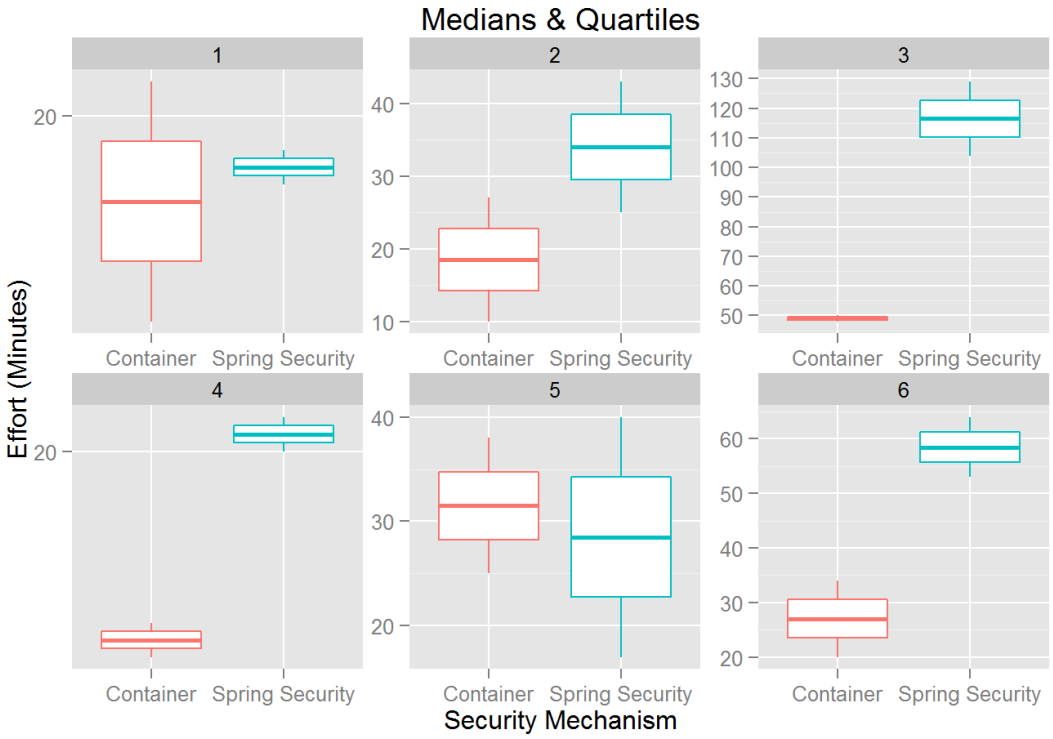


Figure 4.2: Effort comparison by participant. Note that the scale differs across boxplots.

Participant 3 had the greatest effort difference between Spring and Container amongst all participants (137.7%). He also took longer than others to change both Container- (49 min) and Spring-based (116.5 min) applications. This participant is not the least or most experienced (Java skills = 3, Skills compared with experienced developer = 2, Skills compared with their colleagues = 3), but he evaluated his skills (before training) for Container and Spring with grade 1. Although this participant declared a significant improvement in his skills after the training (Table 4.9), this result suggests that the training might not have been so effective for him as he declared. Possible implications of this result in the validity of this experiment are discussed in Section 4.2.5.

In fact, the size of differences across participants suggest that their background had also some impact on the total effort. For example, participant 6 was the less experienced (youngest, little time as professional, little time programming, few skills in Java). On the other hand, participant 1 was one of the two most experienced participants. When we compared both, we observe that the effort difference between Spring and Container is far greater for participant 6 (116%) than participant 1 (5.7%).

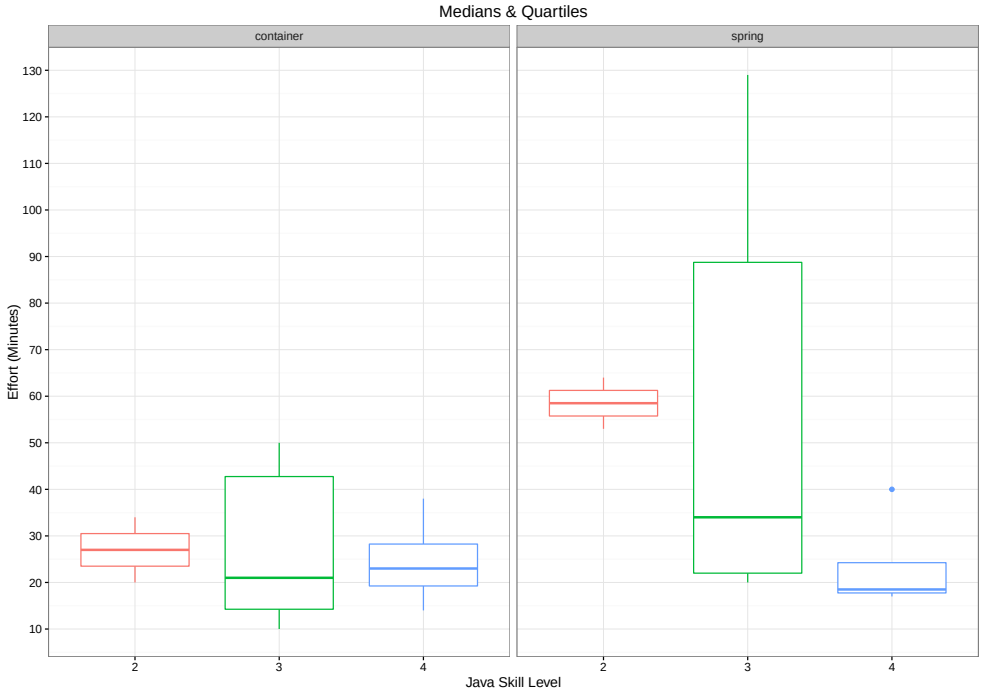


Figure 4.3: Impact analysis of skills on security system modification effort.

4.2.3.5 Effort Analysis by Skill

To investigate the impact of participants' development skills on the effort, we grouped results according to participants' Java skills (Figure 4.3). It is important to note that grades for Java skills and time developing as a professional include the same participants. Therefore, the result of our analysis would be exactly the same if we had considered the time developing as a professional rather than Java skills.

Figure 4.3 shows a clear decreasing trend of effort in function of skills when considering only Spring-based applications. However, we cannot observe the same trend when considering only Container-based applications. We applied *Spearman's* correlation to test the correlation between effort and skills. Whereas the correlation was both large and statistically significant for Spring (rho = -0.67, p = 0.007), it was negligible and no statistically significant for Container (rho = -0.04, p = 0.43). In addition, we applied *Kruskal-Wallis* to test for a statistically significant difference amongst effort medians for the three skill levels in each *treatment*. Results confirmed a statistically significant difference for medians in the Spring group (p = 0.07, chi-squared = 5.26), but not in the Container group (p = 0.977, chi-squared = 0.04). The causes for these differences need further investigation.

4.2.4 Discussion

This section discusses the impact of our results on the hypothesis testing and on the research question. We also discuss practical implications of our results.

4.2.4.1 Hypothesis Testing

The *null hypothesis* investigated in this experiment states that the median time to modify how the application authenticates the user is the same regardless of the security system adopted (Section 4.2.1.4). From the previous section, we can observe that the median time vary with the security system (Difference = 41.3%, p = 0.007, $\gamma = 0.54$, Power = 0.89).

As the *p-value* is significant at $\alpha = 0.10$, we **reject** the *null hypothesis* and **accept** the *alternative hypothesis* that states that the median time is greater for Spring than Container. It is important to highlight that even a smaller α, like $\alpha = 0.05$ or $\alpha = 0.01$, would not change the *null hypothesis* rejection for this experiment.

To analyse the relevance of this result, we compare its effect size and power with that obtained by Software Engineering (SE) experiments. According to the classification proposed by Kampenes et al. [120], the effect size for this experiment can be considered large ($0.456 < \gamma < 0.868$). Moreover, only 30% of experiments in SE achieved such a large effect ($\bar{x} = 0.34$).

For the statistical power, we consider the classification proposed by Dybå, Kampenes & Sjøberg [70]. For a large γ, like in this experiment, the statistical power achieved in this experiment is greater than the mean achieved by experiments in SE that used the *Wilcoxon* test ($\bar{x} = 0.74$). Moreover, when compared with all experiments analysed in [70], only 16% of experiments achieved such a high statistical power ($\bar{x} = 0.63$).

4.2.4.2 Research Question

Our research question is whether security systems impact on cloud application portability. The rejection of the *null hypothesis* and acceptance of the *alternative hypothesis* support the conclusion that, for this experiment, Container-based take less time than Spring-based applications to be modified. Thus, we can suggest that a cloud migration that requires similar modification to that performed in this experiment will be more efficient if the application's security system is based on Container rather than Spring. Hence, a Container-based is more portable than a Spring-based application, which answers our research question: *security system impacts cloud application portability.*

4.2.4.3 Implications

Once confirmed by independent replications, our results have two main implications. Numerous researchers from industry and academia have sought solutions that enable or increase cloud portability, typically based on the claim that lock-in is due to the use of cloud platforms with different semantics, technologies and interfaces [210]. Although these claims make sense from a practical perspective [211], they lack empirical evidence. This experiment provides an empirical evidence that the *application technology* impacts cloud portability too. This finding shows the need for studies investigating techniques to make applications more portable in cloud rather than only proposing new solutions for abstracting cloud differences.

Another practical implication of our findings bears on the recruitment of technical staff for performing migration activities. Taking into consideration the result reported in Section 4.2.3.5, a company could save some money by hiring technical staff with specific characteristics. For instance, for Spring-based applications, the company should hire experienced developers, as they took one-third of the time (18.5 min) that less experienced developers (58.5 min), as long as they do not cost three times more. On the other hand, for Container-based applications, the company should hire less experienced developers as their effort does not differ significantly from experienced developers.

4.2.5 Threats to Validity

This section discusses the main internal and external threats to the validity of this experiment.

4.2.5.1 Internal

Internal threats impact on the cause-effect relationship. These threats might lead to an alternative cause for the effect (known as *confound*) [215]. In this section, we identify four internal threats.

As Section 4.2.3.2 shows, the software application impacts the effort. As one of the researchers involved in this study re-engineered the applications for adopting a different security system, it might have biased our results. To mitigate this threat, we followed common conventions for adopting the Container security system[1], and we were careful to translate security rules from Spring into Container to keep exactly the same semantics.

[1] `http://docs.oracle.com/javaee/7/tutorial/partsecurity.htm`

Adopting two different APIs for implementing OpenID Connect protocol might have introduced some bias in our results as one API could require more effort than other (Section 4.2.1.6). However, this was a necessary risk. The Container API would require critical and very complex modifications in order to be adopted by Spring, whereas the Spring-API could not be adopted by Container-based applications as they are specific for Spring. This might be a confound variable for this experiment, and the complexity of these APIs should be evaluated in isolation to identify its confounding effect.

Using one XML-based (Spring) and another code-based (Container) security system is a possible threat because some participants could be more skilled in one technology than another. In addition, this heterogeneity in our variables can be a confounding factor. As explained in Section 4.2.1.3, it was necessary due to the version of applications and the Spring version used. Like the use of different OpenID Connect APIs, this might be a confound variable that should be evaluated in isolation to identify its confounding effect in future studies.

As Section 4.2.3.3 shows, the training session and the pretest might not have been enough to prepare participants and rule out all *learning effects*. Although some participants had some knowledge on OpenID, no one had experience in OpenID implementations. Therefore, we can assume that all participants learnt at the same time, but we cannot guarantee that all of them acquired similar skills after training.

Finally, the small sample size could also affect our conclusions. However, it is important to emphasise that the experiment achieved a large effect size and a high statistical power when compared to experiments in Software Engineering (Section 4.2.4.1). Moreover, Section 4.2.3.5 shows that results do not differ across participants, apart from one participant. Nevertheless, more participants should be considered for future replications of the experiment.

4.2.5.2 External

External threats restrict the result generalisation beyond the scope of the study [256]. In this section, we identify two external threats.

Applications used in this experiment are a very small sample of the target *objects* for this experiment (Section 4.2.1.2). This decision was necessary to keep the experiment simple and possible to execute in the limited amount of time. These applications are small information systems, and therefore, they limit the generalisation of this result beyond this category.

However, it is worth to note that these are real commercial applications. Future

replications of this experiment should increase the number and type of applications analysed to enable wide generalisations.

Using participants from the same company might affect our power to generalise our results. Although it is not uncommon to limit studies to a single group [94, 195], we recognise that it can restrict our generalisation. However, from our work experience with SMEs, we believe that these participants are a good representative of developers employed by Brazilian software development companies.

4.3 Building Prediction Models

This section details the exploratory process of building regression models for predicting the effort of increasing cloud application portability by modifying the security system of legacy three-tier web applications to enable the use of single sign-on by adopting implementations of the OpenID Connect protocol. By using concepts and techniques explained in Appendix C and results from the experiment reported in Section 4.2, we built a set of simple and multiple linear regression models using OLS and, alternatively, robust regression methods. Leave-one-out cross-validation (LOO-CV) was used to evaluate the prediction models. The prediction model accuracy was assessed by using the MMRE summary measure. Additionally, we compared the accuracy of our models with maintainability prediction models reported in Section C.4. The rationale for techniques used in this section is detailed in Appendix C.

4.3.1 Simple OLS Linear Regression Model

The experiment reported in Section 4.2 shows that security system can be used as an indicator of cloud portability when adoping SSO by using OpenID Connect implementations. Therefore, we built a simple linear regression model by using the security system as a single predictor. Figure 4.4 shows a jitter plot with all 24 observations and a model line representing the prediction model. Unlike the prediction model reported in Section 3.4.1, the single predictor is a categorical variable with only two values representing the two security systems evaluated in our experiment.

Table 4.14 summarises the accuracy obtained with LOO-CV, and the goodness of fit for this simple model. According to the classification presented in Section C.4 for maintainability prediction models, the simple model #1 achieved a *fair* accuracy ($0.49 < \text{MMRE} \leq 0.88$). Similar to the simple model in Section 3.4, the Pred(0.37) shows that 41.7% of predictions using this model could be classified as *excellent*.

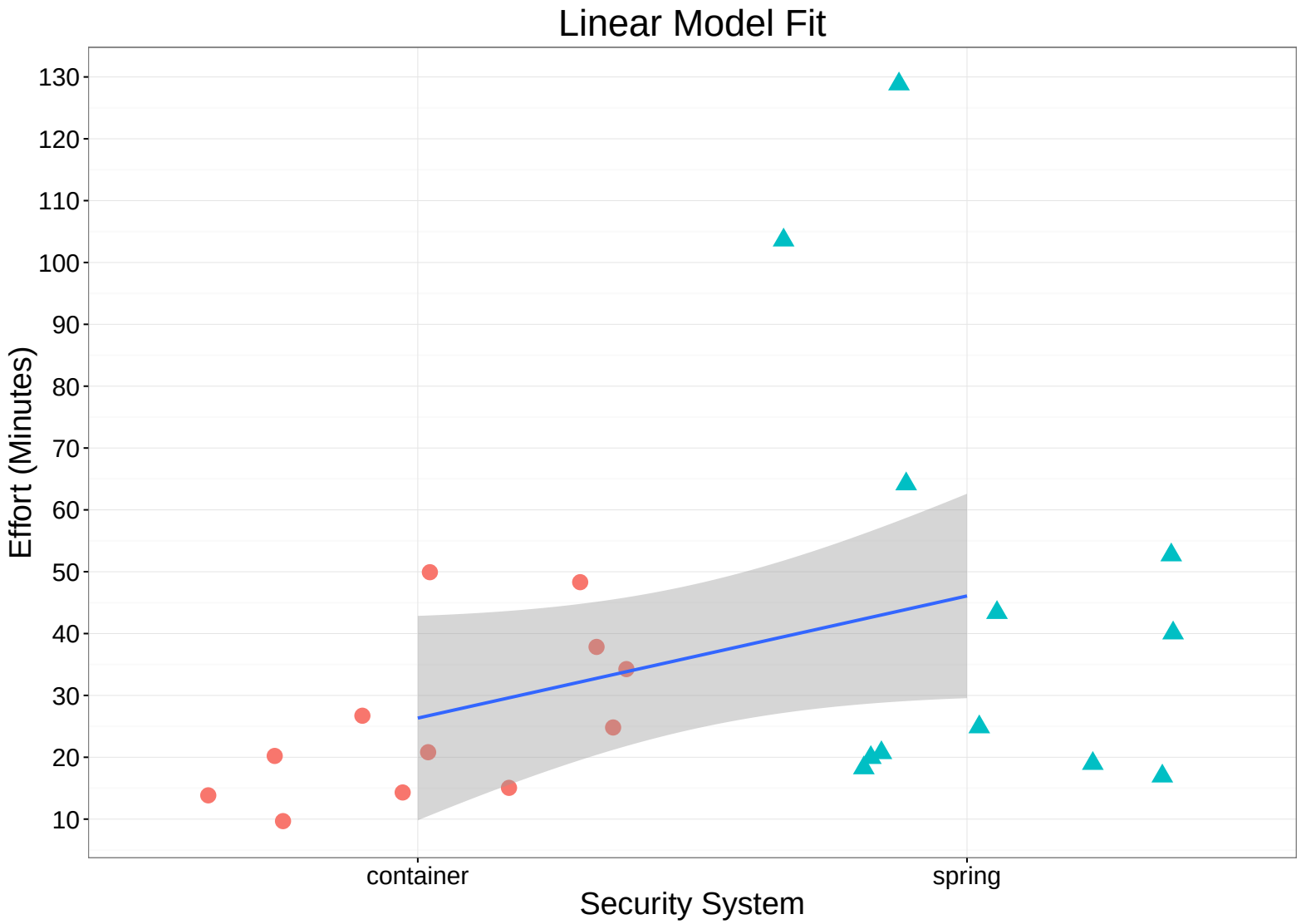


Figure 4.4: Effort prediction model line using security system as a single predictor.

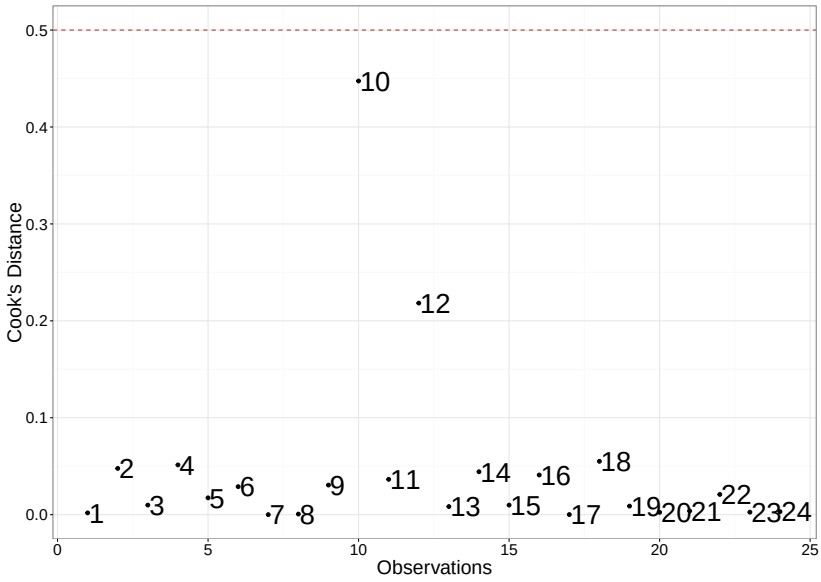


Figure 4.5: Cook's distance plot of data points. No observation is acknowledged as an outlier though two observations differ from the rest.

However, the goodness of fit ($R^2 = 0.122$; RSE = 27.600) is significantly lower than that for the simple model in Section 3.4 ($R^2 = 0.224$; RSE = 15.130).

4.3.1.1 Outlier analysis

As outliers can influence the model, we tried to identify possible outliers and quantify their impact on the model. The plot in Figure 4.4 suggests two outlier candidates with Effort >100 min (Observations 10 and 12). In addition to the jitter plot, we also used the Cook's distance to identify outlier candidates. As Figure 4.5 shows, no observation is above the cuttoff for Cook's distance (0.5) though observations 10 and 12 differ from most observations.

We analysed observations 10 and 12 to check whether they were influential cases for the model. These two observations came from participant 3 when modifying the Spring security system for the two applications used in the experiment. This participant took the highest amount of effort for Spring applications amongst all participants (Section 4.2.3). Therefore, these two observations were not considered as outliers.

Table 4.14: Accuracy and goodness of fit for effort prediction models built. Variables are identified according to Table 4.15

#	Prediction Model (Variables)	Accuracy Measures					Goodness of fit	
		MMRE	MAR	Pred(0.25)	Pred(0.30)	Pred(0.37)	R^2	RSE
1	Simple model (Security System)	0.733	21.000	0.250	0.292	0.417	0.122	27.600
2	Simple robust model (Security System)	0.581	19.685	0.167	0.250	0.375	0.021	17.190
3	Multiple model (Security System + #1)	0.728	20.191	0.125	0.250	0.417	0.289	25.420
4	Multiple model (Security System + #2)	0.758	21.147	0.208	0.333	0.333	0.272	25.730
5	Multiple model (Security System + #3)	0.719	20.534	0.292	0.333	0.375	0.193	27.090
6	Multiple model (Security System + #4)	0.767	21.141	0.292	0.292	0.333	0.216	26.700
7	Multiple model (Security System + #3 + #1)	0.738	20.754	0.250	0.292	0.417	0.294	25.970
8	Multiple model (Security System + #3 + #2)	0.804	22.098	0.208	0.292	0.292	0.272	26.360
9	Multiple model (Security System + #3 + #4)	0.787	21.750	0.250	0.292	0.292	0.216	27.360
10	Multiple model (Security System + #3 + app.name)	0.756	21.613	0.250	0.292	0.333	0.193	27.750
11	Multiple model (Security System + #3 + time.p.dev)	0.787	21.750	0.250	0.292	0.292	0.216	27.360
12	Multiple robust model (Security System + #3)	0.561	19.014	0.167	0.250	0.250	0.100	20.970

4.3.1.2 Simple Model Diagnostics

As explained in Section C.5, meeting some underlying assumptions is a pre-condition to accurately generalise a model beyond its data set. We tested the extent to which model #1 meet seven underlying assumptions for simple linear regression models based on the OLS method. Figure 4.6 shows four plots to visually support the diagnostics.

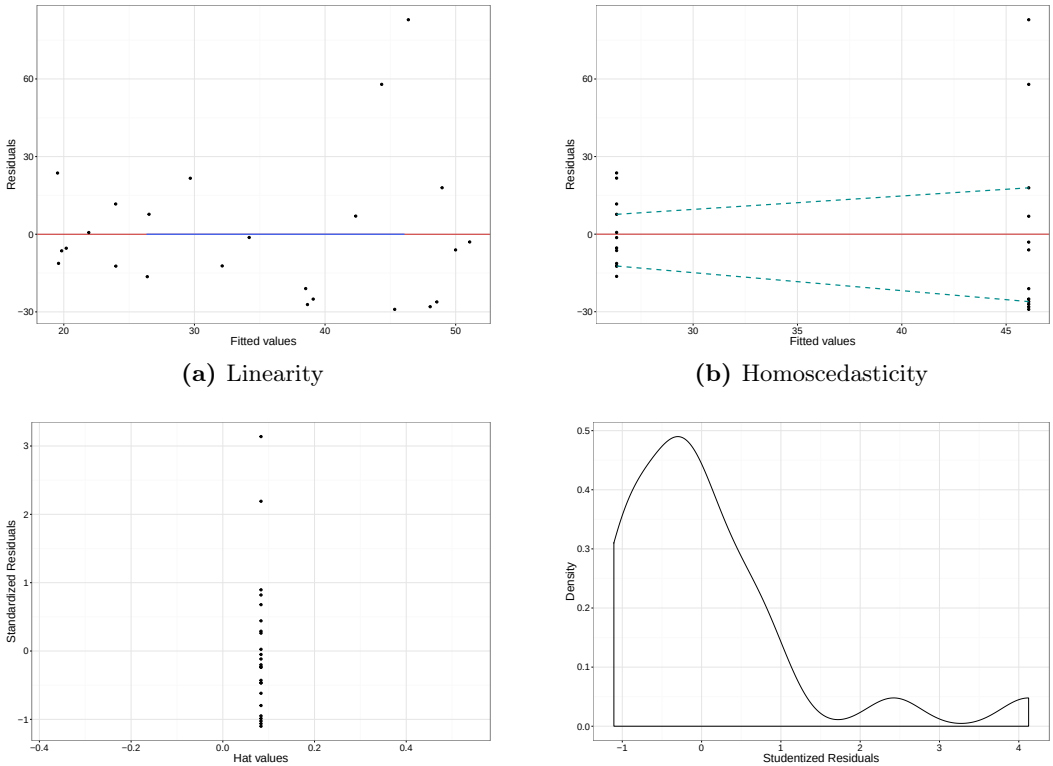
Sample size. No observation was removed from the original sample. Therefore, the simple model was based on the full sample that consists of 24 observations. As only one predictor was used for building the model, the sample-predictor rate is within the acceptable range (i.e., ≥ 10).

Linearity. Figure 4.6 (a) shows the jitter plot for residuals and fitted values of the simple model. Data points are scattered across the plot around the red line (middle), suggesting a homogeneous distribution. The blue line, which supports the recognition of patterns, overlaps the red line in the middle. Additionally, the Rainbow test confirms the linearity assumption fulfilment (Rain = 0.324, p = 0.965).

Independent residual terms (errors). We tested the lack of autocorrelation for residual terms by using Durbin-Watson test (DW = 1.172, p = 0.008). The DW statistics is within a critical region, suggesting that errors are not independent. Furthermore, the p-value is significant for $\alpha = 0.05$, confirming that the simple model #1 does not meet this assumption.

Homoscedasticity. To meet the homoscedasticity assumption, the variance of residual terms should be homogeneous. Figure 4.6 (b) helps to identify the homogeneity of the variance. The two dashed lines suggest a funnel pattern, which could mean a violation of homoscedasticity (i.e., the presence of heteroscedasticity). However, we can observe a homogeneous distribution of data points around the middle (red line). Finally, the Goldfeld-Quandt test confirms that the simple model meet this assumption for $\alpha = 0.05$ (GQ = 7.403, p = 0.001).

Low leverage. The plot in Figure 4.6 (c) suggests no major leverage issues as data points are centred around zero (y-axis) and concentrated below the leverage cutoff (0.25). Therefore, we considered that the simple model meet this assumption.



(a) Linearity **(b)** Homoscedasticity

(c) Low Leverage **(d)** Normality

Figure 4.6: Diagnostics plots for effort prediction model #1

No perfect multicollinearity. As the simple model has only one predictor, there is no risk of multicollinearity.

Normality. The density plot in Figure 4.6 (d) shows a positive skew, which suggests deviation from normality. Indeed, as the Shapiro-Wilk test confirms, the simple model violated the normality assumption (W = 0.851, p = 0.002).

4.3.1.3 Alternative Simple Robust Regression Model

As explained in Section C.5, robust regression is a method less sensitive to outliers and make no assumption regarding the distribution type. This section uses this method to test whether it can improve the prediction accuracy of our regression model. Additionally, for the simple model #1, the robust regression is an alternative to overcome the lack of normality. Table 4.14 shows the accuracy and goodness of fit for the simple robust model #2.

Table 4.15: Ratio variables in our data set and their correlation with *effort*.

#	Ratio variables	rho	S	p
1	participant.openid.knowledge	-0.502	3456.8	0.012
2	participant.spring.skills	-0.496	3441.5	0.013
3	participant.java.skills	-0.349	3104.1	0.094
4	participant.container.skills	-0.300	2990.8	0.153

Although we can observe a significant improvement in the MMRE accuracy (-20.7%) and RSE goodness of fit (-37.7%) when compared to model #1, Pred measures got worse as well as the R^2.

4.3.2 Multiple OLS Linear Regression Model

A multiple linear regression model is similar to a simple model, but it includes more than one predictor. We built a set of multiple linear models to investigate whether other variables in our data set could improve the prediction accuracy of our simple model. Firstly, we analysed possible predictor candidates. Secondly, we applied the forward stepwise method to identify the impact of predictor candidates in our model. Next, we analysed the existence of outliers. Then, we checked whether the best multiple linear model meet underlying assumptions for multiple linear regression models based on OLS. Finally, a multiple robust regression model was built to evaluate the impact of this method in the prediction accuracy.

4.3.2.1 Analysis of Predictor Candidates

As the literature recommends [84, 206], we quantified the correlation between ratio variables in our data set and our dependent variable (effort) to identify predictor candidates. As Table 4.15 shows, *rho* values vary from -0.300 (lowest) to -0.502 (highest). To be considered as a predictor candidate, we set $rho = 0.193$ as the minimum cutoff. This value represents the minimum value for medium correlations in SE experiments [120]. As all variables are above the cutoff, all of them were considered as possible predictors for the forward stepwise method.

4.3.2.2 Forward Stepwise Method

This method was used to define whether a predictor candidate should enter the multiple regression model. The entry condition was a MMRE $<$0.733, which was the best

accuracy achieved by the simple OLS regression model. The stepwise method consists of several iterations. For each iteration, (i) one predictor candidate is added to the model, (ii) the model accuracy is re-evaluated, and (iii) the entry criterion is applied to decide whether the predictor candidate should be kept in the model. As the variable that represents the security system is already in the model, we omitted this step and started with the second predictor candidate (variable #1 in Table 4.15).

Iterations 1, 2, 3 and 4 Prediction models #3–#6 were built in these four iterations as an attempt to find a second predictor that could improve the accuracy of our simple model #1. As Table 4.14 shows, model #5 (MMRE = 0.719) achieved the best accuracy amongst all models built in these iterations (-1.9% better than model #1). Thus, variable #3, which was used to produce model #5, was kept in the model and a new set of iterations was performed.

Iterations 5, 6 and 7 For the second round of iterations (models #7–#9), we tried to find a third predictor to improve the prediction accuracy of our model. As Table 4.14 shows, no variable could achieve this goal although models in this round presented a far better R^2.

Iterations 8 and 9 The experiment presented in Section 4.2 shows that the Web application used in the experiment has some impact on the effort for changing the security system. The Web application is represented by a categorical variable in our data set. Therefore, for this set of iterations, we forced the entry of two categorical variables as an attempt to improve the accuracy. However, models #10 and #11, produced in these iterations, did not improve the accuracy (Table 4.14).

4.3.2.3 Outlier analysis

This section tries to identify possible outliers and quantify their impact on the multiple model #5. As there are multiple predictors in the model, it is no longer possible to infer outliers just by inspecting the jitter plot. Therefore, we used the Cook's distance to identify possible outliers.

The result of Cook's distance analysis (Figure 4.7) does not differ from that of the simple model (Figure 4.5). As no observation was removed from the data set, the cutoff for Cook's distance remains the same. No observation was found above the cutoff, and observations 10 and 12 still differ from most observations. As previously explained in Section 4.3.1.1, these are not outliers.

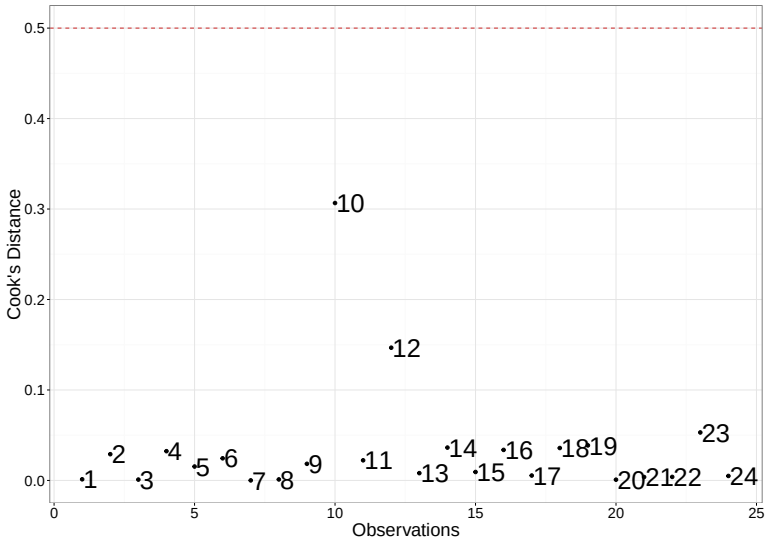


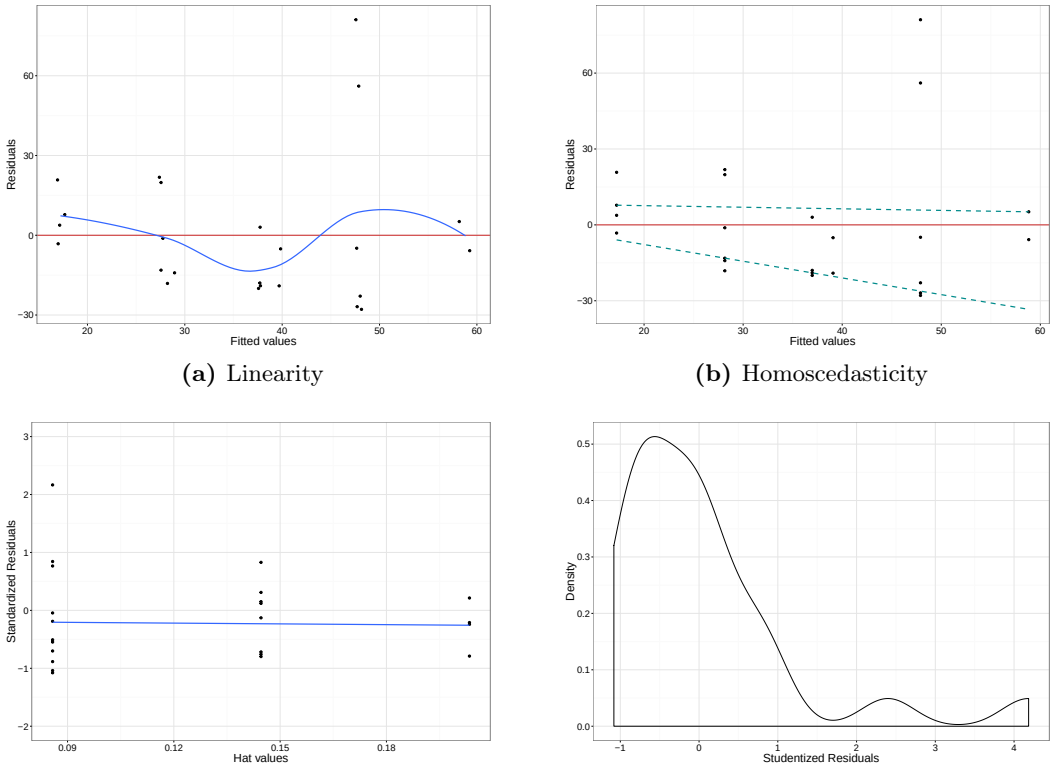
Figure 4.7: Cook's distance plot for the multiple effort prediction model #5.

As a matter of investigation, we removed observations 10 and 12 and re-built model #5. We noted a significant improvement (-23.7%) in the accuracy (MMRE = 0.548). We also observed an improvement in the goodness of fit (R^2 = 0.209, RSE = 14.020). However, removing these two observations would result in a major change in our data set as they represent the real effort that one of our participants took to perform the experiment task. This change would also unbalance the distribution of observations since these two observations refer to the Spring security system. Therefore, we decided for keeping these observations in the data set.

4.3.2.4 Multiple Model Diagnostics

As for the simple model, we carried out a set of tests to identify the extent to which the best multiple model meet seven underlying assumptions for multiple linear regression models based on the OLS method. As a reference, we used model #5, as it obtained the best accuracy amongst all models built (Table 4.14). Figure 4.8 shows four plots to support the diagnostics.

Sample size. The multiple model #5 consists of two predictors. Our original sample consists of 24 observations, which means a sample-predictor rate of 12. As this is greater than the minimum cutoff of 10, the model #5 meet this assumption.



(a) Linearity **(b)** Homoscedasticity

(c) Low Leverage **(d)** Normality

Figure 4.8: Diagnostics plots for effort prediction model #5

Linearity. Figure 4.8 (a) shows the jitter plot for residuals and fitted values of the multiple model #5. Data points are scattered across the plot around the red line (middle) although the blue line suggests a pattern of ups and downs. To test the assumption that the model has no linearity issues, we applied the Rainbow test that confirms that model #5 meet the linearity assumption (Rain = 0.278, p = 0.978).

Independent residual terms (errors). We tested the lack of autocorrelation for residual terms by using Durbin-Watson test (DW = 1.289, p = 0.018). Like in the simple model, the multiple model also does not meet this assumption.

Homoscedasticity. To meet the homoscedasticity assumption, the variance of residual terms should be homogeneous. Figure 4.8 (b) helps to identify the homogeneity of the variance. Like for the simple model (Figure 4.6 (b)) the two dashed lines suggest a funnel pattern, which could indicate the presence of heteroscedasticity. Although we can observe a homogeneous distribution of data points around the middle (red line), we

applied the Goldfeld-Quandt to test whether the homoscedasticity assumption is met. The test result (GQ = 6.271, p = 0.005) is significant for $\alpha = 0.05$, which confirms the assumption fulfilment.

Low leverage. The average leverage for this model is 0.125. Therefore, the cutoff for analysis is three times this average (0.375). The plot in Figure 4.8 (c) suggests no major leverage issues as data points are centred around zero (y-axis) and we can observe no data point above the cutoff. Therefore, we considered that model #5 meet the low leverage assumption.

No perfect multicollinearity. Unlike the simple model #1, the multiple model #5 might violate the multicollinearity assumption as it has 2 predictors. To test this assumption, we checked the variance inflation factor (VIF) that indicates whether a predictor has strong relationship with other predictors. The VIF calculated for both predictors was equals to 1. Consequently, the tolerance (1/VIF) was also equals to 1. As the values for the VIF are lower than 10 and the tolerance is greater than 0.2, we considered that the model #5 meet this assumption.

Normality. As for the simple model #1, the density plot in Figure 4.8 (d) suggests a positive skew. As confirmed by the Shapiro-Wilk test, model #5 also deviates from normality (W = 0.830, p < 0.001).

4.3.2.5 Alternative Multiple Robust Regression Model

As model #12 in Table 4.14 shows, the accuracy of our robust model improved -21.6% when compared to the multiple model #5, reducing the MMRE to 0.561. However, all Pred measures decreased as well as the R^2. It is important to note that the multiple robust model produced a better accuracy than the simple robust model (MMRE = 0.581), unlike that reported in Section 3.4.2.5.

4.3.3 Discussion

From 12 prediction models built in Sections 4.3.1 and 4.3.2, the multiple robust model #12 achieved the best accuracy as measured by the MMRE summary measure (MMRE = 0.561). This model consists of two predictors, the variable that represents security systems and the Java skills of participants. Although the goodness of fit improved nearly four times ($R^2 = 0.1$) when compared to the simple robust model ($R^2 = 0.021$),

it is surprisingly low when compared to the multiple robust model built in the previous chapter ($R^2 = 0.824$).

According to the classification presented in Section C.4, model #12 achieved a *fair* accuracy ($0.49 < \text{MMRE} \leq 0.88$), a bit lower than that achieved by the best model in the previous chapter (MMRE = 0.412). In addition, only a quarter of predictions made by model #12 could be classified as *excellent*. This result is one-third lower than the result of multiple OLS model #5, which has the same predictors.

Although model #12 can be used to predict the effort of increasing cloud application portability within the context defined in Section 4.2, its generalisation beyond this data set is limited. The robust model is less sensitive to the lack of normality, present in the multiple OLS models, but it is not robust to the violation of independent errors. In future studies, machine learning approaches that are robust to this type of violation can be used to enable the model generalisation beyond this data set.

4.4 Summary

This chapter represents the second step to addressing the objectives defined in Section 1.2. We empirically investigated a technology, I&A security systems, to assess whether it impacts cloud application portability. The result of the experiment in Section 4.2 confirmed our hypothesis - security system does impact cloud application portability within the context defined for the experiment. In addition, the experiment achieved an effect size and a statistical power that are among the highest of the SE experiments in the literature. Moreover, the use of software developers working on their usual environment and using their usual tools increased significantly the realism of our experiment.

However, this experiment was conducted within a scope and different results could be obtained if authorisation was used instead of authentication. Similarly, using a technology other than security systems could lead to different results although we strongly believe that most technologies would produce similar results. Increasing the generalisation of our findings is an important future work (Section 6.3).

Additionally, we used the data set produced in the experiment to build 12 prediction models by using two different prediction methods for linear regression. Our best model achieved a *fair* accuracy when compared to other prediction models reported in the related literature (Section C.4). However, the ability to generalise our prediction models beyond this data set is limited due to the violation of independent errors assumption. Nevertheless, results presented in this chapter can potentially contribute to the outcomes envisaged for this research.

Chapter 5

Investigating the Impact of Cloud Platforms and Services on Cloud Application Portability

Cloud platforms and services are two key factors of a cloud migration since they represent the target environment (Sections 2.3 and 2.4). In addition, as presented in Section 2.2.3, semantics, technologies and APIs used by cloud platforms have been regarded as the main causes for cloud lock-in, which, in turn, hinders cloud portability. Moreover, three out of four activities in the cloud migration process involve the cloud platform and service (Section 2.4.4). Therefore, it makes sense to investigate these two key factors.

This chapter empirically investigates the impact of cloud platforms and services on cloud application portability in the context of configuring cloud services (Section 5.2) and deploying cloud applications (Section 5.3). Unlike previous chapters, this investigation required two experiments to be carried out. Therefore, we present in Sections 5.1 and 5.4 common aspects of both experiments. We explore the outcomes of both investigations by building prediction models to support decision makers when analysing the required effort to configure cloud services (Section 5.5) and deploy cloud applications (Section 5.6).

5.1 Empirical Investigations

As explained in Section 1.2, we used guidelines provided in [256] to prepare and conduct the two experiments. This section details aspects that are common for both experiments undertaken. Sections 5.2 and 5.3 provide further information regarding protocol and results of each experiment. Terminology associated with the experimentation framework is highlighted in the main text; this terminology is explained in Appendix B.

5.1.1 Experiment Plan and Execution

5.1.1.1 Participant Selection and Study Design

Fourth-year undergraduate students of two different universities were invited to take part in our experiments. Their acceptance was the only criterion considered for their participation (*convenience sampling*). The first group of students was enrolled in the Information Systems course at UNIPAR - Paranavaí (UNIPAR) whereas the second group was enrolled in the Computer Science course at the University of York (UoY). The former is a Brazilian university whereas the latter is a British university. Participants were rewarded with vouchers in the equivalent value of £10.00 for the time they dedicated to this study (about five hours, see Section 5.1.1.3).

The first group was the *control group* (CG) whereas the second group was the *treatment group* (TG). Both groups initially consisted of 12 participants. Therefore, we had a *balanced between-subject* design. However, throughout training and experimentation sessions, some participants gave up taking part in the activities (*drop-outs*). We ended up with nine participants in the CG. For the TG, the number of participants varied according to the experiment. Whereas for the experiment reported in Section 5.2, the TG consisted of four participants, six participants took part in the experiment in Section 5.3. These *drop-outs* impacted on the design, leading us to adopt an *unbalanced* design. The impact of *drop-outs* and sample size on results are discussed in Section 5.4.2.

5.1.1.2 Data Analysis

As explained in Section 1.2, we adopted only *non-parametric* statistics in this experiment as summarised in Table 5.1 whereas cutoffs we used for inferential tests are summarised in Table 5.2.

Table 5.1: Summary of statistics and statistical tests adopted in this study.

Statistic Type	Statistic/Test	Configuration	Purpose
Descriptive	Median and quartiles	25%, 50%, 75%, 100%	To identify the middle value and categorise occurrences into quartiles according to their values.
	Range	Maximum - Minimum	To quantify the data dispersion.
Inferential	Kolmogorov - Smirnov	Two-tailed	To test if two distributions come from the same population.
	Shapiro-Wilk	Two-tailed, $\alpha = 0.1$	To test the assumption that the data is normally distributed.
	Wilcoxon rank-sum	Two-tailed, one-tailed	To test the statistical significance of differences between two medians.
	Fisher's Exact Test	Two-tailed	To test the relationship between two categorical variables.
	Spearman's correlation	Two-tailed	To test the strength of a relationship between two numerical variables.

5.1.1.3 Execution of Experiments

This study was carried out in two parts: training and experimentation. The training session covered cloud concepts and prepared participants to perform the task required for each experiment. The total training time was three hours. Subsequently, each experiment required about 1 hour to be completed.

In our first meeting with the participants, an introduction script was handed out to all of them. This script provided an overview of each experiment, highlighting their purpose, the task that participants should perform, and benefits of taking part in this experiment. Those who accepted taking part in the training session were asked to fill out the *participant characterisation form*. This form collected information regarding the participants' background, such as previous knowledge on cloud computing. Questions were based on guidelines proposed in [82].

Just before starting the first experimentation session, participants were asked to sign

Table 5.2: Cutoff values for evaluating the hypotheses. Table adapted from [70].

		Unknown true state of nature	
		H_0	H_1
Statistical Conclusion	Accept H_0	0.95 1 - α: Correct *(Confidence level)*	0.70 β: Type II error
	Reject H_0	0.05 α: Type I error *(Significance level)*	0.30 1 - β: Correct *(Statistical power)*

a *participant consent form*. Next, each participant received a unique ID that was used to identify them throughout the experiment. This ID bound participants and *objects*.

During the experiment execution, participants were free to get in and out of the room, e.g. go to the toilet, as long as this time off was not counted as part of the task performed in the experiment. The experimenter observed the experiment execution without interfering in tasks performed by participants. Participants were strongly advised to not copy and paste any data as it could bias the result.

When the experiment finished, participants were asked to fill a *feedback form*, reporting their experience in this experiment. Participants received their vouchers and participation certificates in a few weeks. Room and materials used for training and experiment sessions were prepared by the experimenter before each session. Materials consisted of computers with Internet connection, slides used during training sessions, forms used during the experiment and guidelines for performing the experiment. For information on ethics in this experiment, refer to Section 1.3.

5.1.2 Characterisation of Participants

This section examines the results of *participation characterisation form* (Section 5.1.1.3). We have separated responses into two groups: one relating to self-evaluation and one relating to prior experience. Tables 5.3 and 5.4 show the frequency of answers for each question according to the university.

Table 5.3 shows the result of self-evaluation questions. The participants were asked to evaluate themselves according to three questions. Their evaluation varies from zero (lowest) to five (highest). Table 5.3 shows that participants vary in their knowledge of Java, Java for Web and cloud.

Table 5.4 shows how participants answered questions about their prior experience. Most participants had neither professional (questions 6 and 10) nor practical experience

Table 5.3: Summary of participants' knowledge on programming and cloud computing.

#	Evaluation Question	UNIPAR						UoY					
		0	1	2	3	4	5	0	1	2	3	4	5
1	How would you grade your current knowledge about Java programming?	0	0	1	5	3	0	0	0	2	0	4	0
2	How would you grade your current knowledge about software development for the Web with Java?	0	0	1	6	2	0	5	1	0	0	0	0
3	How would you grade your knowledge about cloud computing BEFORE this training?	1	1	5	1	1	0	1	1	2	1	1	0

with cloud APIs (question 8) and deployment (question 9). On the other hand, most of them had used some cloud services (question 7).

These differences between participants from the two universities could raise some questions about the validity of our design, described in Section 5.1.1.1. Therefore, we checked the homogeneity of participants by applying *Fisher's Exact Test*. In the context of this study, a relationship between questions and universities means that participants are heterogeneous (i.e., their profile varies according to the university).

Apart from questions two and three, the test shows an independent relationship between answers and universities ($p > 0.05$), suggesting that participants are homogeneous regarding the aspects investigated by questions. Note that questions two and three were not essential for any conclusion in this study since this study involved no programming activity and required no previous development knowledge other than that taught during their undergraduate course.

5.2 Empirical Investigation - Cloud Platform

The goal of this experiment is to analyse two cloud platforms for the purpose of evaluation with respect to their impact on the effort to increase cloud application portability from the point of view of the software developers in the context of undergraduate

Table 5.4: Summary of participants' practical experience on programming, cloud and maintenance.

#	Experience Question	UNIPAR		UoY	
		No	Yes	No	Yes
6	Considering any programming language, have you ever developed an application professionally?	8	1	4	2
7	Have you ever used any cloud service?	0	9	2	4
8	Have you ever used any cloud API?	9	0	6	0
9	Have you ever deployed any application in a cloud service?	8	1	4	2
10	Have you ever changed an application developed by a third party?	3	6	5	1

students configuring a cloud service by creating message queues in two public cloud platforms. In the next sections, we detail the experiment protocol (Section 5.2.1) and show the main results of our experiment (Section 5.2.2).

5.2.1 Experiment Plan

This section details specific aspects of this experiment, such as instruments, task, variables and hypotheses.

5.2.1.1 Instrumentation

In this experiment, two cloud platforms are investigated (*treatments*): Amazon Web Services (AWS) and Microsoft Azure (Azure). Participants within the CG were assigned to AWS and participants within the TG were assigned to Azure. Both platforms provide types of services that are functionally equivalent, such as VM, storage and message queuing. A cloud service was used as the *object* since it is not possible to evaluate the cloud platform directly [142]. The type of service selected for this experiment is the message queuing service (*object*). As explained in Section 2.4.5, adopting message queuing is essential to decouple application components so as to enable their individual scalability and portability within and across cloud platforms. In addition, the current focus of our research provided the rationale for this choice (Section 1.2). The two services used were Amazon Simple Queue Service[1] for AWS, and Service Bus[2] for Azure.

[1] `http://aws.amazon.com/sqs/`
[2] `http://azure.microsoft.com/en-us/services/service-bus/`

5.2.1.2 Task

The *task* performed in this experiment consists of configuring a cloud service by creating message queues. As explained in Sections 2.4 and 3.2, message queues are critical to integrate components of distributed systems. The procedure for creating message queues is slightly different for each service, and can be performed in different ways even for the same service. Hence, this task can be considered as *non-deterministic*, as the randomness involved in how the task is performed might affect the result of effort, e.g. by using shortcuts provided by the platforms.

In summary, the task consists of performing the following steps:

1. Signing in to the platform;
2. Accessing the service;
3. Creating a single queue by providing a unique name. This step is performed a number of times;
4. Signing out of the platform.

For the Azure platform, an additional step was required: the creation of a namespace for a set of queues. This step can be done before or during the third step. Note that no preferred region or additional configuration was required, such as the size of queues. Participants registered the start and end time on an online form just before starting the first step, and after completing the last step. Four *trials* were performed, each of them consisted of a set of queues as presented in Table 5.5.

Our previous experience re-engineering applications to adopt message queuing provided the rationale for the number of queues considered, as follows. A single entity, like a *Customer* in a sales management system, needs at least five queues. Each queue is responsible for the communication of one CRUD[1] operation. The exception is the *Read* operation that requires two queues - one to provide the request and another to the response. In practice, this number of queues cover small applications varying from 5 to 20 entities. This number is a balance between what we found in small companies, and what we believed participants would be able to do within the available time.

5.2.1.3 Selection of Variables and Definition of Hypotheses

The cloud platform represents the *independent* variable, or *factor*, evaluated in this experiment. AWS and Azure are the two possible values (*treatments*), measured on

[1] Create, Read, Update and Delete

Table 5.5: Trials performed and # of queues considered in this experiment. The # of queues was randomly assigned to trials.

Trial	Number of Queues per Platform	
	AWS	Azure
1	25	100
2	50	25
3	100	75
4	75	50

a *categorical scale*. The effort is the *dependent* variable, measured in the number of minutes[1] (*ratio scale*) to perform the task presented in Section 5.2.1.2.

Two hypothesis were defined for this experiment, as formalised in Table 5.6. T is the number of minutes spent to create message queues in each trial. Thus, the *null hypothesis* states that the median time is the same regardless of the platform whereas the *alternative hypothesis* states the contrary.

Table 5.6: Formal definition of hypotheses.

Null hypothesis (H_0)	$\tilde{x}\,T_{\text{AWS}} = \tilde{x}\,T_{\text{Azure}}$
Alternative hypothesis (H_1)	$\tilde{x}\,T_{\text{AWS}} \neq \tilde{x}\,T_{\text{Azure}}$

5.2.2 Results

5.2.2.1 Analysis of Individual Platforms

Figures 5.1 and 5.2 show the effort distribution across different groups of queues considered in this experiment. For the AWS platform (Figure 5.1), apart from a single group of queues (# queues = 75), we can note an increasing trend in the data distribution across the occurrences of different treatments. This difference might have been motivated by the *last trial/task effect* [20], which means that participants could be in a hurry to finish the task, or competing against them to be the first to finish. It could have motivated participants to apply more effort in this last trial. This is a possible threat to validity that is addressed in Section 5.4.2.

We can also observe an overlap of several data points from different treatments in

[1]We use the terms *effort* and *time* interchangeably throughout this study.

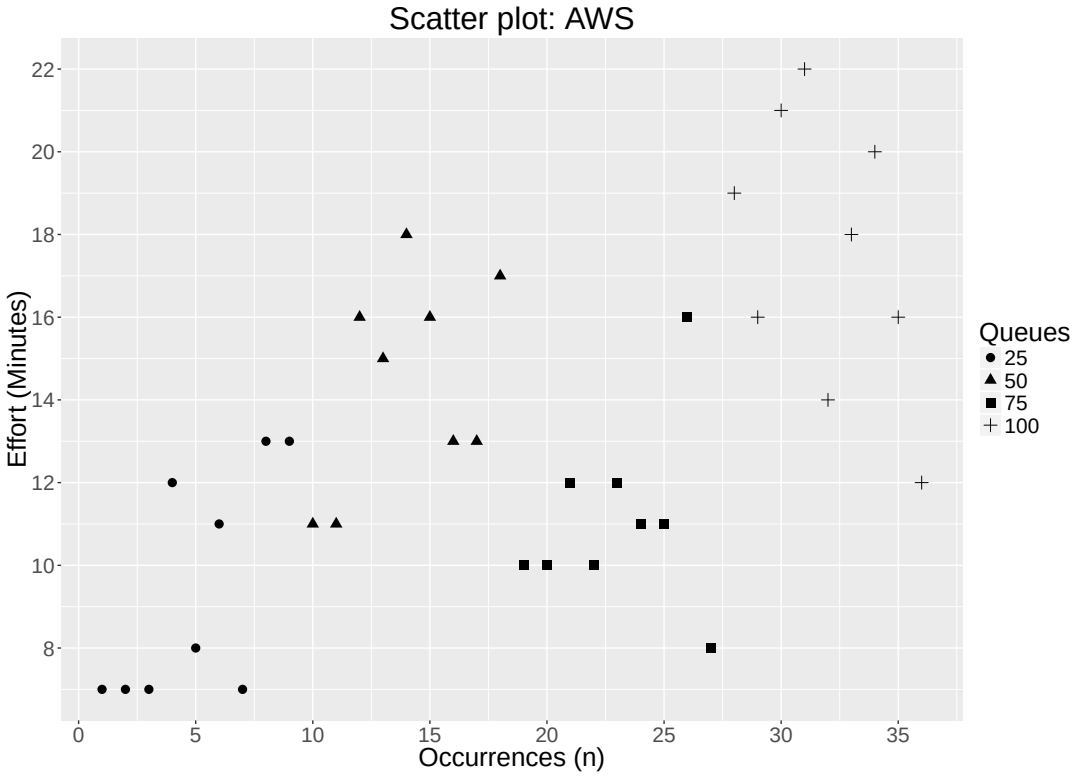


Figure 5.1: Data point distribution for the AWS platform. Increasing trend across group of queues and overlap of data points in the y-axis.

the y-axis, in the range from 10 to 14 minutes. This overlap comprises all four treatments, suggesting a non-uniform distribution of effort among treatments. In practice, it suggests that the effort is not directly related to the number of queues.

Similar to the AWS, the Azure platform (Figure 5.2) shows an increasing trend of effort across occurrences of the four treatments. However, unlike the AWS platform, the data distribution for Azure does not present any overlap between treatments, apart from two data points in the y-axis, in the range from 10 to 15 minutes. Although we can note some differences between AWS and Azure graphs, the *Kolmogorov-Smirnov* test indicates that both distributions are similar ($p = 0.39$, D = 0.27).

By analysing the scatter plots, we can also observe the frequency of data distributions. For example, we can note that effort varies from 7 to 22 minutes (range = 15 min) for the AWS platform. In contrast, the Azure platform had a larger range, varying from 4 to 30 minutes (range = 26 min). It shows that, considering all treatments, the effort varied more for Azure than for AWS.

To test whether the data distribution is normal, we applied the *Shapiro-Wilk* test

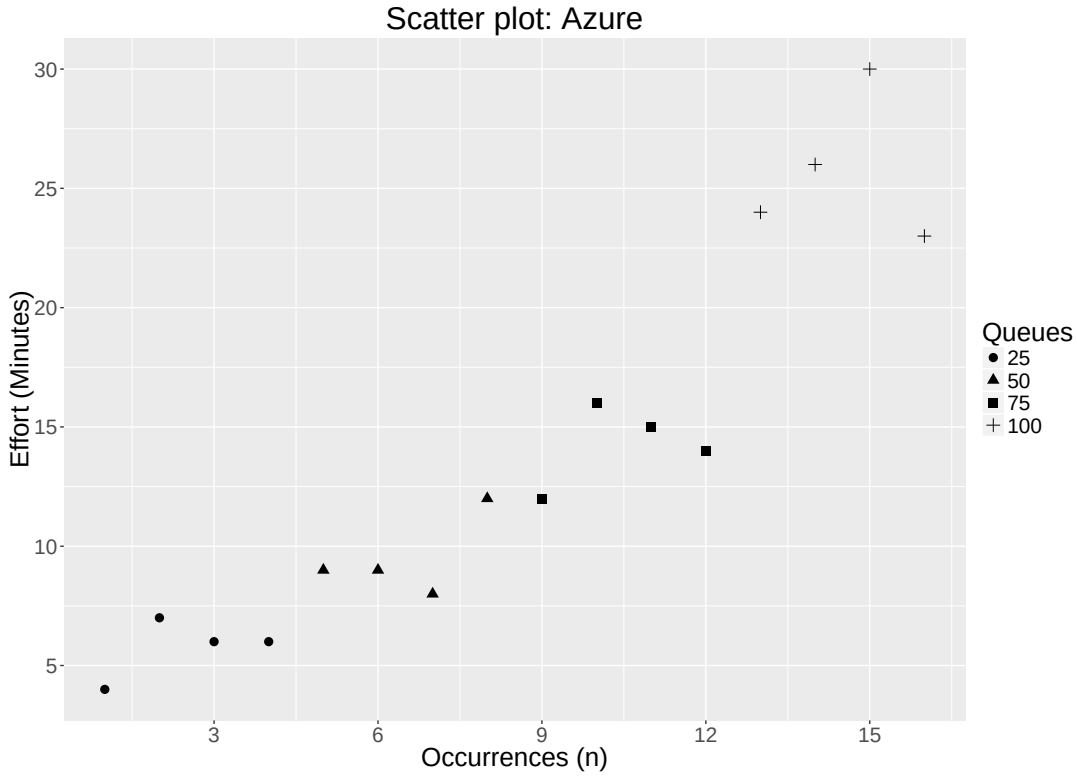


Figure 5.2: Data point distribution for the Azure platform. There is an apparent linear increase across group of queues and nearly no overlap.

for each treatment for both platforms, as each treatment is an independent distribution. Table 5.7 presents W and p values of normality tests. For the AWS platform, apart from one treatment (# queues = 25), all treatments present normal distributions (p >0.10). For the Azure platform, all treatments presented normal distributions. Thus, both *parametric* and *non-parametric* tests could be applied to test most data collected. As defined in Section 5.1.1.2, we concentrate only on *non-parametric* statistics, however.

Figures 5.3 and 5.4 show medians and quartiles for each treatment, for both platforms. These boxplots quantify data points shown in Figures 5.1 and 5.2. For the AWS platform, apart from one treatment (# queues = 75), the boxplot shows an increasing trend of medians, suggesting that the effort varies according to the # of queues. We can also observe an overlap between lower and upper quartiles among three treatments (# queues = 25, 50 and 100).

Unlike the AWS platform, the Azure boxplot shows an increasing trend across all treatments and no overlap among them. Another difference between platforms is that variances are larger for the AWS than for the Azure when considering treatments in-

Table 5.7: Data normality test for both platforms. Apart from one distribution for AWS platform, all distributions are considered normal.

# Queues	AWS		Azure	
	W	p	W	p
25	0.77892	0.0117	0.89495	0.4064
50	0.92393	0.4257	0.8397	0.1945
75	0.8798	0.1563	0.97137	0.85
100	0.96663	0.8645	0.9202	0.5381

dependently. This result is the opposite of the result obtained when analysing all treatments together. For example, the effort for the first group of queues (# queues = 25) varies from 11 to 18 for AWS platform (range = 7 min) whereas it varies from 4 to 7 for Azure (range = 3 min).

To test the correlation between effort and number of queues, we used the *Spearman's* correlation coefficient (rho). The *Spearman's* correlation test indicates a large significant correlation between effort and the number of queues for both data sets (rho >0.5, p <0.001). However, the Azure platform (rho = 0.96) achieved a correlation far larger than AWS (rho = 0.53). In practice, it shows that for both platforms, increasing the number of queues also increases the effort, but this is more evident for Azure than for AWS platform.

5.2.2.2 Comparison of Platforms

Figure 5.5 summarises medians and quartiles for the four trials performed in this experiment. The first boxplot (25, top-left) shows medians and quartiles for the first group of queues (# queues = 25). The boxplot shows that the AWS median effort ($\tilde{x}$ = 8.0) is 33.3% greater than the effort for Azure ($\tilde{x}$ = 6.0). In addition, we can observe that there is no overlap between medians and quartiles from the two platforms. The *Wilcoxon* test confirms a significant difference between medians (p = 0.01) and the effect size test shows a large effect size (γ = -0.68). Analogously, the *post hoc* power of the test is also large (Power = 0.63).

The second boxplot (50, top-right) shows medians and quartiles for the second group of queues (# queues = 50). Similar to the first group, the boxplot shows the Azure effort ($\tilde{x}$ = 9.0) outperforming AWS ($\tilde{x}$ = 15.0). In this trial, the AWS median time was 66.6% greater than Azure. Neither the boxes nor the whiskers from the two platforms overlap. The *Wilcoxon* test indicates a significant difference between medians (p =

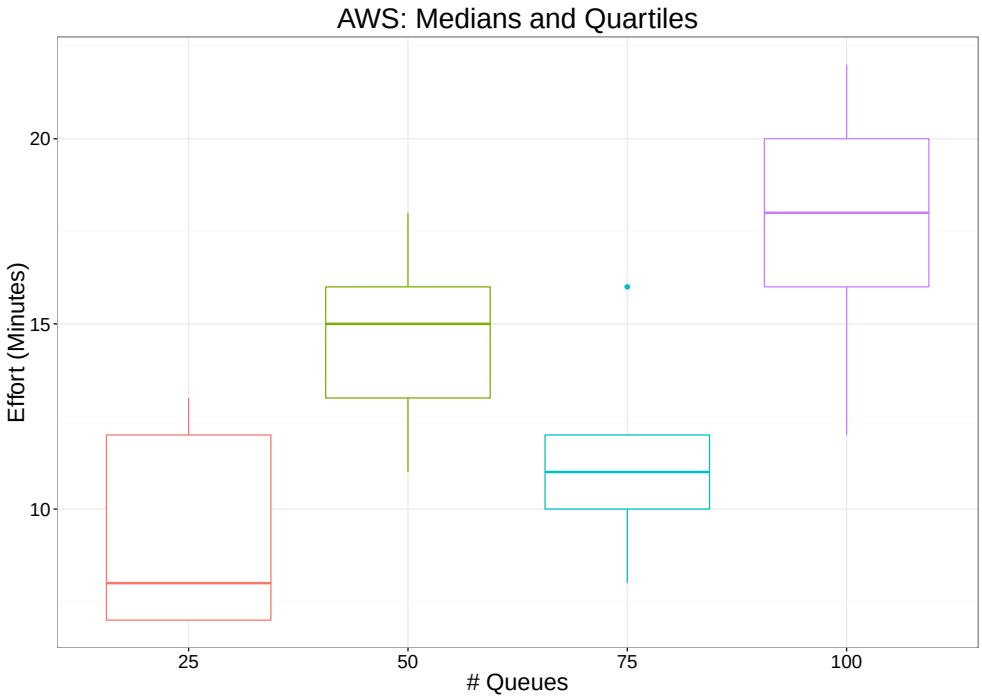


Figure 5.3: Medians and quartiles for AWS platform. Upper and lower quartiles overlap.

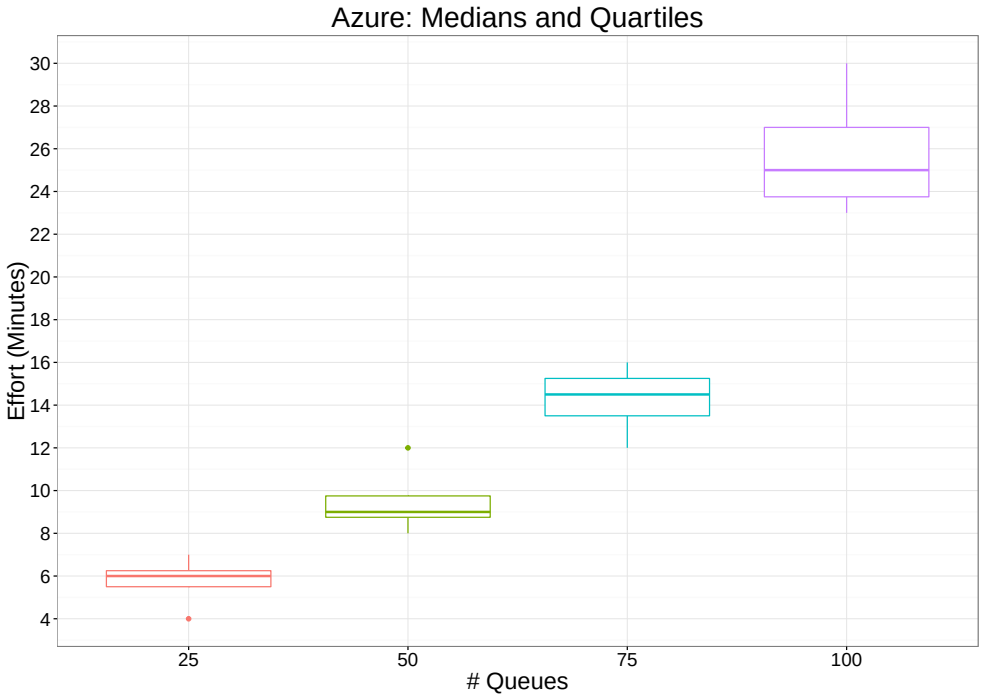


Figure 5.4: Medians and quartiles for Azure platform. Increasing trend across treatments.

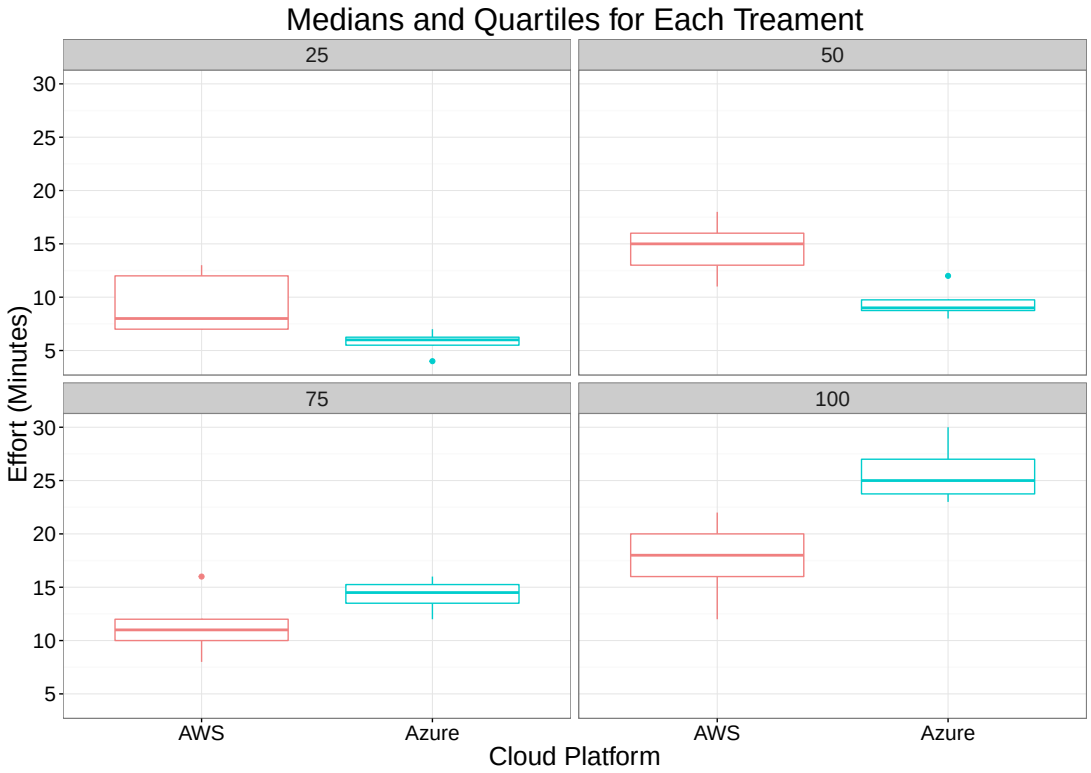


Figure 5.5: Effort comparison for different treatments. Surprising pattern change between the first and last two treatments.

0.01). The effect size is similar to the first group of queues (γ = -0.66), and the power is even higher (Power = 0.88).

Unlike the first two trials, the boxplot of the third group (# queues = 75) shows an inversion of results (75, bottom-left). Azure and AWS swap positions when compared with previous trials. The boxplot shows AWS ($\tilde{x}$ = 11.0) outperforming Azure ($\tilde{x}$ = 14.5). In this trial, Azure median time was 31.8% longer than AWS. Likened to the first two trials, there is no overlap between either boxes or whiskers from the two platforms. The *Wilcoxon* test confirms that there is a significant difference between the two medians (p = 0.04). Similar to previous trials, the effect size is large (γ = 0.56), and the *post hoc* power test presents a value similar to the first group of queues (Power = 0.62).

Like the previous trial, the boxplot of the last trial (# queues = 100) shows that the median effort for Azure ($\tilde{x}$ = 25.0) is 38.8% greater than that calculated for AWS ($\tilde{x}$ = 18.0), as observed in Figure 5.5 (100, bottom-right). Following on the same result of previous trials, boxes and whiskers from the two platforms do not overlap. The difference between medians is significant (p = 0.006). In addition, this group has the

highest effect size ($\gamma = 0.74$) and power (Power = 0.96) among all four groups of queues. Table 5.9 summarises the descriptive statistics presented in this section.

It is important to note that the sample size in this experiment is relatively small. Therefore, we also calculated the confidence intervals of medians using the bootstrap technique (Table 5.8). This technique can be used to increase the confidence by resampling data [258]. Apart from the third trial (# queues = 75), there is no overlap between confidence intervals. This result suggests that medians were calculated from different populations, confirming previous results from *Wilcoxon* tests. For the third trial (# queues = 75), the AWS upper limit (12) overlaps the Azure lower limit. Nevertheless, it is not enough to conclude that medians come from the same population, given the effect size and power of previously applied statistical tests for this trial.

Table 5.8: 95% confidence intervals calculated by using the bootstrap technique. The lack of overlap confirms the *Wilcoxon* test.

Treatment	Platform	Lower limit	Median	Upper limit
# Queues = 25	AWS	7.0	8.0	12.0
	Azure	4.0	6.0	6.0
# Queues = 50	AWS	11.0	15.0	16.0
	Azure	8.0	9.0	9.0
# Queues = 75	AWS	10.0	11.0	12.0
	Azure	12.0	14.5	15.5
# Queues = 100	AWS	14.0	18.0	20.0
	Azure	23.0	25.0	28.0

5.2.2.3 Hypothesis Testing

The *null hypothesis* investigated in this experiment states that the median time to create message queues is the same regardless of the cloud platform adopted. From the previous section, we can observe that medians of time vary with platform. For two trials (# queues = 25 and 50), the AWS platform required more effort than Azure to create message queues. For other two remaining trials (# queues = 75 and 100), the Azure platform required more effort than AWS. For all four comparisons, the inferential statistics confirmed significant differences at $\alpha = 0.05$. In addition, the effect size calculated for this experiment can be considered large according to values observed in Software Engineering (SE) studies, and standard conventions [120], as well as the statistical power [70]. In addition, 95% confidence interval does not present any major

Table 5.9: Summary of descriptive statistics for each trial.

Treatment	Platform	Min	1st Quartile	Median	Mean	3rd Quartile	Max
# Queues = 25	AWS	7.0	7.0	8.0	9.4	12.0	13.0
	Azure	4.0	5.5	6.0	5.8	6.3	7.0
	(%) Variance	-42	-21	-25	-38	-47	-46
# Queues = 50	AWS	11.0	13.0	15.0	14.4	16.0	18.0
	Azure	8.0	8.8	9.0	9.5	9.8	12.0
	(%) Variance	-27	-32	-40	-34	-38	-33
# Queues = 75	AWS	8.0	10.0	11.0	11.1	12.0	16.0
	Azure	12.0	13.5	14.5	14.3	15.3	16.0
	(%) Variance	50	35	31	28	27	0
# Queues = 100	AWS	12.0	16.0	18.0	17.6	20.0	22.0
	Azure	23.0	23.8	25.0	25.8	27.0	30.0
	(%) Variance	91	48	38	46	35	36

overlap, confirming the inferential tests. Therefore, for this experiment, taking into account the experimental design set up, we **reject** the *null hypothesis* and **accept** the *alternative hypothesis* that states that the median time to create message queues varies according to the targeted cloud platform.

5.3 Empirical Investigation - Cloud Service

The goal of this experiment is to analyse cloud services. We aim to evaluate cloud services with respect to their impact on the effort to increase cloud application portability from the point of view of the software developers. This evaluation is performed in the context of undergraduate students deploying cloud applications. To deploy these applications, our participants used two purpose-equivalent cloud services offered by a single cloud platform (AWS). In the next sections, we detail the experiment protocol (Section 5.3.1) and show the main results of our experiment (Section 5.3.2).

5.3.1 Experiment Plan

This section details specific aspects of this experiment, such as instruments, task, variables and hypotheses.

5.3.1.1 Instrumentation

The AWS platform was the platform used in this experiment. AWS was chosen because it is one of the most prominent public cloud platforms [140]. Two purpose-equivalent services were considered (*treatments*): Container and Virtual Machine (VM). Whereas the container service provides limited access to underlying configurations, such as parameters of the application server, the VM service provides wider access to a range of different configurations, including operating system parameters.

In this experiment, the container service is represented by Amazon Beanstalk[1] whereas the VM is represented by Amazon EC2[2] service. Participants in the CG were assigned to the container service whereas participants in the TG were assigned to the VM service. A single *war* file was used as the *object* in this experiment. This file represents the application deployed in the cloud service. The application is a simple web application that prints a welcome message.

[1] `http://aws.amazon.com/elasticbeanstalk`
[2] `http://aws.amazon.com/ec2/`

5.3.1.2 Task

As explained in Section 2.3, tasks in the software migration process vary significantly in the literature although an overall process can be identified. The software migration consists of transferring software components from one cloud to another, therefore, requiring the re-deployment of original, modified and possibly new application components (Section 2.4.4). Therefore, re-deployment is a critical task to realise the scenario target of this research, as presented in Section 2.4.4.

The *task* consists of deploying a small application (*object*), representing application components, a number of times (*trials*) by using the cloud service. The steps to deploy the application are:

1. Signing in to the platform;
2. Accessing the service;
3. Configuring an instance of the service;
4. Uploading and deploying the *war* file;
5. Signing out of the platform.

It is important to note that steps three and four vary substantially depending on the targeted cloud service. For example, for the container service, a new project (*Application*, in the AWS vocabulary) and environment need to be defined. In addition, uploading and deploying the application is done as a single step. For the VM service, configuring an instance of the service consists of starting a new VM. It requires that a set of configurations is defined. In addition, uploading and deploying the application is done in two separate steps[1]. As in the first experiment, the aforementioned steps might vary even for the same service (i.e., the process is *non-deterministic*).

Participants registered the start and end time on an online form just before starting the first step, and after completing the last step. Note that participants did not wait until the deployment has finished, i.e., the time the platform took to deploy the application was not taken into consideration for measuring the deployment effort. Each participant deployed the same application four times (*trials*).

[1]This process might be addressed in different ways. For example, a tool can be used to upload and deploy at once. However, our experience with small companies show that these steps are often addressed as two separated activities, as in this experiment.

5.3.1.3 Selection of Variables and Definition of Hypotheses

The cloud service is the *independent* variable, or *factor*, evaluated in this experiment. Container and VM, represented by Amazon Beanstalk and Amazon EC2 services respectively, are the two possible values (*treatments*), measured in a *categorical scale*. The effort is the *dependent* variable, measured in the number of minutes (*ratio scale*) to deploy the application in the cloud service.

Two hypothesis were defined for this experiment, as formalised in Table 5.10. T is the number of minutes spent to deploy the application for each trial. The *null hypothesis* states that the median time to deploy the application is the same regardless of the cloud service. Unlike the first experiment, the *alternative hypothesis* does not state the contrary, but that the median time for deploying by using the VM service is greater than that by using the container service. This assumption is based on our previous experience working with both services.

Table 5.10: Formal definition of hypotheses.

Null hypothesis (H_0)	$\tilde{x}\,T_{\text{VM}} = \tilde{x}\,T_{\text{Container}}$
Alternative hypothesis (H_1)	$\tilde{x}\,T_{\text{VM}} > \tilde{x}\,T_{\text{Container}}$

5.3.2 Results

5.3.2.1 Analysis of Individual Services

Figures 5.6 and 5.7 show the distribution of deployment effort for container and VM services, respectively. Data points are scattered across occurrences; some patterns can be observed. For example, the VM service chart shows a sequence of ups and downs, varying from 3 to 7 minutes. Investigating the data set, we identified that these occurrences are linked to the first occurrences of each participant, which are often longer than others. We assign this fact to the *learning effect*. It is common in experiments that participants take more time to perform the first task as they are getting familiar with the experiment [131]. This is a possible threat to validity that is addressed in Section 5.4.2.

Scatter plots suggest that the range of effort for occurrences is similar for both VM-based and containter-based deployments - most occurrences took <6 minutes. It is important to note that the sample size differs between two services (n(container) =

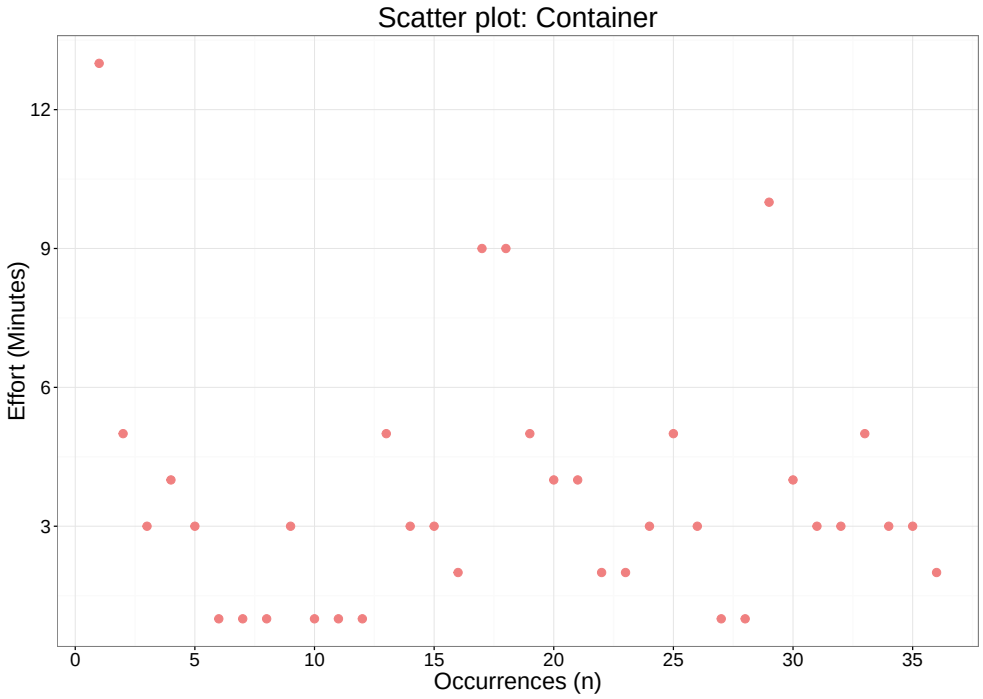


Figure 5.6: Deployment effort for the container service. Data points are scattered.

36, n(VM) = 24). This difference is due to the unexpected *drop-outs* (Section 5.1.1.1).

Some outliers can also be observed. For example, for the container service, four deployments took 9 minutes or more whereas for the VM service, three deployments took more than 10 minutes. Two aspects might have contributed for these outliers. Firstly, sometimes, we observed an abnormal response time from the cloud platform. Secondly, boredom or tiredness might have influenced participants in these few cases. These threats are addressed in more detail in Section 5.4.2.

Figures 5.8 and 5.9 show the frequency of deployment distribution for both container and VM service. Histograms show that most container-based deployments (75%) took less than 5 minutes. In contrast with container-based, only 42% of VM-based deployments took less than 5 minutes. It suggests that container service was faster to participants to deploy than VM. In addition, for the container service, the two most common occurrences of effort were three minutes (n = 11), followed by one minute (n = 8). For the VM service, the two most common occurrences of effort were four minutes (n = 7), followed by five minutes (n = 5). These two most common occurrences represent nearly half-percent of all occurrences for both container-based (53%) and VM-based (50%) deployments.

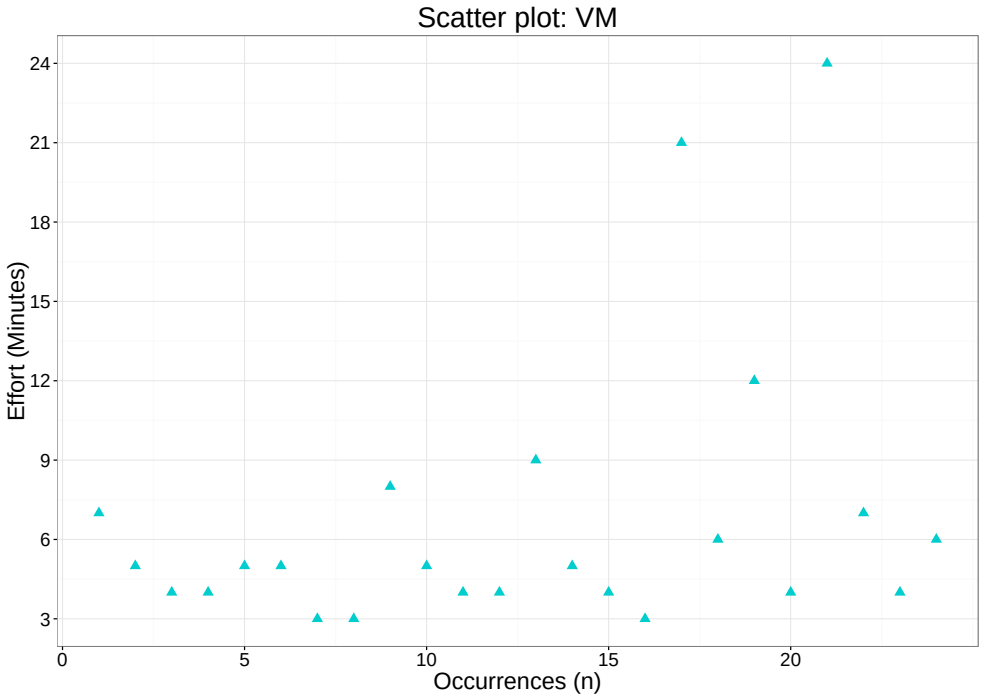


Figure 5.7: Deployment effort for the VM service. First occurrences took more time than others due to the *learning effect*.

Both histograms present long right tails, suggesting a non-normal distribution (positive skewness). We checked the data normality by applying *Shapiro-Wilk* test (Table 5.11). Both distributions are considered non-normal according to the test (p <0.1).

Table 5.11: Normality test for container-based and VM-based deployments. Both distributions are considered non-normal.

Service Type	W	p
Container-based	0.7951	<0.0001
VM-based	0.64083	<0.0001

5.3.2.2 Comparison of Services

Figure 5.10 shows medians and quartiles for container and VM services. The boxplot shows that the median effort for VM-based deployments ($\tilde{x}$ = 5 min) is 66.6% greater than that calculated for the container-based ($\tilde{x}$ = 3 min). In addition, we can observe an overlap between the half-percent higher values of container-based and half-percent lower values of VM-based deployments. This overlap indicates that VM-based deployments

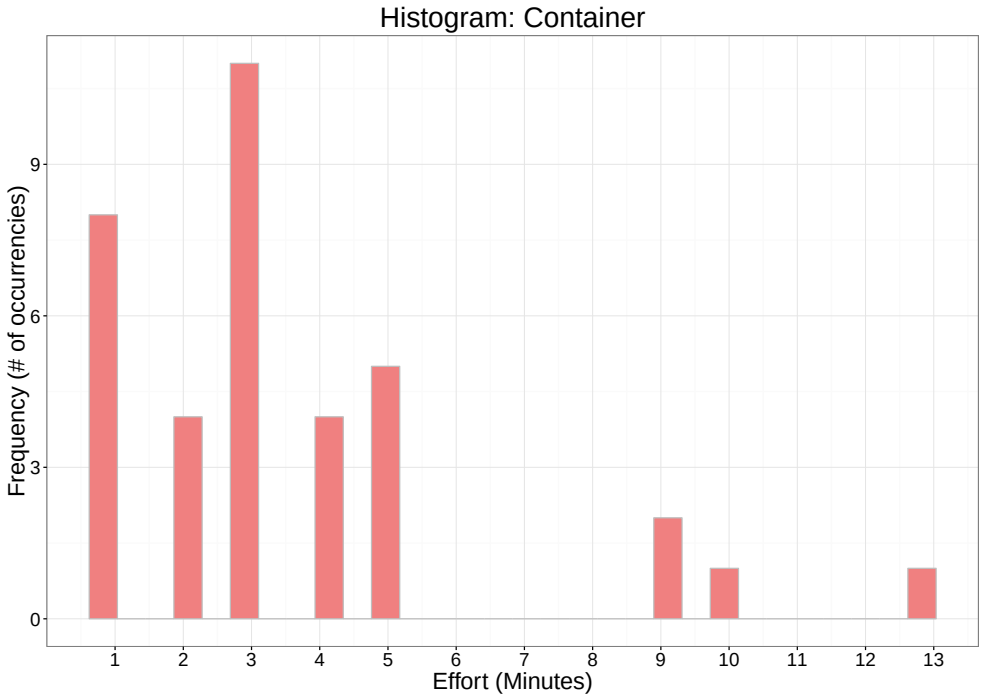


Figure 5.8: Histogram for container service. Most occurrences took less than 5 minutes (75%).

took considerably more time than container-based. For example, whereas 4 minutes was considered high effort for container service (3rd quartile), for the VM service it was considered low effort (1st quartile). Table 5.12 summarises further descriptive statistics.

Figures show that the median effort for VM-based deployments was 66% greater than container-based. However, this difference is not homogeneously distributed across quartiles. For example, in the first quartile, the VM service was 100% slower than the container service. In addition, the range of effort also varies across services.

The range of effort for VM-based deployments (range = 21) is significantly larger than that calculated for the container-based (range = 12). This larger variation can be explained by longer human procedures to deploy an application by using the VM service. These longer procedures are human-dependent, therefore, subject to greater variance.

We tested the statistical significance of the difference between medians of both data sets by applying *Wilcoxon* test. As presented in Section 5.3.1, this was a one-tailed test. The test confirmed that the difference is significant at $\alpha = 0.05$ (W = 675, $p = 0.0001$). Moreover, the effect size and power of this test is high ($\gamma = 0.50$, Power = 0.90).

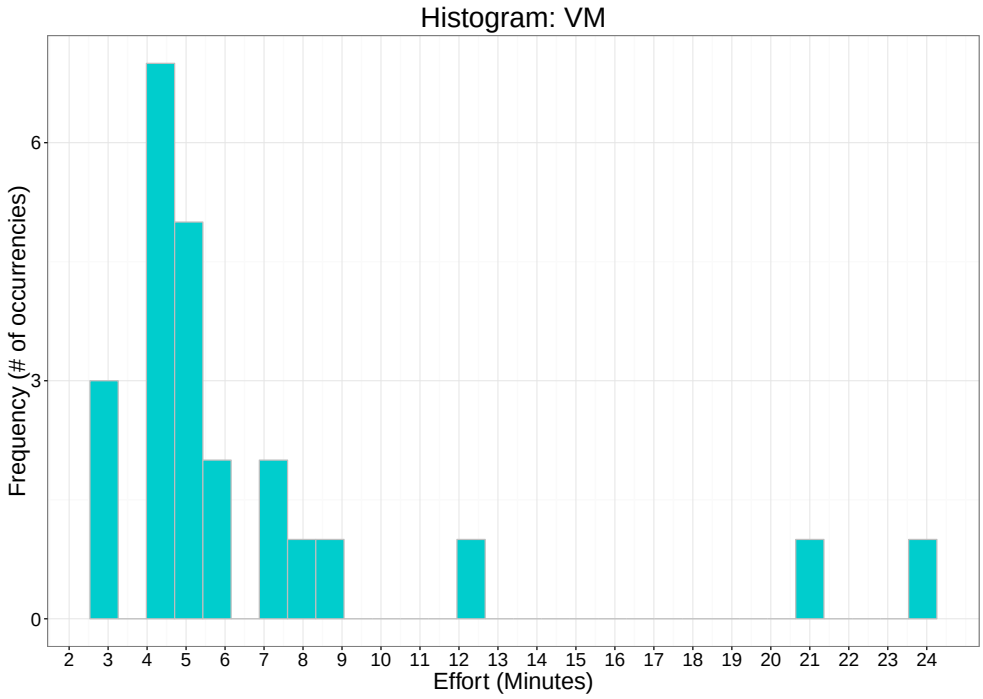


Figure 5.9: VM service histogram. Most occurrences took less than 9 minutes (83%).

Table 5.12: Summary of descriptive statistics for both cloud services.

Service	Min	1st Quartile	Median	Mean	3rd Quartile	Max
Container-based	1.0	2.0	3.0	3.63	4.25	13.0
VM-based	3.0	4.0	5.0	6.75	7.0	24.0
(%) Variance	200	100	66	87	64	84

5.3.2.3 Hypothesis Testing

The *null hypothesis* states that the median time to deploy an application into a cloud service is the same regardless of the cloud service adopted. From the previous section, we can observe that medians of time vary according to the service. In particular, the VM service took 66% more time to deploy the application than the container service. This difference was confirmed by a significant difference at $\alpha = 0.05$. In addition, both the effect size and statistical power calculated for this experiment can be considered large according to values observed in SE studies, and standard conventions [120]. Therefore, for this experiment, taking into account the experimental design set up, we **reject** the *null hypothesis* and **accept** the *alternative hypothesis* that states that the median time to deploy an application by using a VM service is longer than a container service.

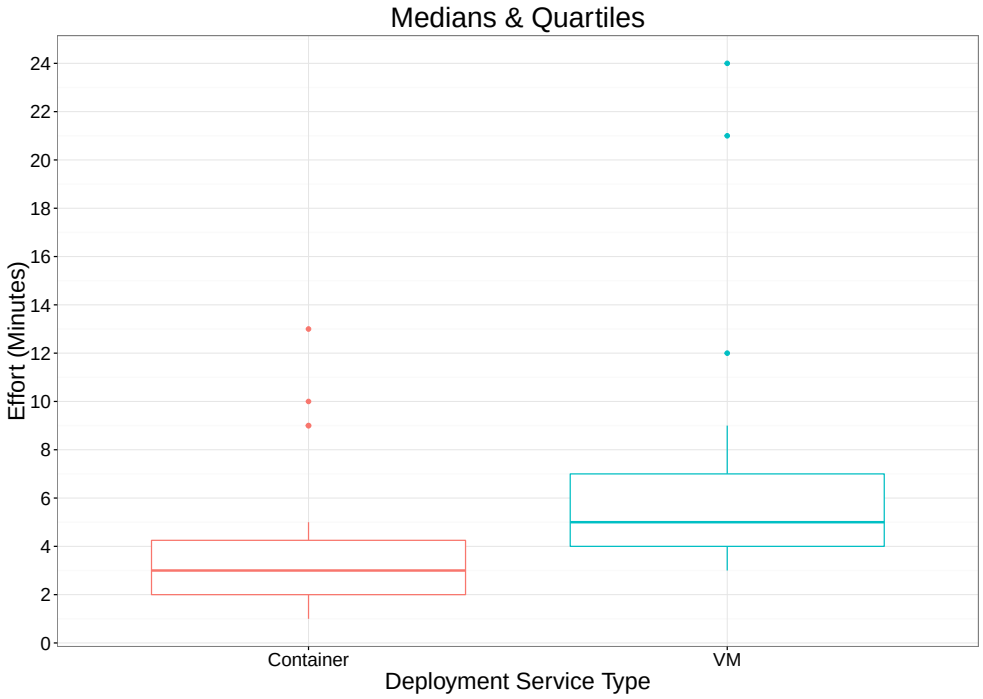


Figure 5.10: Comparison of medians between two treatments. VM-based deployments are 66% longer than container-based ($p = 0.0001$, $\gamma = 0.50$, Pwr $= 0.59$).

5.4 Discussion & Threats to the Validity of Empirical Investigations

5.4.1 Discussion

5.4.1.1 Research Question

This chapter investigates whether cloud platform and service impact cloud portability. To do so, two experiments were carried out (Sections 5.2 and 5.3). In both experiments, the hypothesis testing supported the rejection of the null hypothesis and acceptance of the alternative hypothesis. This means that, for those tasks evaluated in these two experiments, the effort differs between treatments. This difference of effort suggests that *cloud platform and service impact on cloud portability.*

5.4.1.2 The Role of Automation

Service configuration and application deployment were the two tasks investigated in this chapter. There are some tools available to support these activities. These tools might be either cloud-specific [83, 199] or general-purpose [170, 222] tools.

The use of tools can dramatically reduce the effort to configure cloud services and deploy applications in cloud services [222]. On the other hand, they often require some time for learning how to use and configure, and they might introduce changes in the software development process (for examples, see [187] and [83]).

In this study, we opted for manual activities because our practice with small companies shows that these companies often configure cloud services and deploy applications manually. However, we believe that benefits and trade-offs of using automated tools to support these tasks as well as their impact on cloud portability should be investigated. In this regard, [134] takes a very first step by carrying out a set of automated migrations and measuring the required steps for each cloud platform.

5.4.2 Threats to Validity

This section discusses the main threats to validity of this study. These threats are classified into internal and external.

5.4.2.1 Internal

Internal threats impact on the cause-effect relationship. These threats might lead to an alternative cause for the effect (known as *confound*) [215]. In this section, we identify four internal threats.

Outliers and different patterns in the data sets We can observe outliers in both experiments carried out. An outlier is an observation that is somehow inconsistent with most of observations in a data set [26]. For example, in Section 5.3.2.1, we assigned outliers, in the scatter plots in Figures 5.6 and 5.7, to the *learning effect* [131]. Some unexpected data patterns were also observed. For instance, in Section 5.2.2.1, we assigned a different pattern observed in Figure 5.1 to the *last trial/task effect* [20].

These possible causes for outliers and different data patterns could be prevented by adopting pre-task [131] and data reduction [256] techniques. We did not perform any *post hoc* data reduction because the size of our sample was small. Removing outliers could decrease the statistical power of both experiments. However, we must highlight

that outliers not always have a negative impact on data analysis [204]. Nevertheless, adopting pre-tasks and defining clear procedures for data reduction should be adopted in future replications of this study.

Different background of participants Participants from two different universities took part in this study in an *unbalanced between-subject* design (Section 5.1.1.1). Although this is a valid design for experiments in SE [121] and the statistical analysis in Section 5.1.2 shows that participants did not differ regarding the group of questions used to characterise them, we recognise that it might raise questions about the validity of findings. For example, a student from a top ranked university could perform the tasks more efficiently than another student from a non-top ranked university. Whereas in future replications students in the CG and TG from the same university should be considered for the matter of comparison, for this study we believe that this fact is irrelevant. Tasks performed by participants were very simple and required no previous knowledge or special skills to be performed, other than those taught in training sessions.

The response time of cloud platform The response time of cloud platforms might have impacted results of the experiment presented in Section 5.2.2. For two trials, AWS required more effort than Azure. In contrast, for other two trials, Azure platform required more effort than AWS. It is essential to underline that the number of queues to be created was randomly assigned to trials (Section 5.2.1). During the experiment, participants reported delays in the Azure platform when the number of queues were higher than 50. No participant reported a similar issue for the AWS platform. However, it is important to highlight that the effect size was large even for the first two trials. Therefore, even if the last two trials are disregarded, the acceptance of the alternative hypothesis remains.

Fatigue The tasks performed in both experiments were repetitive. Therefore, a possible threat is the fatigue resulting from constant repetition. This can impact on the time to perform the tasks in different ways. For instance, a bored participant could copy and paste content to finish the task quicker. To reduce this threat, experimental tasks were performed in different periods of the day. A training session was performed between the execution of each experimental task. In addition, we restrained ourselves to a few trials to keep the experiment execution within about an hour.

5.4.2.2 External

External threats restrict the result generalisation beyond the scope of the study [256]. In this section, we identify three external threats.

The use of students as participants A major threat of using students in this study, rather than developers with experience in cloud platforms and services, is that students depended exclusively on the training session to learn how to perform the tasks. In addition, their lack of expertise could have contributed to increasing the average time to undertake the tasks. Although it reduces our ability to generalise our findings, it is acceptable for this study as our aim is to obtain early evidence of the phenomenon under study [49].

Cloud service used for evaluating cloud platforms The experiment in Section 5.2 evaluates the impact of cloud platform on cloud portability by using a cloud service that is less common than VM or storage services - the most prominent services in the cloud literature [44]. This might lead to the misleading assumption that the result of this experiment is not valid because top used cloud services should be considered.

It is important to point out that the experiment in Section 5.2 evaluated the cloud platform, not the cloud service. The cloud service was used as an *object* as cloud platforms are not evaluated directly [142]. Both AWS and Azure are leader cloud platforms that have driven the innovation in the cloud market [140]. However, we acknowledge that using only one cloud service to evaluate two cloud platforms restricts any generalisation beyond this type of cloud service. Different cloud services should be used as *objects* in future replications to test whether generalisations are possible.

Sample size In this study, 15 participants took part in two experiments. Compared with the subset of SE experiments that accounts only for undergraduate students, this is less than the average ($\tilde{x} = 43$), but greater than the minimum (n = 10) [215]. The small sample reduces the statistical power of inferential tests, and limits its generalisation. Although the bootstrap technique was used to mitigate the effect of a small sample, future replications of this experiment should consider more participants.

5.5 Building Prediction Models - Cloud Platform

This section details the exploratory process of building regression models for predicting the effort of increasing cloud application portability when configuring cloud services in different platforms. By using concepts and techniques explained in Appendix C and results from the experiment reported in Section 5.2, we built a set of simple and multiple linear regression models using OLS and, alternatively, robust regression methods. Leave-one-out cross-validation (LOO-CV) was used to evaluate prediction models. The prediction model accuracy was assessed by using the MMRE summary measure. Additionally, we compared the accuracy of our models with maintainability prediction models reported in Section C.4. The rationale for techniques used in this section is detailed in Appendix C.

5.5.1 Simple OLS Linear Regression Model

The experiment reported in Section 5.2 shows that cloud platform can be used as an indicator of cloud application portability when configuring cloud services in different platforms. Therefore, we built a simple linear regression model by using the cloud platform as a single predictor. Figure 5.11 shows a jitter plot with all 52 observations and a model line representing the prediction model.

Table 5.13 summarises the accuracy results obtained with LOO-CV, and the goodness of fit for this simple model. According to the classification presented in Section C.4 for maintainability prediction models, the simple model #1 (MMRE = 0.415) achieved a *good* accuracy ($0.37 < \text{MMRE} \leq 0.49$). Yet according to the accuracy classification, the Pred(0.37) shows that 63.5% of predictions using this model could be classified as *excellent*. The goodness of fit, as measured by the R^2, is very low when compared to prediction models in previous chapters ($R^2 = 0.003$), however.

5.5.1.1 Outlier analysis

As outliers can influence the model, we tried to identify possible outliers and quantify their impact on the model. The plot in Figure 5.11 suggests four outlier candidates for Azure platform with effort >20 min (Observations 40, 44, 48 and 52). The plot also suggests another six outlier candidates (Observations 1, 5, 9, 17, 25 and 35) for AWS platform (Effort <10 min).

In addition to the jitter plot, we also used the Cook's distance to identify outlier candidates. As Figure 5.12 shows, only observation 48 was above the cuttoff for Cook's

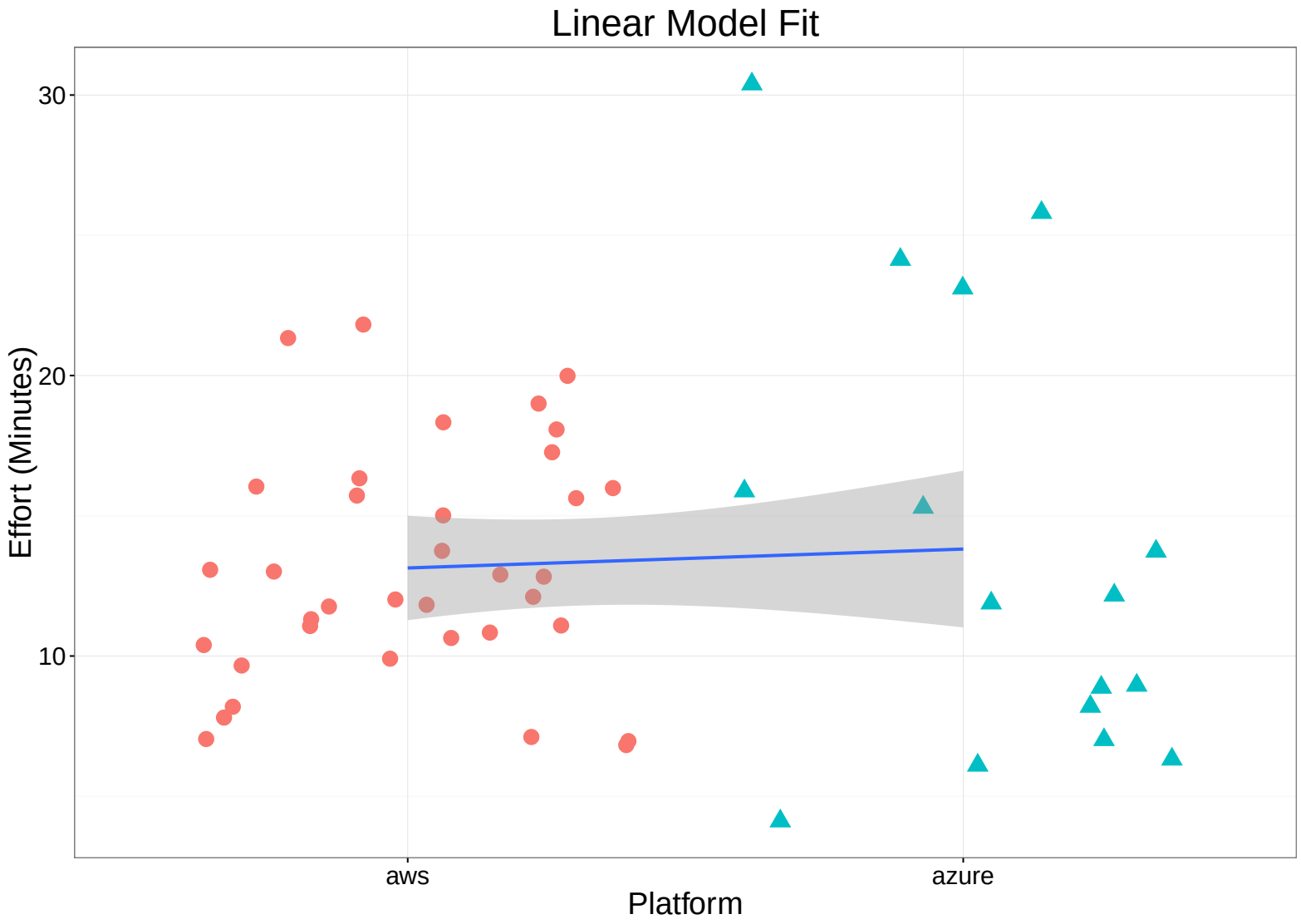


Figure 5.11: Simple effort prediction model line using the cloud platform as a single predictor.

Table 5.13: Accuracy and goodness of fit for effort prediction models built.

#	Prediction Model (Variables)	Accuracy Measures					Goodness of fit	
		MMRE	MAR	Pred(0.25)	Pred(0.30)	Pred(0.37)	R^2	RSE
1	Simple model	0.415	4.481	0.500	0.538	0.635	0.003	5.562
2	Simple robust model	0.379	4.493	0.462	0.538	0.654	0.016	5.274
3	Multiple model (Platform + config load)	0.284	3.381	0.500	0.596	0.750	0.500	3.976
4	Multiple robust model (Platform + config load)	0.278	3.397	0.500	0.596	0.769	0.399	4.020

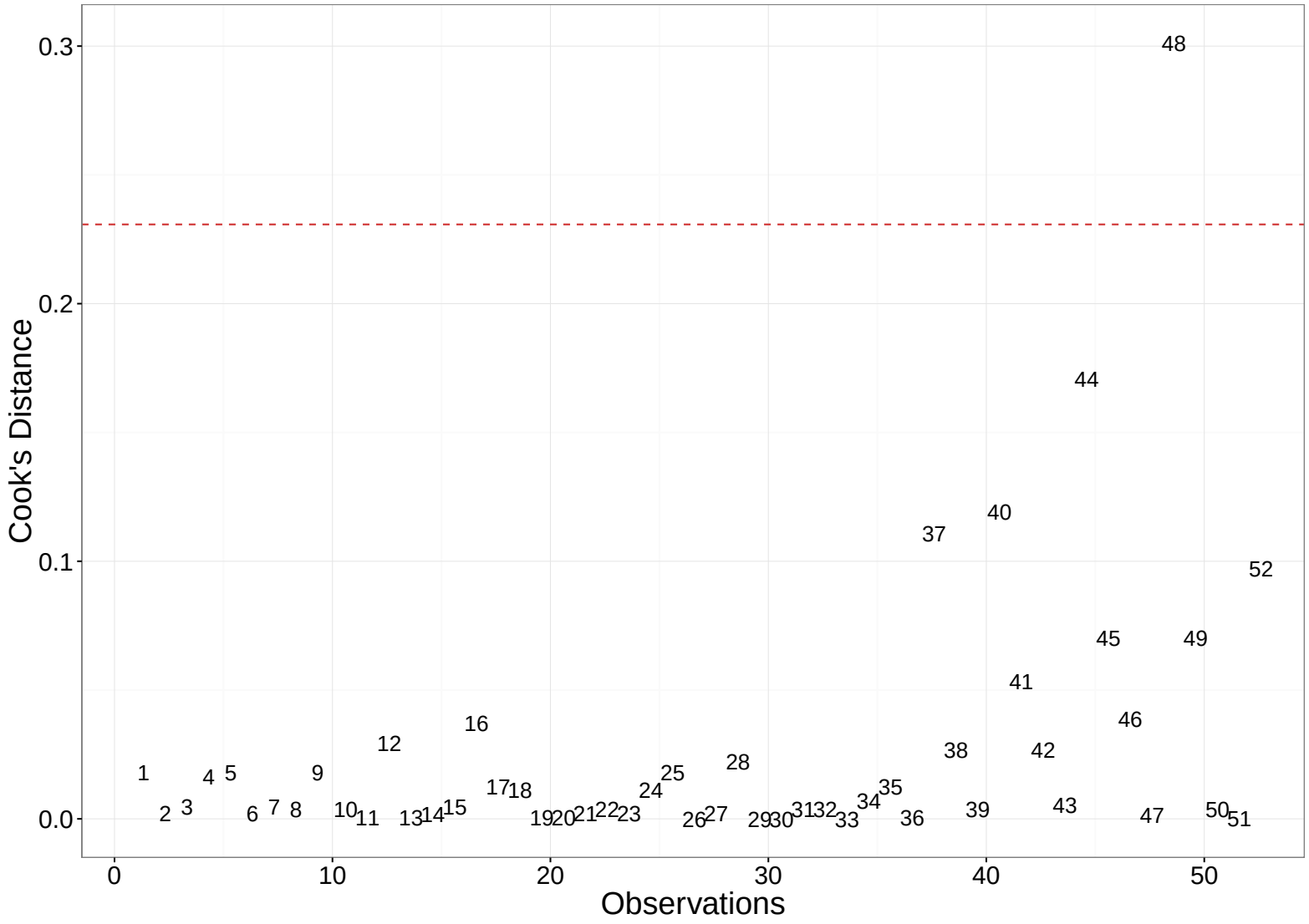


Figure 5.12: Cook's distance plot of data points. Only one observation is acknowledged as an outlier though several others are possible candidates.

distance (0.23) though several other observations are possible outlier candidates.

We analysed 11 observations (1, 5, 9, 17, 25, 35, 37, 40, 44, 48, 52) to check whether they were influential cases for the model. Ten participants were responsible for these 11 observations. We identified that four observations with extreme high effort values for Azure platform (Observations 40, 44, 48 and 52) came from the queue size = 100. Therefore, they could not be considered as outliers.

Five out of six extreme low effort values for AWS platform were result of the queue size = 25 and they did not differ significantly from other observations with the same queue size. Therefore, these five observations (1, 5, 9, 17 and 25) could not be characterised as outliers. Finally, although observation 35 differs from other observations for queue size = 75, we noted that removing this observation brought no improvement to the model (MMRE = 0.414; R^2 = 0.002). Therefore, the original data set was kept.

5.5.1.2 Simple Model Diagnostics

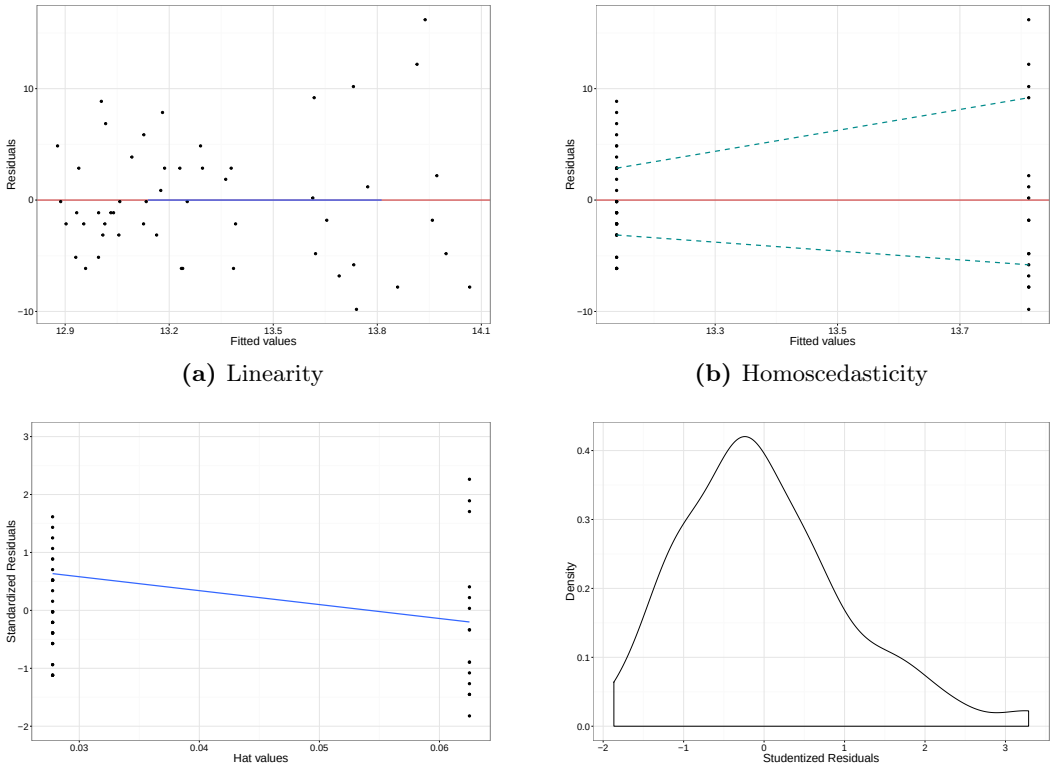
As explained in Section C.5, meeting some underlying assumptions is a pre-condition to accurately generalise a model beyond its data set. We tested the extent to which model #1 meets seven underlying assumptions for simple linear regression models based on the OLS method. Figure 5.13 shows four plots to visually support the diagnostics.

Sample size. No observation was removed from the original sample. Therefore, the simple model was based on the full sample that consists of 52 observations. As model #1 is based on a single predictor, the sample-predictor rate is within the acceptable range (i.e., ≥ 10).

Linearity. Figure 5.13 (a) shows the jitter plot for residuals and fitted values of model #1. The plot suggests two clusters grouping data points - one for fitted values below 13.5 and another for values above it. In fact, the Rainbow test confirmed that the linearity assumption is not met (Rain = 3.044, p = 0.003).

Independent residual terms (errors). We tested the lack of autocorrelation for residual terms by using Durbin-Watson test (DW = 2.424, p = 0.921). The DW statistics is close to 2, meaning that errors are independent. The p-value is not significant for $\alpha = 0.05$, confirming that model #1 meets this assumption.

Homoscedasticity. To meet this assumption, the variance of residual terms should be homogeneous. Figure 5.13 (b) helps to identify the homogeneity of the variance. The



(a) Linearity **(b)** Homoscedasticity

(c) Low Leverage **(d)** Normality

Figure 5.13: Diagnostics plots for effort prediction model #1

two dashed lines suggest a funnel pattern, which could mean a violation of homoscedasticity (i.e., the presence of heteroscedasticity). However, we should consider that our single variable is dichotomous, which might contribute to this perspective. Furthermore, we can observe a homogeneous distribution of data points around the middle (red line). Finally, the Goldfeld-Quandt test confirmed that model #2 meets this assumption for $\alpha = 0.05$ (GQ = 2.223, p = 0.027).

Low leverage. The plot in Figure 5.13 (c) suggests no major leverage issues as data points are centred around zero (y-axis). The average leverage for this model is 1.019, therefore the cutoff is three times this value (3.057). As we can observe in Figure 5.13 (c), no observation is above this cutoff. Thus, the model meets this assumption.

No perfect multicollinearity. As Model #1 has only one predictor, there is no risk of multicollinearity.

Normality. The density plot in Figure 5.13 (d) shows a slightly positive skew, which suggests deviation from normality. Indeed, as the Shapiro-Wilk test confirmed, model #1 violated the normality assumption for $\alpha = 0.1$ (W = 0.960, p = 0.085).

5.5.1.3 Alternative Simple Robust Regression Model

As explained in Section C.5, robust regression is a method less sensitive to outliers and make no assumption regarding the distribution type. This section uses this method to test whether it can improve the prediction accuracy of our regression model. Additionally, for the simple model #1, the robust regression is an alternative to overcome the lack of normality

Table 5.13 shows the accuracy and goodness of fit for the simple robust model #2. Compared to model #1, the robust regression improved the model in -8.6% (MMRE = 0.379) though all Pred measures decreased. Regarding the goodness of fit, we can observe a slight improvement of both measures.

5.5.2 Multiple OLS Linear Regression Model

A multiple linear regression model is similar to a simple model, but it includes more than one predictor. In addition to the variable that defines the cloud platform used, our data set also has another variable to define the number of queues that was created during the service configuration in each cloud platform (i.e., the configuration load). Unlike previous chapters, we used the forced entry method to build a multiple linear model and investigate whether this variable could improve the prediction accuracy of our simple model.

We applied the forced entry method to identify the impact of the configuration load (i.e., the additional variable) in our model. Next, we analysed the existence of outliers. Then, we checked whether the multiple linear model meets underlying assumptions for multiple linear regression models based on OLS. Finally, a multiple robust regression model was built to evaluate the impact of this method in the prediction accuracy.

5.5.2.1 Forced Entry Method

As we had only one additional variable, this method was used rather than the forward stepwise method. As result, model #3 consists of two variables. As Table 5.13 shows, the second variable significantly improved the model (MMRE = 0.284) when compared

to the simple OLS model (-31.5%) and also the simple robust model (-25%). The goodness of fit also increased considerably ($R^2 = 0.5$; RSE = 3.976).

5.5.2.2 Outlier analysis

This section tries to identify possible outliers and quantify their impact on the multiple model #3. As there are multiple predictors in the model, it is no longer possible to infer outliers just by inspecting the jitter plot. Therefore, we used the Cook's distance to identify possible outliers.

Using default values for the cutoff of Cook's distance, the plot in Figure 5.14 highlights the same observation presented in the simple model (48). As discussed in Section 5.5.1.1, this is not an outlier. Therefore, no change was made to the original data set.

5.5.2.3 Multiple Model Diagnostics

As for the simple model, we carried out a set of tests to identify the extent to which the best multiple model #3 meets seven underlying assumptions for multiple linear regression models based on the OLS method. Figure 5.15 shows four plots to visually support the diagnostics.

Sample size. Unlike the simple model, the multiple model #3 consists of two predictors. As no observation was removed from the original sample, the sample-predictor rate is 26, which is still greater than the minimum cutoff of 10. Therefore, model #3 meets this assumption.

Linearity. Figure 5.15 (a) shows the jitter plot for residuals and fitted values of multiple model #3. The blue line suggests that data points follow a pattern. To test the assumption that the model has no linearity issues, we applied the Rainbow test that confirmed that the model meets the linearity assumption (Rain = 1.618, p = 0.123).

Independent residual terms (errors). We tested the lack of autocorrelation for residual terms by using Durbin-Watson test (DW = 2.057, p = 0.532). Like in the simple model, the DW statistics is close to 2, meaning that errors are independent. The p-value is not significant for $\alpha = 0.05$, confirming that model #3 meets this assumption.

Homoscedasticity. To meet this assumption, the variance of residual terms should be homogeneous. Like for the simple model (Figure 5.13 (b)) the two dashed lines in Figure

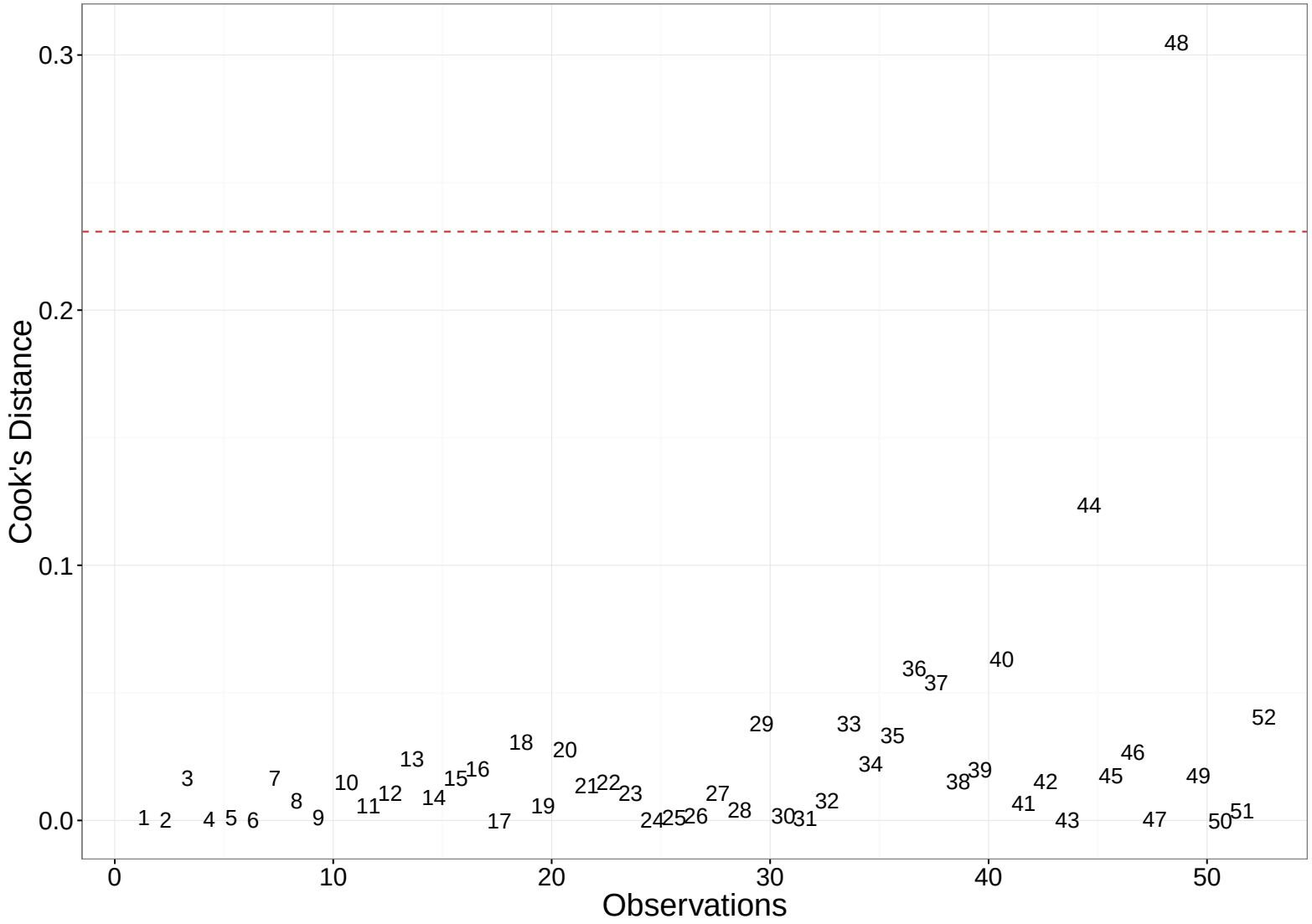


Figure 5.14: Cook's distance plot of data points for the multiple effort prediction model #3.

5.15 (b) suggest a funnel pattern, which could indicate the presence of heteroscedasticity. As the Goldfeld-Quandt test confirms (GQ = 1.857; p = 0.072), the homoscedasticity assumption is not met for $\alpha = 0.05$.

Low leverage. The average leverage for this model is 2.019, therefore the cutoff is three times this value (6.057). As we can observe in Figure 5.13 (c), no observation lies above this cutoff. In fact, the largest Hat value is 0.10. Thus, we considered that the model #3 meets this assumption.

No perfect multicollinearity. Unlike simple model #1, multiple model #3 might violate the multicollinearity assumption as it has multiple predictors. To test this assumption, we checked the variance inflation factor (VIF) that indicates whether a predictor has strong relationship with other predictors. The VIF calculated for both predictors was equals to 1. Consequently, the tolerance (1/VIF) was also equals to 1. As the values for the VIF are lower than 10 and the tolerance is greater than 0.2, we considered that the model #3 meets this assumption.

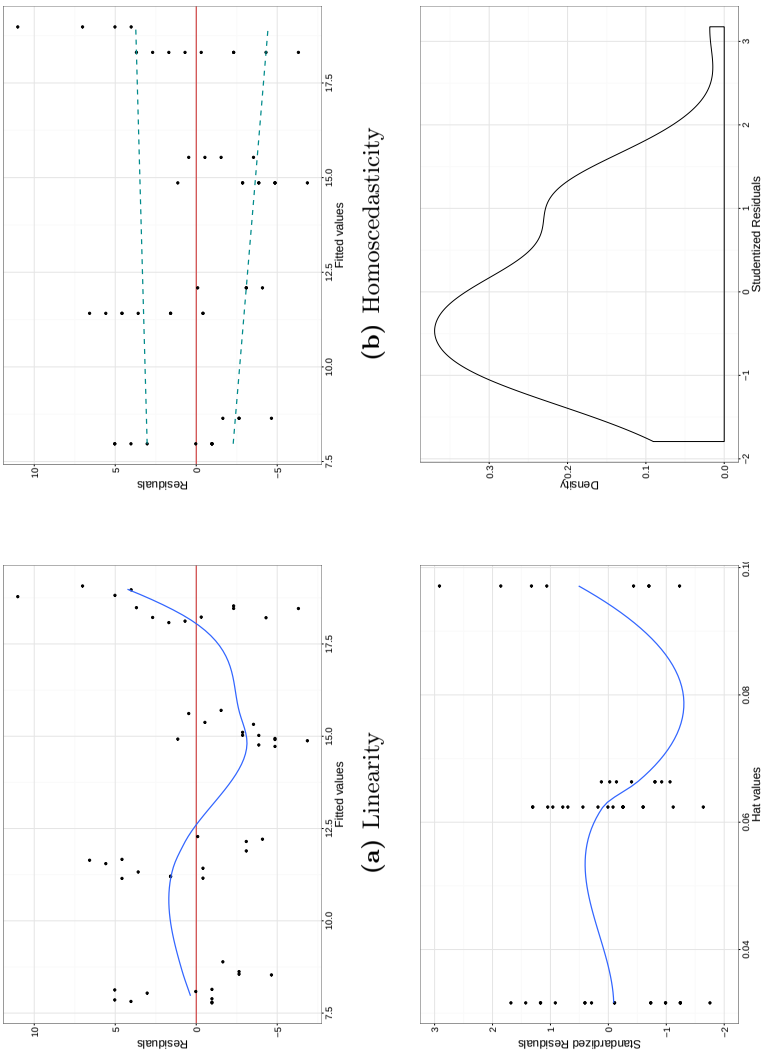
Normality. As for the simple model #1, the density plot in Figure 5.15 (d) suggests a slightly positive skew, which might means deviation from normality. However, as confirmed by the Shapiro-Wilk test, model #3 meets the normality assumption for $\alpha = 0.1$ (W = 0.966, p = 0.150).

5.5.2.4 Alternative Multiple Robust Regression Model

As Table 5.13 shows, the robust method improved the multiple model in -2.1% (MMRE = 0.278). This is also the best accuracy achieved amongst all four prediction models built. Additionally, the Pred(0.37) shows that 76.9% of predictions made by this model achieved an *excellent* accuracy, according to the classification presented in Section C.4.

5.5.3 Discussion

From 4 prediction models built in Sections 5.5.1 and 5.5.2, the multiple robust model #4 achieved the best accuracy as measured by the MMRE summary measure (MMRE = 0.278). This was the best accuracy obtained so far for all prediction models reported in this document. Taking into consideration the accuracy classification proposed in Section C.4, this prediction model achieved an *excellent* prediction (MMRE $\leq$ 0.368). Moreover, only one quarter of predictions made by this model could not be classified



(a) Linearity (b) Homoscedasticity (c) Low Leverage (d) Normality

Figure 5.15: Diagnostics plots for effort prediction model #3

as *excellent* (Pred(0.37) = 0.769). The goodness of fit ($R^2 = 0.399$) is better than that obtained by the best multiple robust model built in Section 4.3, but it is still lower than that achieved by the best model in Section 3.4. Finally, as the robust method is less sensitive to the heteroscedasticity presented by the multiple OLS model #3, we conclude that this model can be used to predict the effort of increasing cloud application portability within the context defined in Section 5.2

5.6 Building Prediction Models - Cloud Service

This section details the exploratory process of building regression models for predicting the effort of increasing cloud application portability when deploying cloud applications in a cloud service. Unlike prediction models built in previous chapters, our data set consisted of one single variable - the cloud service used. Therefore, we were able to build only simple models. By using concepts and techniques explained in Appendix C and results from the experiment reported in Section 5.3, we built a set of simple linear regression models using OLS and, alternatively, robust regression methods. Leave-one-out cross-validation (LOO-CV) was used to evaluate prediction models. The prediction model accuracy was assessed by using the MMRE summary measure. Additionally, we compared the accuracy of our models with maintainability prediction models reported in Section C.4. The rationale for techniques used in this section is detailed in Appendix C.

5.6.1 Simple OLS Linear Regression Model

The experiment reported in Section 5.3 shows that the cloud service can be used as an indicator of cloud application portability when deploying cloud applications during a cloud migration. Therefore, we built a simple linear regression model by using the deployment service (i.e., container or VM) as a single predictor.

Figure 5.16 shows a plot with all 60 observations and a model line representing the prediction model. As the plot suggests, the model line is influenced by outlier candidates for the VM service (Effort >10 min), which might have impacted the model fit.

Table 5.14 summarises the accuracy results obtained with LOO-CV, and the goodness of fit for this simple model. According to the classification presented in Section C.4 for maintainability prediction models, the simple model #1 (MMRE = 0.755) achieved a *fair* accuracy ($0.49 < \text{MMRE} \leq 0.88$). Yet according to the accuracy classification, the Pred(0.37) shows that half of predictions using this model could be classified as

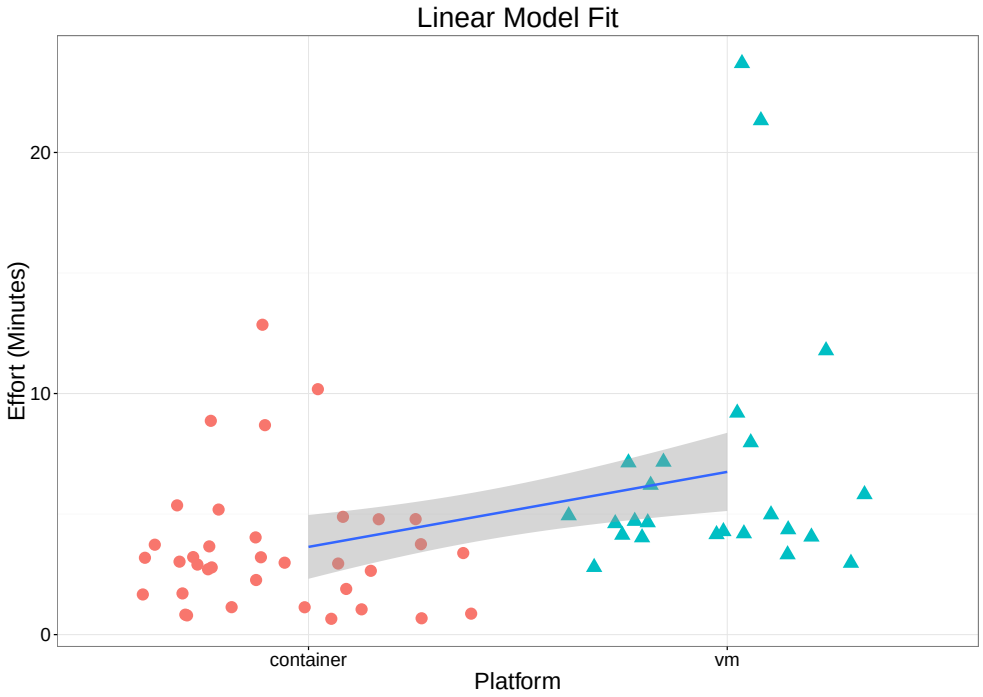


Figure 5.16: Simple effort prediction model line using the cloud service as a single predictor.

excellent (Pred(0.37) = 0.517). The goodness of fit, as measured by the R^2, is relatively low ($R^2 = 0.132$), however.

5.6.1.1 Outlier analysis

As outliers can influence the model, we tried to identify possible outliers and quantify their impact on the model. The plot in Figure 5.16 suggests three outlier candidates with Effort >10 min for the VM service (Observations 53, 55 and 57). Additionally, observations 1 and 29 presented an extreme high effort for Container service (Effort $\geq$ 10 min). In addition to the jitter plot, we also used the Cook's distance to identify outlier candidates. As Figure 5.17 shows, observations 53 and 57 were above the cuttoff for Cook's distance (0.2). We can also observe that the Cook's distance statistics for data points 1, 29 and 55 differ from most observations.

We analysed these six observations identified in the jitter and Cook's distance plot to check whether they were influential cases for the model. Four participants were responsible for the six outlier candidates. We identified that most of these observations were result of the first attempt to deploy the application. Therefore, they might have

Table 5.14: Accuracy and goodness of fit for effort prediction models built.

#	Prediction Model (Variables)	Accuracy Measures					Goodness of fit	
		MMRE	MAR	Pred(0.25)	Pred(0.30)	Pred(0.37)	R^2	RSE
1	Simple model	0.755	2.607	0.333	0.433	0.517	0.132	3.967
2	Simple model w/out outliers	0.561	1.398	0.400	0.564	0.564	0.189	1.868
3	Simple robust model	0.527	2.263	0.300	0.417	0.517	0.856	1.813

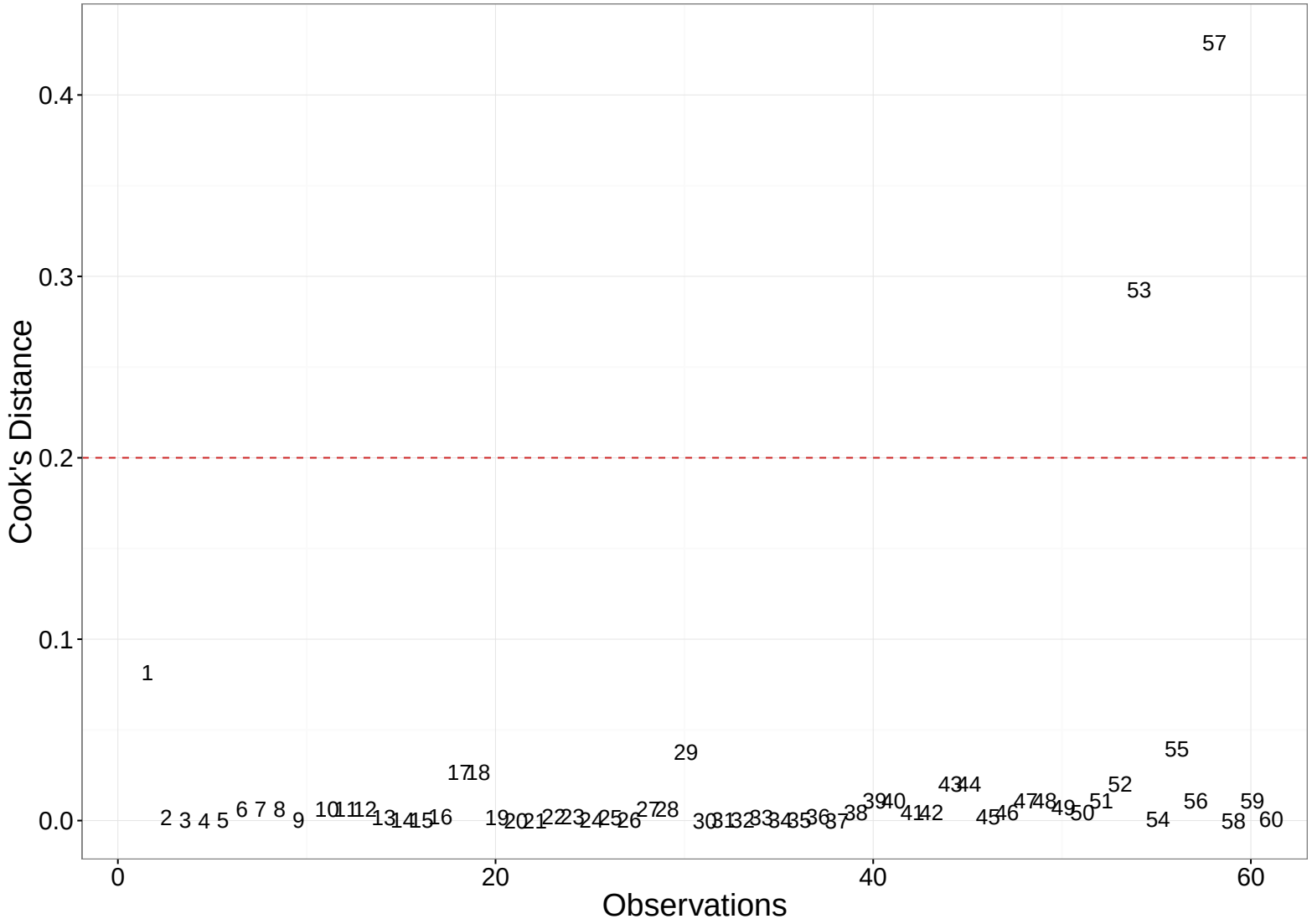


Figure 5.17: Cook's distance plot of data points.

been impacted by the *learning effect* (Section 5.3.2.1).

As result, we concluded that these six observations were real outliers, and we decided to remove them from our data set. Table 5.14 shows the resulting model #2. The accuracy of the model #2 (MMRE = 0.561) improved -25% when compared to the previous model. We can also observe some improvement in all other accuracy and goodness of fit measures.

5.6.1.2 Simple Model Diagnostics

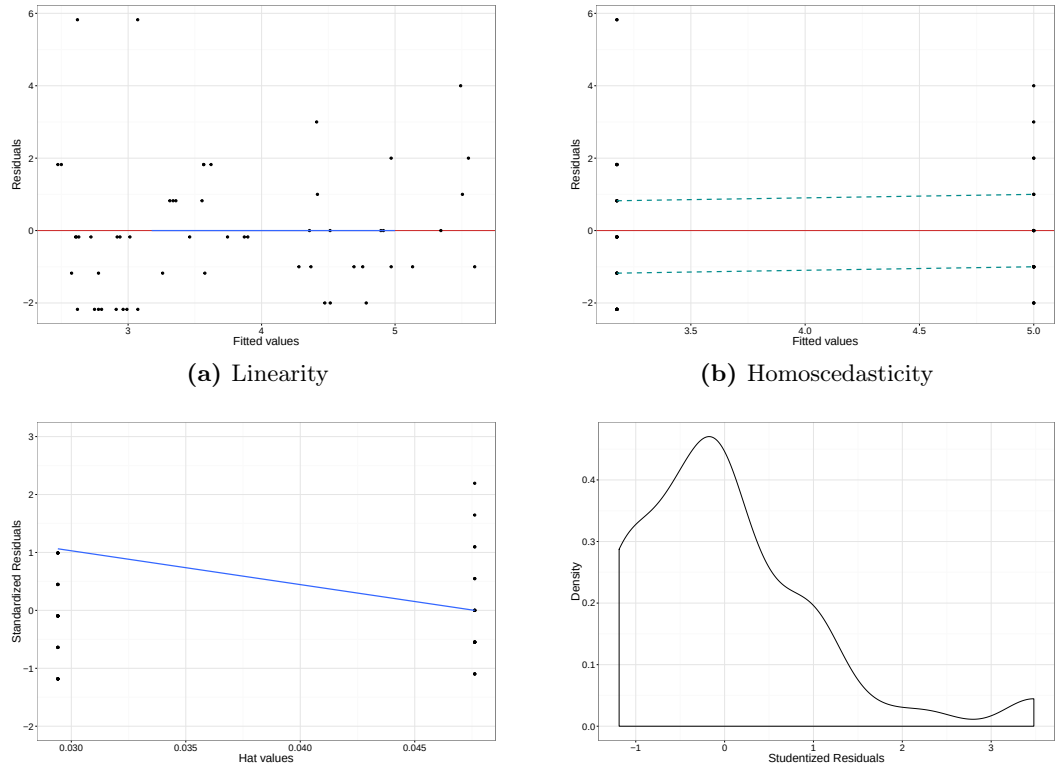
As explained in Section C.5, meeting some underlying assumptions is a pre-condition to accurately generalise a model beyond its data set. We tested the extent to which model #2 meets seven underlying assumptions for simple linear regression models based on the OLS method. Figure 5.18 shows four plots to visually support the diagnostics.

Sample size. In total, six observations were removed from the original sample, resulting in a reduced sample with 54 observations. As model #2 is based on a single predictor, the sample-predictor rate is within the acceptable range (i.e., ≥ 10).

Linearity. Figure 5.18 (a) shows the jitter plot for residuals and fitted values of model #2. Data points are scattered across the plot around the red line (middle), suggesting a homogeneous distribution. The blue line is straight, supporting the assumption that residuals follow no pattern. The Rainbow test confirmed that the linearity assumption was met (Rain = 0.758, p = 0.761).

Independent residual terms (errors). We tested the lack of autocorrelation for residual terms by using Durbin-Watson test (DW = 1.482, p = 0.017). The DW statistics is close to 1, meaning that errors are not independent. The p-value is significant for $\alpha = 0.05$, confirming that model #2 does not meet this assumption.

Homoscedasticity. To meet this assumption, the variance of residual terms should be homogeneous. The two dashed lines in Figure 5.18 (b) suggest no pattern. Additionally, we can observe a homogeneous distribution of data points around the middle (red line). However, the Goldfeld-Quandt test shows that model #2 does not meet this assumption for $\alpha = 0.05$ (GQ = 0.455, p = 0.974).



(a) Linearity **(b)** Homoscedasticity

(c) Low Leverage **(d)** Normality

Figure 5.18: Diagnostics plots for effort prediction model #2

Low leverage. The plot in Figure 5.18 (c) suggests no major leverage issues as data points are centred around zero (y-axis). The average leverage for this model is 1.018, therefore the cutoff is three times this value (3.055). As we can observe in Figure 5.13 (c), no observation is above this cutoff. Thus, we considered that the model #2 meets this assumption.

No perfect multicollinearity. As Model #2 has only one predictor, there is no risk of multicollinearity.

Normality. The density plot in Figure 5.18 (d) shows a positive skew, which suggests deviation from normality. Indeed, as the Shapiro-Wilk test confirmed, model #2 violated the normality assumption ($W = 0.881$, $p < 0.001$).

5.6.1.3 Alternative Simple Robust Regression Model

As explained in Section C.5, robust regression is a method less sensitive to outliers and make no assumption regarding the distribution type. This section uses this method to test whether it can increase the prediction accuracy of our regression model. Additionally, for the simple model #2, the robust regression is an alternative to the lack of normality and heteroscedasticity.

Table 5.14 shows the accuracy and goodness of fit for the simple robust model #3. Compared to model #2, the robust regression slightly improved (-6%) the model (MMRE = 0.527) though all Pred measures decreased. Regarding the goodness of fit, the R^2 of model #3 (0.856) increased more than 3 times the value of model #2 (0.189).

5.6.2 Discussion

From 3 prediction models built in Section 5.6.1, the simple robust model #3 achieved the best accuracy as measured by the MMRE summary measure (MMRE = 0.527). This model consists of one predictor – the variable that represents the cloud service used for deployment. The robust method also significantly improved the goodness of fit (R^2 = 0.856) when compared to the OLS models.

According to the classification presented in Section C.4 for maintainability prediction models, model #3 achieved a *fair* accuracy ($0.49 < \text{MMRE} \leq 0.88$), which is within the accuracy range obtained by most models presented in this document. In addition, half of predictions made by model #3 could be classified as *excellent*.

Although model #3 can be used to predict the effort of increasing cloud application portability within the context defined in Section 5.3, its generalisation beyond this data set is limited. The robust model is less sensitive to the lack of normality and heteroscedasticity, but it is not robust to the violation of independent errors. In future studies, machine learning approaches that are robust to this type of violation can be used to enable the model generalisation beyond this data set.

5.7 Summary

This chapter describes our final contributions to the objectives defined in Section 1.2. We empirically investigated environmental aspects of cloud migration, cloud platforms and services, to identify whether they impact cloud application portability. The result of the two experiments in Sections 5.2 and 5.3 confirmed our hypothesis - cloud platform and service do impact cloud application portability within the context defined for the

experiments. In addition, both experiments achieved a large effect size and a high statistical power when taking into consideration SE experiments.

Prediction models built in this chapter achieved *excellent* (MMRE = 0.278) and *fair* (MMRE = 0.527) prediction accuracy when compared to other prediction models reported in the related literature (Section C.4). The model built for predicting the effort of configuring cloud services in different platforms meet assumptions required for regression model generalisation, whereas the model built for predicting the effort of deploying cloud applications in a cloud service does not.

Chapter 6

Conclusion and Future Directions

This research (i) establishes four important factors that impact cloud portability as a means to mitigate risk associated to vendor lock-in, and (ii) uses the experimental results from assessment of these factors to devise prediction models for the effort to increase the cloud portability of legacy web applications. To do so, we combined and exploited sound techniques from empirical software engineering, software quality and software effort prediction. In addition, we identified established strategies for dealing with portability issues in related areas including service-oriented computing and distributed systems, and we extended the applicability of these strategies to the cloud computing domain.

The first part of the hypothesis examined in this thesis (Section 1.2) states, *"design properties and technologies adopted by the application, along with environmental aspects, impact cloud application portability (...)."* Software coupling (design property), security systems (technology), and cloud platform and service (environmental aspects) were found statistically significant and scientifically relevant factors that impact cloud application portability in experiments presented in Sections 3.3, 4.2, 5.2 and 5.3. Within the limitations of our experiments, these findings provide empirical evidence **to accept** the first part of our hypothesis.

The second part of the hypothesis examined in this thesis (Section 1.2) states, *"(...) and [design properties and technologies adopted by the application, along with environmental aspects] can be used as indicators for predicting the effort of improving the cloud application portability."* Using results from experiments in Sections 3.3, 4.2, 5.2 and 5.3, we built 30 regression models (Sections 3.4, 4.3, 5.5 and 5.6) to predict the effort of improving cloud application portability regarding (i) application re-engineering to replace method calls with message queuing, (ii) application re-engineering to modify

configuration and code related to security systems to enable single sign-on by using implementations of OpenID Connect protocol, (iii) cloud configuration to create message queues in public cloud platforms, and (iv) application deployment to deploy cloud applications by using purpose-equivalent cloud services.

Regression models were evaluated using MRE, the *"de facto"* standard measure for prediction models, and leave-one-out cross-validation, a widely used evaluation procedure. To assess the accuracy of our regression models and test the second part of our hypothesis, we carried out a systematic literature review to identify the accuracy of prediction models for maintainability. Maintainability was chosen because we could not find any prediction model for portability in the literature, and because maintainability is a quality attribute closely related to portability.

Comparing the accuracy of our prediction models with the accuracy of prediction models for maintainability, our prediction models produced *excellent* (MMRE ≤ 0.368), *good* ($0.368 <$MMRE ≤ 0.493) and *fair* ($0.493 <$MMRE ≤ 0.875) predictions. Therefore, we **accept** the second part of our hypothesis as we conclude that factors identified in experiments reported in Sections 3.3, 4.2, 5.2 and 5.3 **can be used** as indicators for predicting the effort of improving cloud application portability.

The next sections summarise this thesis.

6.1 Research Objective

As presented in Section 1.2, the research presented in this thesis investigates challenges associated with increasing the portability of legacy web applications in cloud, focusing on the migration analysis phase. Increasing application portability is important to mitigating the risks of vendor lock-in, such as service failures and providers going out of business. The rationale is that a highly portable cloud application (or a subset of its components) can be migrated between clouds with minimal effort whenever it is needed. Therefore, the cloud user is less dependent on the cloud platform or provider. As vendor lock-in is an inhibitor of cloud adoption, mitigating its risks contributes to increase the cloud adoption.

Unlike recently developed applications that use microservices to decompose applications into independent and collaborative components to enable deployment in multiple clouds, service and deployment models, legacy web applications (that were migrated to the cloud) were not developed to be distributed across the Internet. Therefore, migrating these important applications to another cloud requires a major effort in terms of engineering time and financial cost. It is important to increase the cloud portability of

legacy web applications so as to reduce the effort of future migrations not only to other clouds, but also to other environments.

A well-managed application migration starts with an analysis, in which the cloud users explore the migration feasibility. We focus on this analysis as it is regarded as critical in real software migration experiences reported in the literature. In contrast, the cloud portability literature focuses on proposing technical solutions and standards as means for supporting the migration phase. It is important to focus on the analysis to provide instruments that support the decision making process regarding the cloud application migration before starting the process of transferring resources between clouds.

6.2 Contributions of the Research

The two major contributions of this research are the empirical identification of four factors that impact cloud application portability and the rigorous development of a set of regression models for predicting the effort of increasing cloud application portability. These contributions are important because they increase our understanding of cloud portability in general. For instance, our findings make clear the importance of the application in reducing the migration effort. Whereas cloud portability solutions offer a rapid, possibly short-term, solution for mitigating differences in APIs, semantics and technologies adopted by cloud platforms and providers as a means of increasing or enabling cloud portability, our findings enable the development of theories that can support long-term and evidence-based decisions.

Furthermore, the research contributions of this thesis address four gaps in the existing cloud portability research by (i) providing empirical evidence for the importance of different factors of portability; (ii) showing that established techniques from other areas can significantly contribute to cloud portability research; (iii) providing prediction models to analyse the technical and financial feasibility of a software migration in the cloud; and (iv) identifying measurable factors that impact on the migration effort. Together, our contributions support informed decisions regarding the cloud migration of software applications, and mitigate the risks associated with vendor lock-in.

The novelty of our contributions can be explained from the perspective of both cloud portability and software quality. From the perspective of cloud portability, we adopt a different stance by understanding cloud portability as a quality attribute of cloud applications (Novelty 1). This unique stance enabled us to use established techniques from software quality and empirical software engineering to investigate cloud application portability. Second, we carry out rigorous empirical investigations rather than a

solution development (Novelty 2). These empirical investigations significantly increase the understanding of cloud portability. Unlike most existing studies on cloud portability that propose solutions which mitigate differences between APIs, semantics and technologies adopted by cloud providers and platforms, we provide empirical evidence showing that *cloud* portability can be achieved by increasing the *application* portability (Novelty 3). This finding shifts the focus from cloud platforms and providers to cloud applications and enable cloud users to become active agents in mitigating the risks of vendor lock-in.

The novelty of this thesis spans beyond cloud portability and covers also software quality. To the best of our knowledge, this thesis provides the first empirical evidence of factors that impact any type of portability (Novelty 4). Our findings confirm the common assumption that *coupling impacts portability*. Additionally, by analysing our empirical findings and empirical studies in the software quality literature, we observe that similar factors impact both maintainability and portability. This shows the close relationship between maintainability and portability, opening the opportunity for the use of results from maintainability studies in portability research. This include, for instance, studies on software architectural differences [20] and on the use of quality guidelines for object-oriented applications [38].

6.3 Directions for Future Research

Our experiments provide empirical evidence that design properties, technologies and environmental aspects impact cloud application portability. However, the scope of empirical work is necessarily limited [250]. As is typical with empirical studies, our experiments reported in Sections 3.3, 4.2, 5.2 and 5.3 constitute a single piece of evidence rather than a definitive proof [209]. It would be unrealistic to expect that a single experiment could enable wide generalisations [28]. Therefore, independent replications are necessary to increase the confidence in these results and enable their generalisation [159] beyond the scope of this research.

On the other hand, single studies have been used as the first step to build theories [209] and estimation models [7, 94], and to shed light on possible trade-offs of even widely adopted technologies [111] and methods [20, 38]. Although we consider our findings as preliminary, the rigorous methodology we applied, along with the statistically significant and scientifically relevant results we obtained are sufficient to generalise our findings to similar scenarios.

We tested the generalisation of regression models reported in Sections 3.4, 4.3, 5.5

and 5.6 by checking seven assumptions for OLS-based regression models. Taking into account only models with the best accuracy, two models meet all assumptions (reported in Sections 3.4 and 5.5) whereas two other models do not (reported in Sections 4.3 and 5.6). These two models that do not meet all assumptions might yield incorrect results when applied to other data sets.

Increasing the generalisation of our findings, using them to developing supportive tools for cloud migration, and testing different techniques for effort prediction represent important directions of future work. Firstly, our experiments need to be rigorously and independently replicated to increase the confidence in our findings. In particular, finding contradictory results would shed light on the importance of possible threats that our experiments did not cover. These experiments would enable the use of meta-analysis to synthesise results and obtain more consistent conclusions.

Secondly, other factors related to software design properties and technologies, and environmental aspects need to be investigated. This would enable the generalisation of our findings to a wide range of factors. Next, our empirical findings can be used to build tools that support cloud application portability. These tools can be used to decrease application coupling, for instance. Diagnostic tools can give a preview of costs on increasing portability whereas model-based tools can automate code re-engineering. Different machine learning techniques, such as neural networks and fuzzy systems, can be used in an effort to improve our predictions of the effort to increase cloud portability. These techniques have a track record of successful application in predicting the effort associated with other software engineering tasks [8, 10, 75, 231]. Finally, decision support systems could be developed to facilitate the use of prediction models by non-experts.

Two other topics that deserve attention in future research are the current trend towards the use of microservices, and the adoption of cloud standards. Microservices is an architecture whereby business functions are implemented in small code units, packaged and deployed as self-contained software units, including their own independent database [165]. In addition, microservices can be combined with unikernels to isolate the software unit in a portable sandbox. Unikernels are small virtual machines with no division between user and kernel memory spaces [181]. Thus, these software units can be deployed in different cloud platforms and moved from one cloud platform to another with minimal effort [25, 72].

Indeed, based on the results of our experiments in this thesis, we could easily argue that microservices may be the definitive solution for cloud portability because it lowers the coupling between application components and reduces technological dependencies,

e.g., by creating one database for each software unit. Whereas new applications can benefit from using microservices, legacy applications still need re-engineering to adopt this new architecture, however. Because architectural re-engineering costs are often high [220], future research should evaluate current techniques for architectural migration and their feasibility when migrating legacy systems to microservices. Furthermore, future research should also devise techniques and tools to predict and reduce migration costs, including automated migration tools.

Finally, although several standards have been proposed for cloud over the past few years [98], only a few have been considered really useful [78]. TOSCA[1] is a promising standard for cloud application portability [33] that consists of an language for specifying cloud application components and their dependencies with cloud services, and a runtime platform [34]. The runtime platform receives the specification and the packaged application as inputs to deploy the cloud application and to provision cloud resources in the target cloud platform. This is also the goal of some DevOps tools, like puppet[2] and chef[3]. DevOps is a set of practices and tools that aim to integrate development and operation teams [72]. Although TOSCA could be a way to standardise DevOps tools [5], it is essential to investigate the extent with which TOSCA cover functionalities implemented by leader DevOps tools and the effort necessary to adopt TOSCA.

[1] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[2] http://puppet.com
[3] http://www.chef.io

Appendix A

Research Framework and Methodology

This appendix briefly summarises Wazlawick's research framework [250] and the research onion methodology[1]. Whereas the former was used to structure the research presented in this thesis, the latter was used to structure the investigation of factors impacting on cloud application portability.

A.1 Research Framework

Figure A.1 uses the UML class notation to show a conceptual model comprising entities and their relationship as covered by the Wazlawick's framework. A research is concentrated on a particular *Area*, such as cloud computing or software engineering. An *Area* consists of several *Subjects*, such as configuration management and agile methods. For each *Subject* there might be several open *Issues*, such as vendor lock-in.

A *Literature Review* identifies open *Issues* and covers *Related Work* addressing the same *Issue*. The research *Objective* addresses the *Issue* and improves the current state-of-the-art, represented by the *Related Work*. The *Objective* is specialised in *General* and *Specific*. Whereas the former represents the research overall contribution, the latter represents sub-products yielded as part of the general solution, such as a tool.

Since the *Objective* is achieved, some benefits should be expected (*Expected Outcomes*), such as *"increasing the adoption of cloud computing in small companies."* The

[1] `http://onion.derby.ac.uk`

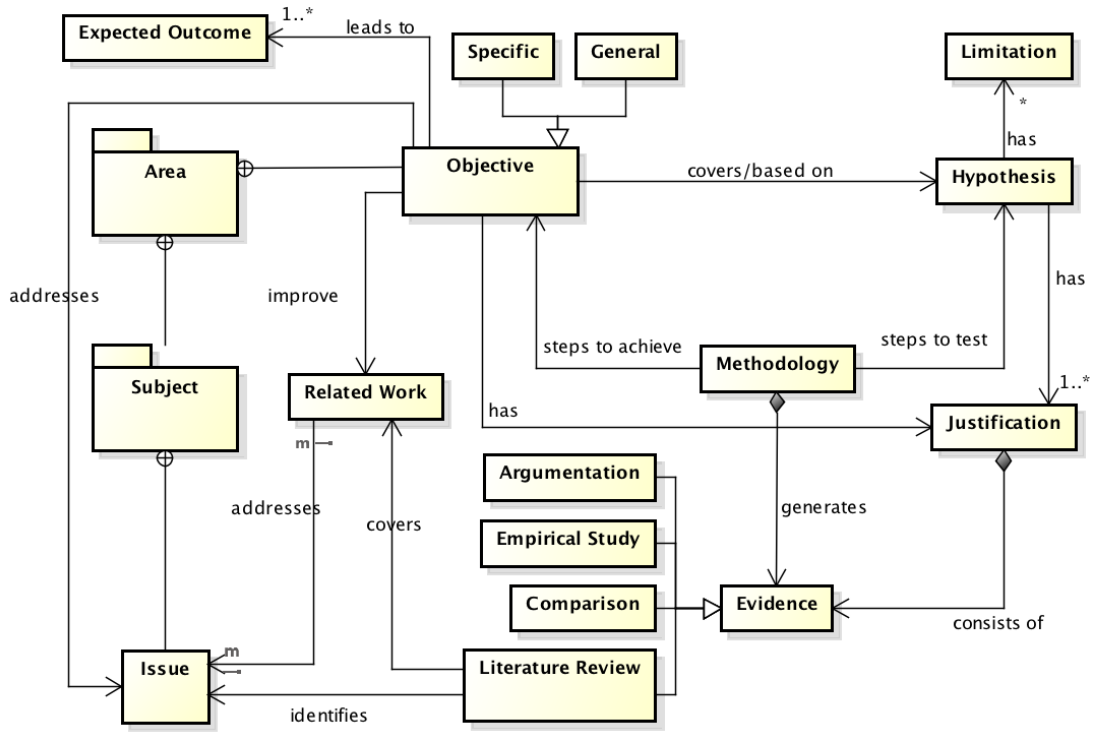


Figure A.1: The conceptual model of Wazlawick's research framework.

Objective has a *Justification* that consists of some *Evidence*, such as *Literature Review* and *Empirical Study*. The *Objective* covers or is based on a *Hypothesis*. For example, for the *Objective "reducing the effort to develop web applications"*, a possible *Hypothesis* could be *"Model-Driven Engineering reduces the effort to develop web applications by automating the generation of artefacts."*

The *Hypothesis* has a set of *Justification* and *Limitation*. A *Limitation* might be related to time, budget, or research interest, for instance. Both *Objective* and *Hypothesis* are achieved through a *Methodology* that is a set of methods addressing different aspects of a research, such as philosophical stance and research techniques. Finally, the identification of each entity in the framework is guided by research questions.

A.2 The Research Onion Methodology

The research onion methodology defines several layers representing different methods and stances for scientific research[1]. The first layer is the philosophical stance. Defining

[1] `http://onion.derby.ac.uk`

a philosophical stance at the beginning of a research is important to explicit what is considered as acceptable evidence [71]. Examples of philosophical stances are positivism, realism and constructivism. The second layer is the approach used to devise a theory, which might be from practice to theory (inductive) or the opposite (deductive) [213].

The third layer refers to the strategy to investigate the hypothesis. Several empirical methods can be used, such as experiment, survey and case study [212]. It is important that the strategy choice is driven by the research objective and other research onion layers [71]. For instance, experiments are the common strategy for positivism using a deductive approach. The fourth layer is the choice regarding the methods to be used. Whereas quantitative methods use numerical data and statistical methods to analyse findings, qualitative methods use raw data (e.g., text and images) and do not rely on precise measures to draw conclusions [212].

The next layer refers to time horizons. They vary according to time of study, which might be short-term (cross-sectional) or long (longitudinal). Whereas in the former groups are evaluated at the same point in time, in the latter different groups are evaluated along the time [126]. Finally, the sixth layer covers techniques and procedures for data collection (e.g., forms) and analysis (e.g., non-parametric statistics). These techniques and procedures vary significantly according to the previous choices [256].

Appendix B

Experimentation in Software Engineering

This appendix briefly summarises fundamental concepts of experimentation in Software Engineering (SE). These concepts are extensively used in Chapters 3, 4 and 5 to address the research objectives presented in Section 1.2. A (controlled) experiment is an empirical method to determine whether a cause-effect relationship exists [71]. A quasi-experiment is an experiment whereby *treatments* are not randomly assigned to *participants* [121]. As Section 2.4.1 explains, experimentation is a common strategy to identify factors that impact a particular QA [7, 21, 38, 40, 112]. Therefore, this is the strategy used in this research to identify factors impacting on cloud application portability.

There are several guidelines for conducting and reporting experiments in SE [118, 128, 131, 214]. This research used the framework proposed by Wohlin et al. [256] as it covers the experiment preparation, execution and result analysis in the depth required by this research. In addition, we use the Goal/Question/Metric (GQM) template of Basili & Rombach [27] to define the goal of each experiment. Figure B.1 illustrates the main elements of the experimentation framework of Wohlin et al..

An experiment investigates a *Hypothesis* that defines a causal relationship between *Dependent* and *Independent* variables. Usually, an independent variable (also called *Factor*) consists of two or more values, called *Treatments*. For example, stating that a Java application (*Treatment* 1) is easier to debug (*Dependent* variable) than a C++ application (*Treatment* 2) is a valid *Hypothesis*.

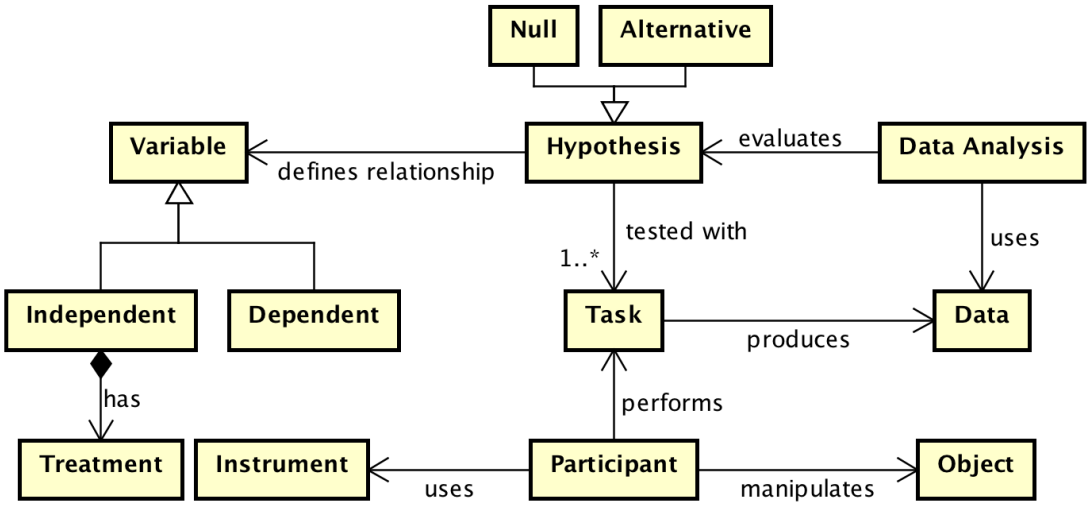


Figure B.1: SE experimentation framework of Wohlin et al. [256].

A *Variable* must be measurable. There are different types of measures, but the most common are *categorical* and *ratio*. In our example, our *Independent* variable is measured in a *categorical* scale since we have two possible values: Java and C++. Although the *Dependent* variable is informally specified (*"easier to debug"*), at some point it must be formally specified. For instance, let's assume that the ease of debugging an application is measured in terms of the amount of time that a developer takes to debug a particular problem. "Amount of time" is a data type that varies from zero to a number, which means that this is a *ratio* variable. Thus, one can objectively compare the time for debugging the two applications.

A *Hypothesis* can be classified into *Null* or *Alternative*. Usually, the *Null Hypothesis* states that there is no difference between *Treatments* whereas the *Alternative Hypothesis* states that there is some difference. For instance, the *Null Hypothesis* could state that the amount of time that a developer takes to debug a Java application is equals to a C++ application whereas the *Alternative Hypothesis* could state that the Java application takes less time. Note that the *Alternative Hypothesis* is what we are trying to show, according to the initial hypothesis defined in the previous paragraph. This is the case in most experiments.

Some *Data Analysis* techniques are used to evaluate the *Null Hypothesis* and define whether it should be rejected or accepted. In the case of the *Null Hypothesis* rejection, the *Alternative Hypothesis* is accepted as truth. The *Data Analysis* uses *Data* produced during the execution of a *Task*. The *Task* is used to test the *Null Hypothesis*. For

instance, identifying the causes of incorrect results in an classification algorithm is an example of debugging task that can be used to test the anecdotal hypothesis introduced in this section. In this case, the *Data* could be the time that each *Participant* took to debug Java and C++ applications.

Participant is the individual that perform a *Task* in an experiment. Along with the *Task*, *Participant* is an important element to increase the realism of SE experiments [214]. For instance, using professional software developers rather than students could increase the realism of the debugging experiment although using students is quite common in SE experiments [215]. *Participant* uses some *Instruments* to perform a particular *Task*, such as tools for software debugging. Training material and data analysis tools can also be considered as *Instruments*. Finally, *Object* is manipulated by *Participant* during the *Task* execution. For instance, the Java and C++ application used in our anecdotal experiment are example of *Objects*.

B.1 Experiment Preparation and Execution

Ideally, an experiment is carried out in four steps [118, 128, 214, 256]. Firstly, a protocol that details the experiment plan and execution is written. Secondly, the experiment is executed and data is collected. Thirdly, data is analysed and conclusions are drawn. Finally, results are reported and made available to the research community.

Preparing an experimental protocol consists of:

- Devising research questions and hypotheses;
- Identifying treatments;
- Identifying, investigating and selecting instruments and objects;
- Defining the strategy for recruiting participants;
- Preparing training and experimental material;
- Carrying out a pilot experiment to test assumptions and experiment material; and,
- Writing down the protocol.

The experiment execution consists of instantiating decisions defined in the experimental protocol. Ko, LaToza & Burnett [131] define eight steps for executing an experiment in SE. They can be summarised into:

- Recruiting, selecting, informing and characterising participants;
- Assigning participants to treatments;
- Training participants;
- Performing experiment tasks; and
- Debriefing, receiving feedback and compensating participants.

B.2 Data Analysis

Descriptive and *inferential* statistics are used for data analysis of experiment results. *Descriptive* statistics describe and present measures of a data set, such as median, minimum and range, whereas *inferential* statistics are used for drawing conclusions based on the comparison of *descriptive* statistics [198, 256]. Additionally, charts such as boxplots, scatterplots and histograms can be used to support data analysis and visualisation [84].

Statistics can be classified into *parametric* or *non-parametric*. Whereas the former is based on some assumptions, such as a normally distributed population, the latter is relaxed regarding these assumptions and less sensitive to *outliers* [256]. *Outliers* are observations that significantly differ from the others [204]. Identifying and understanding *outliers* is important to prevent misleading conclusions [84].

Experiment design and research goals must be taken into consideration to select a set of appropriate *descriptive* and *inferential* statistics and prevent misleading conclusions [198]. Guidelines and frameworks proposed by Andrews et al. [14], Wohlin et al. [256] and Field, Miles & Field [84] are broadly used in this research to select appropriate statistics.

For drawing conclusions using *inferential* statistics, some cutoffs must be set, as exemplified in Table B.1. The *type I error*, also known as *significance level* (α), is the probability of falsely rejecting the null hypothesis [256]. The traditional α value is 0.05 [158], which leads to a *confidence level* of 0.95 when accepting the null hypothesis. Setting up the *type II error* (β) and the *statistical power* require some further analysis.

The *statistical power* is the probability of correctly rejecting the null hypothesis [256]. The *statistical power* is influenced by the α, the sample size and the *effect size* (γ) [70]. Usually, the sample size is well-defined at the beginning of the experiment. However, determining the γ requires investigating similar experiments [120].

Table B.1: Traditional cutoff values for evaluating hypotheses. Table adapted from [70].

		Unknown true state of nature	
		H_0: Null Hypothesis	H_1: Alternative Hypothesis
Statistical Conclusion	Accept H_0	0.95 1 - α: Correct *(Confidence level)*	0.30 β: Type II error
	Reject H_0	0.05 α: Type I error *(Significance level)*	0.70 1 - β: Correct *(Statistical power)*

The γ is important because it measures the presence of a phenomenon in the population and indicates the *scientific* importance of empirical findings [160]. However, similar experiments are not always available. Therefore, an alternative is using common conventions, such as those calculated by Kampenes et al. for SE experiments [120].

Once that the value of γ is defined, one can calculate the *statistical power* by following guidelines provided by Dybå et al. [70]. The last cutoff to define is the *type II error*. The *type II error* (β) is the probability of falsely accepting the null hypothesis [256]. It is related to the *statistical power* ($1-\beta$). Therefore, given the statistical power set up in Table B.1 (0.70), the value for the *type II error* is 0.30.

Additionally, one can calculate the *relative seriousness* of *type I* to *type II error* (β/α) [160]. For example, a *relative seriousness* of 6 (30/0.05) means that the false rejection of the null hypothesis is about 6 times "*more serious than erroneously accepting it.*"

Several statistical software packages can be used for automating the calculation of *descriptive* and *inferential* statistics, such as Matlab and Octave. This research uses the R statistical software[1] for performing data analysis. R is an open source and powerful environment that provides libraries to address analyses required for this research.

B.3 Use of Experimental Results

An experiment constitutes a single piece of evidence rather than a definitive proof [209]. It would be unrealistic to expect that a single experiment could enable wide generalisations [28]. As Dybå, Sjøberg & Cruzes [69] explain, the context of an empirical study (i.e., participants, environment, tools and tasks) is key to define whom, where,

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

when and why the research result aim to. Therefore, independent replications are necessary to increase the confidence in empirical results and enable their generalisation beyond their initial context [159].

On the other hand, as Shull, Singer & Sjøberg [209] advocate, even a single empirical study can be used as the first step to building theories. In addition, single studies have been used to build estimation models [7, 94] and shed some light on possible trade-offs of even widely adopted technologies [111] and methods [20, 38].

B.4 Threats to Validity

Results of an experiment must be valid to the population of interest [256]. Although mitigating threats is important to increase the validity of empirical results, it is not always possible [215]. Nevertheless, identifying and discussing possible threats to validity when reporting an experiment is important to highlight its limitations [128].

Threats can be classified into *internal* or *external*. *Internal* threats impact the cause-effect relationship. These threats might lead to an alternative cause for the effect (*confound* variable) [215], such as systematic differences between *treatments*, different *instruments* used for *tasks*, and *participants* with different knowledge levels. *External* threats restrict the result generalisation [256]. Examples of external threats include the use of students when the target population is professionals and the use of sample sizes that are not representative of all sizes encountered in practice.

Appendix C

Software Effort Prediction

Effort prediction is key to deliver software on schedule and within the budget [234]. Poor predictions can lead to severe consequences, such as business opportunity losses and increased risk of project failure [100, 116]. As Myrtveit, Stensrud & Shepperd [162] and Mair & Shepperd [151] explain, a *prediction approach* is used to create *prediction models* that predict the effort of some software maintenance or development task. *Prediction models* have an *effort factor*, such as time, and one or more *predictors* that influence the *effort factor*, such as developer skill level or project size. The *prediction model accuracy* is measured using *accuracy measures*, and evaluated using a *measuring procedure*, such as cross-validation.

The emphasis in the text above represent elementary concepts of software effort prediction that are presented and discussed in this chapter. Firstly, we present an overview of literature on prediction approaches and show results of empirical work investigating the myth of the best prediction approach (Section C.1). Secondly, we present results of an empirical study on using cross-company data sets for effort prediction, and briefly discuss two common problems in software engineering data sets: outliers and small sample size (Section C.2). Next, we explain the evaluation process of a prediction model (Section C.3). Section C.4 shows preliminary results of a systematic literature review we carried out to identify the accuracy of prediction models for software maintainability. Finally, we introduce the regression approach, which is used in this research for predicting the effort of increasing cloud application portability (Section C.5).

C.1 Prediction Approaches

As Trendowicz, Münch & Jeffery [234] summarise, prediction approaches mainly differ in the input (e.g., data sets or personal experience) they require and the output (i.e., prediction model) they produce. For instance, regression approaches are data-intensive whereas expert judgement does not require any data as input. In a systematic literature review, Jørgensen & Shepperd identify more than 11 prediction approaches in 304 papers published from 1989 to 2004 in 76 high quality journals [117]. The top 3 prediction approaches they identified (and the frequency they appear) are:

1. *Regression* (49%) is a simple and easy approach whereby data is fitted to a pre-specified model consisting of dependent and independent variables [151];

2. *Function point* (22%) is an approach that measures system functionalities [233];

3. *Expert judgement* (15%) is an approach performed by an (set of) expert(s) on a task; most of the prediction process is based on intuition [115].

Other common approaches that Jørgensen & Shepperd identified are analogy (10%), neural networks (7%) and bayesian (2%). The top 3 approaches identified by Jørgensen & Shepperd slightly differs from a survey of most used approaches by practitioners performed by Trendowicz, Münch & Jeffery [234], whereby expert judgement appears at the first position (80%), followed by regression (70%). It is worth to note that the survey was performed from 2005 and 2008 with 10 companies mainly focused on developing applications for the finance sector. Although these studies reveal approaches more investigated by academics or more used by practitioners, Trendowicz, Münch & Jeffery [234] highlight that no prediction approach comply with all requirements expected by companies for prediction approaches, such as flexibility, robustness, comprehensiveness and reliability.

Many studies about prediction approaches aim to identify "the best" approach [116], such as [8, 74, 143, 231, 239, 265]. These studies often present contradictory results [115, 151]. Low study quality, small sample size, and flawed measurement process are pointed out by Myrtveit, Stensrud & Shepperd [162] as reasons for contradictory results. Jørgensen [116] concludes that there is no such "best" prediction approach or model, and Gray & MacDonell [99] explain that modelling capabilities of prediction approaches should be taken into account when selecting a prediction approach. Finally, Mair & Shepperd [151] suggest using more than one approach to reduce risks.

C.2 Data Sets, Outliers and Sample Size

Usually, data sets, used by data-intensive approaches to build prediction models, consist of observations from an experiment [94], historical data from past projects [116], or open available industrial data sets [204, 224]. One question that arises when using data sets from other companies (cross-company) rather than within-company projects is whether the source of the data set impacts the accuracy of prediction models.

To investigate this question, Kitchenham, Mendes & Travassos [130] carried out a systematic literature review whereby they analyse ten studies that empirically compare the prediction accuracy of prediction models using cross-company and within-company data sets. Kitchenham, Mendes & Travassos conclude that their review is inconclusive since (i) only seven out of ten studies were independently undertaken, (ii) three studies show significant difference between models with different data sets whereas four studies show that cross-company models lead to worse predictions than within-models, and (iii) Kitchenham, Mendes & Travassos observed some trends in these studies that might have biased conclusions drawn by experimenters.

Software engineering data sets often suffer from several problems [99], such as outliers and small sample size. Barret & Lewis [26] define outlier as *"a data point that appears to be inconsistent with the rest of the data set."* When detecting an outlier, it is important to investigate whether it really is an inconsistency [84]. Several methods for outlier elimination exist [204], such as boxplots and Cook's distance. Whereas boxplots visually highlight hidden patterns, Cook's distance measures the extent with which an observation (outlier) impacts the residuals of a regression equation [84].

Although the presence of outliers in a data set might affect the prediction accuracy, Seo & Bae [204] found no significant difference of prediction accuracy between data sets with and without outliers in their empirical study. Nevertheless, Seo & Bae strongly recommend removing outliers before carrying out any analysis. Robust methods for effort prediction are less sensitive to outliers [84]. Therefore, they are a feasible alternative on the suspicious of outliers in a data set [99].

The number of data points in a data set (i.e., sample size) is also important for the prediction model accuracy although software engineering data sets quite often are small [99, 130, 162]. Indeed, some data sets broadly used in the literature consists of less than 40 data points [8, 94, 143, 224]. In an empirical study, Stensrud et al. [224] identified a negative correlation between project size and accuracy for the MRE accuracy measure. Therefore, large data sets could lead to better accuracy as MRE is the most used accuracy measure [127].

C.3 Prediction Model Evaluation

As Myrtveit, Stensrud & Shepperd [162] explain, a common design for evaluating a prediction model is using a data set, an accuracy measure and a cross-validation technique. There are several accuracy measures [87], such as Magnitude of Relative Error (MRE) and Balanced Relative Error (BRE). The accuracy measure is computed for each data point in a data set. As data sets consist of several data points, the statistic produced by the accuracy measure is summarised by using mean or median, which leads to the mean of MRE (MMRE) or median of MRE (MdMRE), for example [162].

MRE is the most used accuracy measure in the literature, along with Pred(m) measures [87, 127, 224]. Whereas MRE is defined as the absolute value of the difference between actual and predicted effort, divided by the actual effort, Pred(m) is defined as the percentage of predictions with MRE $\leq m$.

However, several concerns have been raised regarding the use of MRE. For instance, Foss et al. [87] found that MRE is unreliable, tending to underestimate the actual value whereas Stensrud et al. [224] found that MRE is dependent of project size. Whereas reporting more than one accuracy measure is a common alternative to mitigate risks [162], Foss et al. [87] concludes, *"at present, we do not have any universal replacement for MRE."*

Cross-validation is a measuring procedure used to evaluate the prediction model accuracy [162]. The cross-validation procedure splits the data set into n groups that are used to train and test the prediction model [133]. As James et al. [113] explain, Leave-One-Out (LOO-CV) is a cross-validation technique whereby one data point is removed from the data set to be used as a *test set*. Remaining data points in the data set are used for training (i.e., building) the prediction model. Then, the *test set* is used to test the accuracy of the prediction model by using an accuracy measure (e.g. MRE). This entire process is repeated for each data point in the data set. The summary statistic of all predictions (e.g., MMRE) represents the prediction model accuracy.

Although there are different cross-validation techniques, Kocaguneli & Menzies [133] recommend using LOO-CV for increasing reproducibility. In addition, Myrtveit, Stensrud & Shepperd [162] explains that LOO-CV is a technique close to a realistic situation whereby a data set of previous projects is used for predicting a single new project.

C.4 Accuracy of Prediction Models for Software Maintainability

Conte, Dunsmore & Shen [60] recommend MMRE ≤ 0.25 and Pred(0.25) ≥ 0.75 as accuracy references for a good prediction model. Although these references are broadly used in the literature of software effort prediction [127, 204], they date back to 1986, and several researchers have emphasised that they are far from achievable [8, 74, 171, 239, 265]. Using recent and achievable accuracy references is central to support researchers and practitioners when evaluating the efficiency of their prediction models regarding prediction accuracy.

This section reports preliminary results of a systematic literature review carried out to synthesise the accuracy of prediction models for software maintainability. This research focus on software portability, but the literature review concentrates on software maintainability because: (i) we could not identify any prediction model for software portability in the literature; (ii) software maintainability is a quality attribute closely related to portability; and (iii) prediction models for software maintainability have been widely studied in the literature.

Our literature review consists of 16 primary studies published between 1993 and 2015, reporting 252 maintainability prediction models. To find relevant studies, we used the snowballing method [255] that uses a set of primary studies as source for identifying additional studies. References (backward snowballing) and citations (forward snowballing) of included studies are investigated to find additional studies. This procedure is repeated till no additional study is identified.

Our review found eight measures used to evaluate the accuracy of maintainability prediction models (Table C.1). The number of studies adopting these eight measures varies from 1 to 8. Table C.1 shows that researchers used different variations of measures to evaluate the accuracy of their models, such as MMRE (mean of MRE) and Mdn AR (median of AR). Pred(0.30) (n = 8), Pred(0.25) (n = 7), and MRE (n = 6) were the most used accuracy measures. Although MRE lies in the third position in the rank of most used accuracy measures, it is the underlying measure to calculate both Pred(0.30) and Pred(0.25), which highlights its importance in the field. In fact, MRE is the *de facto* standard for measuring the accuracy of software prediction models [162].

Table C.2 summarises statistics for 17 variations of measures used to evaluate the accuracy of 252 maintainability prediction models found in our review. This summary provides a rigorous way to evaluate the accuracy efficiency of maintainability prediction

Table C.1: Measures and their variations used in studies. MRE is the *de facto* standard measure as it is the basis for Pre(0.30) and Pred(0.25).

#	Name	Variations	# of Occurrencies
1	Pred (0.30)		8
2	Pred (0.25)		7
3	Magnitude of Relative Error (MRE)	First/Second quartile of MRE, Max MRE, MdMRE, MMRE, StdMRE	6
4	Absolute Residual (AR)	MAR, Mdn AR, Sum AR	4
5	R-squared	Adjusted r-squared, r-squared	3
6	Correlation (actual-prediction)	Spearman's correlation	3
7	Mean Square Error (MSE)	MSE, NRMSE	2
8	Standard Error of Mean (SEM)		1

Table C.2: Summary of statistics calculated for accuracy measures used in the 252 prediction models found in our review.

#	Name	n	Min	25th Percentile	Median	75th Percentile	Max
1	MSE	132	0.015	0.027	0.031	0.043	0.064
2	MMRE	79	0.00007	0.368	0.493	0.875	4.950
3	Pred (0.30)	54	0.100	0.285	0.455	0.590	0.958
4	Pred (0.25)	43	0.100	0.275	0.370	0.515	0.862
5	NRMSE	30	0.007	0.131	0.192	0.399	0.886
6	Correlation	28	0.059	0.544	0.664	0.713	0.858
7	MdMRE	21	0.079	0.170	0.313	0.420	0.680
8	Max MRE	19	0.000014	1.970	4.820	12.260	24.570
9	SdAR	19	8.418	20.080	46.650	54.390	63.470
10	Sum AR	19	22.0	520.1	1153.0	1548.0	2397.0
11	MdAR	18	9.260	11.630	16.210	17.520	29.540
12	StdMRE	14	0.320	0.582	1.565	2.305	4.430
13	1st Quartile MRE	9	0.048	0.060	0.080	0.192	0.240
14	R-squared	9	0.471	0.560	0.630	0.711	0.740
15	2nd Quartile MRE	9	0.189	0.192	0.220	0.630	0.876
16	SEM	8	0.013	0.018	0.023	0.031	0.052
17	MAR	8	0.083	0.127	0.133	0.149	0.155

Table C.3: Accuracy classification system for MMRE.

#	Classification	Range
1	Excellent	MMRE $\leq$ 0.368
2	Good	0.368 <MMRE $\leq$ 0.493
3	Fair	0.493 <MMRE $\leq$ 0.875
4	Negligible	MMRE >0.875

models. When using this summary to compare the efficiency of accuracy of prediction models, it is important to take two aspects into consideration. First, the number of models used to calculate the summary in Table C.2 varies. Second, the number of studies using a particular type of measure also varies (Table C.1).

Therefore, combining the results of these two tables might increse the confidence for a comparison. For example, Table C.2 shows that MMRE was calculated for 79 prediction models (31.3%). In addition, Table C.1 shows that this measure was used in 6 different studies (37.5%). On the other hand, MSE was calculated for 132 models (52.3%), but this measure was present in only two studies (12.5%). It shows that although MSE was used in a greater number of models, using its summary is less reliable than MMRE that was used by a greater number of independent researchers.

To give a reference for accuracy of prediction models built in this research, we created an accuracy classification system as presented in Table C.3. We used the 25th percentile in Table C.2 as the reference for the best classification (*excellent*). Accuracy values between the 25th percentile and median are considered *good* whereas accuracy values between median and the 75th percentile are considered *fair*. Finally, any MMRE value greater than the 75th percentile is considered *negligible*.

C.5 The Regression Approach

Regression is recognised as the *"de facto"* approach for software effort prediction [99, 117, 151]. As Field, Miles & Field [84] explain, *"Regression analysis is a way of predicting an outcome variable from one predictor variable (simple regression) or several predictor variables (multiple regression)."* The regression model is based on a line that best fits the data set [113]. As a result, the regression model accuracy is evaluated by the goodness with which the line fits the data set. Usually, the goodness of fit is measured by the R^2 or Residual Standard Error (RSE) measures [84, 113]. However, it is worth to note that these measures are specific to regression models and therefore they are not appropriate

for comparing prediction models built with different approaches.

To find the best line, several lines are drawn trying to minimise the distance between data points and lines [84]. The most common method to fit a line to the data (i.e., build the regression model) is the ordinary least squares (OLS). Robust regression is another method to fit a line to the data set that is more resilient to outlying data points [258]. Gray & MacDonell [99] advocate the use of robust regression as software data sets very often are small and subject to outliers.

A data set might include several predictor variables. The literature recommends to quantify the correlation between predictor variables and dependent variable (effort factor) to identify predictor candidates [84, 206]. Forward stepwise is a method to add predictor candidates to a model. An entry condition is defined and several iterations are performed. For each iteration, (i) one predictor candidate is added to the model, (ii) the model accuracy is re-evaluated by applying an evaluation method (e.g., cross-validation), and (iii) the entry criterion is applied to decide whether the predictor candidate should be kept in the model [113]. Forced entry is another method that simply add predictors direct to the model [84].

Gray & MacDonell [99] highlight that generalisability is an important concern when building prediction models. A high generalisable model enables that conclusions are drawn to the entire population from which the data was collected [84]. Generalisability of OLS-based regression models depends on the fulfilment of the following assumptions [84, 113]:

- *Sample size* impacts on the strength of the relationship one want to measure. Ideally, the sample size should be between 10 to 20 data points per predictor on average. Thus, for a model with 3 predictors, the data set should have between 30 to 60 data points in total;

- *Linearity* is a requirement of regression models as the prediction model is based on a line that best fits the model. Therefore the relationship between predictor and effort factor must be linear. Linearity might be evaluated using jitter/scatter plots and statistical tests, such as the Rainbow test. The Rainbow test evaluates the hypothesis that the relationship between variables is linear. Therefore, a p-value less than 0.05 means that the model does not meet the linearity assumption;

- *Independence of residual terms (errors)* or *lack of autocorrelation* is important to achieve accurate confidence and prediction intervals. A common way to test for this assumption is using the Durbin-Watson test that evaluates the null hypothesis

that there is no autocorrelation. The Durbin-Watson test statistics varies from 0 to 4, whereby a value close to 2 means that the independence of residual terms is met. Field, Miles & Field [84] point out that values less than 1 and greater than 3 are real reason for concern;

- *Homoscedasticity* means an homogeneous variance of residual terms. Homoscedasticity might be evaluated using charts and statistical tests, such as Goldfeld-Quandt. The Goldfeld-Quandt test evaluates the null hypothesis that there is heteroscedasticity (i.e., the homoscedasticity assumption is not met);

- *Low leverage* means that data points do not have unusual values in the x-axis. High leverage data points are those that present a *hat value* greater than a cutoff. An usual cutoff is three times the average leverage for the data set, calculated as: $3(k+1)/n$, in which k is the number of predictors and n is the number of observations;

- *No perfect multicollinearity* means that there is no perfect linear relationship between predictors. To test this assumption, the variance inflation factor (VIF) might be used. A VIF value greater than 10 is an indication for multicollinearity to be present. Another measure to test for multicollinearity is the tolerance (1/VIF) that should be greater than 0.1 to consider that the assumption of no multicollinearity is met;

- *Normality* is expected for the distribution of residuals. Normality can be evaluated by a density plot or statistical tests, such as Shapiro-Wilk. The Shapiro-Wilk test evaluates the null hypothesis that the distribution is normally distributed.

Field, Miles & Field [84] highlight that a regression model can be used to drawn conclusions even if assumptions are violated. However, when assumptions are violated, conclusions are limited to the data set only.

Appendix D

Consent Form

Your participation in this experiment is entirely voluntary. You will get paid for the total of X hours, including training and experiment sessions. The payment is not associated to your performance in this experiment, but to the completion of tasks. All data gathered from the experiment will be treated in a confidential fashion: It will be archived in a secure location and will be interpreted only for purposes of this evaluation. When your data are reported or described, all identifying information will be removed. There are no known risks to participation in this experiment. Please feel free to ask the researcher if you have any other questions; otherwise, if you are willing to participate, please sign this consent form and proceed with the experiment.
Date:
Signature:
Researcher's contact details:

Name: Gabriel Costa Silva
Address: RCH/207, Ron Cook Hub, Heslington East, University of York, York, UK
Email : gabriel@cs.york.ac.uk

Supervisors' contact details:

Name: Louis Rose
Address: RCH/102A, Ron Cook Hub, Heslington East, University of York, York, UK
Email: louis.rose@york.ac.uk

D. CONSENT FORM

Name: Radu Calinescu
Address: RCH/102B, Ron Cook Hub, Heslington East, University of York, York, UK
Email: radu.calinescu@york.ac.uk

References

[1] Becoming an Effective Researcher - Part 2 - Ethics: A Practical View for Researchers. Technical report, University of York, York. 25

[2] Cloud Computing Use Cases, 2010. URL `http://cloudusecases.org/`. 22, 36, 53

[3] Use Cases and Functional Requirements for Inter-Cloud Computing, 2010. URL `http://www.gictf.jp/doc/GICTF{_}Whitepaper{_}20100809.pdf`. 37

[4] Toward an Open Cloud Standard. *IEEE Internet Computing*, 16(4):15–25, jul 2012. ISSN 1089-7801. doi: 10.1109/MIC.2012.65. 21, 32

[5] Streamlining devops automation for cloud applications using {TOSCA} as standardized metamodel. *Future Generation Computer Systems*, 56:317 – 332, 2016. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2015.07.017. 180

[6] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. RACS: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, pages 229–240, New York, New York, USA, 2010. ACM Press. ISBN 9781450300360. doi: 10.1145/1807128.1807165. 31, 32, 36, 40, 42

[7] Jehad Al Dallal. Object-oriented class maintainability prediction using internal quality attributes. *Information and Software Technology*, 55(11):2028–2048, November 2013. ISSN 09505849. doi: 10.1016/j.infsof.2013.07.005. 178, 185, 190

[8] Hamdi A. Al-Jamimi and Moataz A. Ahmed. Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model. *IET Software*, 7(6):317–326, December 2013. ISSN 1751-8806. doi: 10.1049/iet-sen.2013.0046. 179, 192, 193, 195

[9] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-642-07888-0. doi: 10.1007/978-3-662-10876-5. 30, 55, 57, 59, 60, 64

[10] Mohammad Alshayeb, Yagoub Eisa, and Moataz A. Ahmed. Object-Oriented Class Stability Prediction: A Comparison Between Artificial Neural Network and Support Vector Machine. *Arabian Journal for Science and Engineering*, 39(11):7865–7876, nov 2014. ISSN 1319-8025. doi: 10.1007/s13369-014-1372-4. 179

[11] David Ameller, Xavier Franch, and Jordi Cabot. Dealing with Non-Functional Requirements in Model-Driven Development. In *2010 18th IEEE International Requirements Engineering Conference*, pages 189–198, Sydney, NSW, 2010. IEEE. ISBN 978-1-4244-8022-7. doi: 10.1109/RE.2010.32. 50

[12] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. How do software architects consider non-functional requirements: An exploratory study. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 41–50, Chicago, IL, 2012. IEEE. ISBN 978-1-4673-2785-5. doi: 10.1109/RE.2012.6345838. 50, 52

[13] Muhammad Bilal Amin, Wajahat Ali Khan, Ammar Ahmad Awan, and Sungyoung Lee. Intercloud message exchange middleware. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication - ICUIMC '12*, pages Article 79 , 7 pages, New York, New York, USA, 2012. ACM Press. ISBN 9781450311724. doi: 10.1145/2184751.2184845. 32

[14] Frank M. Andrews, Laura Klem, Terrence N. Davidson, Patrick M. O'Malley, and Willard L. Rodgers. *A Guide for selecting statistical techniques for analyzing social science data*. Survey Research Center, Institute for Social Research, University of Michigan, Michigan, 2nd edition, 1981. ISBN 9780879442743. 188

[15] Vasilios Andrikopoulos, Tobias Binz, Frank Leymann, and Steve Strauch. How to Adapt Applications for the Cloud Environment. *Computing, Springer*, 95(6):493–535, 2013. doi: 10.1007/s00607-012-0248-2. 28, 29, 30, 34, 49, 50, 53

[16] Vasilios Andrikopoulos, Zhe Song, and Frank Leymann. Supporting the Migration of Applications to the Cloud through a Decision Support System. In *Proceedings of the*

IEEE Sixth International Conference on Cloud Computing, pages 565–572, Santa Clara Marriott, CA, 2013. IEEE. 46

[17] Samuil Angelov and Paul Grefen. An e-contracting reference architecture. *Journal of Systems and Software*, 81(11):1816–1844, November 2008. ISSN 01641212. doi: 10.1016/j.jss.2008.02.023. 45

[18] Danilo Ardagna, Elisabetta Di Nitto, Parastoo Mohagheghi, Sebastien Mosser, Cyril Ballagny, Francesco D'Andria, Giuliano Casale, Peter Matthews, Cosmin-Septimiu Nechifor, Dana Petcu, Anke Gericke, and Craig Sheridan. MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds. In *2012 4th International Workshop on Modeling in Software Engineering (MISE)*, pages 50–56, Zurich, June 2012. IEEE. ISBN 978-1-4673-1757-3. doi: 10.1109/MISE.2012.6226014. 29, 31, 33, 38, 40

[19] Yuji Arimura and Masako Ito. Cloud Computing for Software Development Environment - In-house Deployment at Numazu Software Development Cloud Center. *Fujitsu Scientific and Technical Journal*, 47(3):325–334, 2011. 46, 47

[20] E. Arisholm and D.I.K. Sjøberg. Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *IEEE Transactions on Software Engineering*, 30(8):521–534, 2004. ISSN 0098-5589. doi: 10.1109/TSE.2004.43. 136, 152, 178, 190

[21] Erik Arisholm. Empirical assessment of the impact of structural properties on the changeability of object-oriented software. *Information and Software Technology*, 48(11):1046–1055, 2006. ISSN 09505849. doi: 10.1016/j.infsof.2006.01.002. 51, 52, 57, 58, 62, 185

[22] Michael Armbrust, Ion Stoica, Matei Zaharia, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, and Ariel Rabkin. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010. ISSN 00010782. doi: 10.1145/1721654.1721672. 21, 28, 31, 32, 35, 36

[23] L. Aversano, G. Canfora, A. Cimitile, and A. De Lucia. Migrating legacy systems to the Web: an experience report. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering*, pages 148–157, Lisbon, 2001. IEEE Comput. Soc. ISBN 0-7695-1028-0. doi: 10.1109/.2001.914979. 31, 34, 44, 45, 47, 48, 49

[24] Felix Bachmann, Len Bass, and Robert Nord. Modifiability tactics. Technical Report CMU/SEI-2007-TR-002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2007. URL `http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8299`. 57, 63

[25] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, 33(3):42–52, May 2016. ISSN 0740-7459. doi: 10.1109/MS.2016.64. 55, 57, 179

[26] V Barret and T Lewis. *Outliers in statistical data*. Wiley, New York, 3rd edition, 1994. 152, 193

[27] V. R. Basili and H. D. Rombach. The TAME project: Towards improvement-oriented software environments. *IEEE Trans. Softw. Eng.*, 14(6):758–773, 1988. ISSN 0098-5589. doi: 10.1109/32.6156. URL `http://dx.doi.org/10.1109/32.6156`. 185

[28] V.R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473, 1999. ISSN 00985589. doi: 10.1109/32.799939. 178, 189

[29] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, Boston, Massachusetts, 3rd edition, 2013. 50, 51, 52, 57, 58

[30] Jacob Bellamy-McIntyre, Christof Luterroth, and Gerald Weber. OpenID and the Enterprise: A Model-Based Analysis of Single Sign-On Authentication. In *2011 IEEE 15th International Enterprise Distributed Object Computing Conference*, pages 129–138, Helsinki, 2011. IEEE. ISBN 978-1-4577-0362-1. doi: 10.1109/EDOC.2011.26. 95

[31] Younes Benslimane, Michel Plaisent, Prosper Bernard, and Bouchaib Bahli. Key Challenges and Opportunities in Cloud Computing and Implications on Service Requirements: Evidence from a Systematic Literature Review. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 114–121, Singapore, 2014. IEEE. ISBN 978-1-4799-4093-6. doi: 10.1109/CloudCom.2014.115. 21, 93

[32] Patricia V. Beserra, Alessandro Camara, Rafael Ximenes, Adriano B. Albuquerque, and Nabor C. Mendonca. Cloudstep: A step-by-step decision process to support legacy application migration to the cloud. In *2012 IEEE 6th International Workshop on the*

Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), pages 7–16, Trento, Italy, September 2012. IEEE. ISBN 978-1-4673-3001-5. doi: 10.1109/MESOCA.2012.6392602. 46, 48

[33] Tobias Binz, G. Breiter, Frank Leymann, and T. Spatzier. Portable Cloud Services Using TOSCA. *IEEE Internet Computing*, 16(3):80–85, May 2012. ISSN 1089-7801. doi: 10.1109/MIC.2012.43. 38, 54, 180

[34] Tobias Binz, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. TOSCA: Portable Automated Deployment and Management of Cloud Applications. In *Advanced Web Services*, pages 527–549. Springer, New York, 2014. doi: 10.1007/978-1-4614-7535-4_22. 180

[35] Jean Bozman and Gary Chen. Cloud Computing: The Need for Portability and Interoperability. Technical report, Red Hat, Inc. and IDC Go-to-Market Services, 2011. URL `http://www.redhat.com/resourcelibrary/whitepapers/idc-1001-1`. 22, 31, 32

[36] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012. ISBN 978-1608458820. 52

[37] L.C. Briand, J.W. Daly, and J.K. Wust. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999. ISSN 00985589. doi: 10.1109/32.748920. 11, 58, 59, 62, 63

[38] L.C. Briand, C. Bunse, and J.W. Daly. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *IEEE Transactions on Software Engineering*, 27(6):513–530, 2001. ISSN 00985589. doi: 10.1109/32.926174. 58, 178, 185, 190

[39] Lionel Briand, Prem Devanbu, and Walcelio Melo. An investigation into coupling measures for C++. In *Proceedings of the 19th International Conference on Software Engineering - ICSE '97*, pages 412–421, New York, New York, USA, 1997. ACM Press. ISBN 0897919149. doi: 10.1145/253228.253367. 58, 62

[40] Lionel C. Briand and Jürgen Wüst. Empirical Studies of Quality Models in Object-Oriented Systems. *Advances in Computers*, 56:97–166, 2002. doi: 10.1016/S0065-2458(02)80005-5. 51, 185

[41] John Brooke, Donal Fellows, Kevin Garwood, and Carole Goble. Semantic Matching of Grid Resource Descriptions. In Marios D. Dikaiakos, editor, *Grid Computing (Second European AcrossGrids Conference, AxGrids 2004, Nicosia, Cyprus, January 28-30, 2004. Revised Papers)*, pages 240–249. Springer Berlin Heidelberg, Nicosia, Cyprus, 2004. ISBN 978-3-540-22888-2. doi: 10.1007/978-3-540-28642-4_28. 30

[42] S. Brunett, K. Czajkowski, S. Fitzgerald, I. Foster, A. Johnson, C. Kesselman, J. Leigh, and S. Tuecke. Application experiences with the Globus toolkit. In *Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No.98TB100244)*, pages 81–88, Chicago, IL. IEEE Comput. Soc. ISBN 0-8186-8579-4. doi: 10.1109/HPDC.1998.709959. 30

[43] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc., New York, NY, USA, 1996. ISBN 0-471-95869-7. 57, 58, 63

[44] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6): 599–616, 2009. ISSN 0167-739X. doi: 10.1016/j.future.2008.12.001. 21, 28, 29, 31, 47, 154

[45] Rajkumar Buyya, Rajiv Ranjan, and RodrigoN. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In Ching-Hsien Hsu, LaurenceT. Yang, JongHyuk Park, and Sang-Soo Yeo, editors, *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 13–31. Springer Berlin Heidelberg, Busan, Korea, 2010. ISBN 978-3-642-13118-9. doi: 10.1007/978-3-642-13119-6_2. 28, 39, 40

[46] E.J. Byrne. A conceptual foundation for software re-engineering. In *Proceedings Conference on Software Maintenance 1992*, pages 226–235, Orlando, FL, 1992. IEEE Comput. Soc. Press. ISBN 0-8186-2980-0. doi: 10.1109/ICSM.1992.242539. 52, 58

[47] Capgemini. Business Cloud: The State of Play Shifts Rapidly. Technical report, Capgemini, 2012. URL `http://www.in.capgemini.com/business-cloud-the-state-of-play-shifts-rapidly`. 21, 47, 48, 93

[48] Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, Laura Ricci, and Giacomo Righetti. Cloud Federations in Contrail. In Michael Alexander, Pasqua D'Ambra, Adam Belloum, George Bosilca, Mario Cannataro, Marco Danelutto, Beniamino Martino, Michael Gerndt, Emmanuel Jeannot, Raymond Namyst, Jean Roman, StephenL. Scott, Jesper-Larsson Traff, Geoffroy Vallée, and Josef Weidendorfer, editors, *Euro-Par 2011: Parallel Processing Workshops*, volume 7155 of *Lecture Notes in Computer Science*, pages 159–168. Springer Berlin Heidelberg, Bordeaux, France, 2012. ISBN 978-3-642-29736-6. doi: 10.1007/978-3-642-29737-3_19. 31, 32, 35

[49] J. Carver, L. Jaccheri, S. Morasca, and F. Shull. Issues in using students in empirical studies in software engineering education. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry*, pages 239–249, Sydney, 2003. IEEE Comput. Soc. ISBN 0-7695-1987-3. doi: 10.1109/METRIC.2003.1232471. 154

[50] David W. Chadwick. Federated Identity Management. In *Foundations of Security Analysis and Design V*, pages 96–120. Springer Berlin Heidelberg, Berlin, 2009. doi: 10.1007/978-3-642-03829-7_3. 93, 94

[51] Anirban Chakrabarti, Shubhashis Sengupta, Adarsh Upadhyay, and Anish Damodaran. A Systematic Approach for Application Migration in a Grid Computing Environment. In *2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06)*, pages 512–519, Guangzhou, Guangdong, 2006. IEEE. ISBN 0-7695-2751-5. doi: 10.1109/APSCC.2006.18. 30

[52] Muhammad Aufeef Chauhan and Muhammad Ali Babar. Migrating Service-Oriented System to Cloud Computing: An Experience Report. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 404–411, Washington, DC, 2011. IEEE. ISBN 978-1-4577-0836-7. doi: 10.1109/CLOUD.2011.46. 29, 34, 47, 48, 49, 50

[53] Muhammad Aufeef Chauhan and Muhammad Ali Babar. Towards Process Support for Migrating Applications to Cloud Computing. In *2012 International Conference on Cloud and Service Computing*, pages 80–87, Shanghai, November 2012. IEEE. ISBN 978-1-4673-4724-2. doi: 10.1109/CSC.2012.20. 46, 53

[54] Nitin Singh Chauhan and Ashutosh Saxena. A Green Software Development Life Cycle

for Cloud Computing. *IT Professional*, 15(1):28–34, 2013. ISSN 1520-9202. doi: 10.1109/MITP.2013.6. 29

[55] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. ISSN 00985589. doi: 10.1109/32.295895. 58

[56] E.J. Chikofsky and J.H. Cross. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, 7(1):13–17, January 1990. ISSN 0740-7459. doi: 10.1109/52.43044. 52

[57] Dickson K.W. Chiu, S.C. Cheung, Patrick C.K. Hung, Sherina Y.Y. Chiu, and Andriy K.K. Chung. Developing e-Negotiation support with a meta-modeling approach in a Web services environment. *Decision Support Systems*, 40(1):51–69, July 2005. ISSN 01679236. doi: 10.1016/j.dss.2004.04.004. 45

[58] P Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith A. Stafford. *Documenting Software Architectures: Views and Beyond*. SEI series in software engineering. Addison-Wesley, 2003. ISBN 9780201703726. 57, 58

[59] Eric K Clemons. Making the Decision to Contract for Cloud Services: Managing the Risk of an Extreme Form of IT Outsourcing. In *2011 44th Hawaii International Conference on System Sciences*, pages 1–10, Kauai, HI, 2011. IEEE. ISBN 978-1-4244-9618-1. doi: 10.1109/HICSS.2011.292. 21, 30, 31, 32

[60] S D Conte, H E Dunsmore, and V Y Shen. *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1986. ISBN 0-8053-2162-4. 195

[61] Bruno Costa, Miguel Matos, and Antonio Sousa. Capi: Cloud Computing API. In Luis Rodrigues and Rui Lopes, editors, *Actas do INForum - Simpósio de Informática 2009*, pages 499–502, Lisboa, Portugal, 2009. Faculdade de Ciências da Universidade de Lisboa. doi: 10455/3168. 31, 32

[62] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello. Class point: an approach for the size estimation of object-oriented systems. *IEEE Transactions on Software Engineering*, 31(1):52–74, 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005.5. 51

[63] M. Creeger. Cloud Computing: An Overview. *Queue*, 7(5):2:3—-2:4, 2009. ISSN 1542-7730. doi: 10.1145/1551644.1554608. 21, 28, 29, 31

[64] Andrea De Lucia, Eugenio Pompella, and Silvio Stefanucci. Effort estimation for corrective software maintenance. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering - SEKE '02*, page 409, New York, New York, USA, 2002. ACM Press. ISBN 1581135564. doi: 10.1145/568760.568831. 75

[65] Harpal Dhama. Quantitative models of cohesion and coupling in software. *Journal of Systems and Software*, 29(1):65–74, 1995. ISSN 01641212. doi: 10.1016/0164-1212(94)00128-A. 52, 57, 58

[66] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud Computing: Issues and Challenges. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33, Perth, WA, 2010. IEEE. ISBN 978-1-4244-6695-5. doi: 10.1109/AINA.2010.187. 22, 28, 29, 31, 32, 36

[67] Fred Douglis. Staring at Clouds. *IEEE Internet Computing*, 13(3):4–6, May 2009. ISSN 1089-7801. doi: 10.1109/MIC.2009.70. 31, 32

[68] Scott Dowell, Albert Barreto, James Bret Michael, and Man-Tak Shing. Cloud to cloud interoperability. In *2011 6th International Conference on System of Systems Engineering*, pages 258–263, Albuquerque, NM, June 2011. IEEE. ISBN 978-1-61284-783-2. doi: 10.1109/SYSOSE.2011.5966607. 32

[69] Tore Dybå, Dag I.K. Sjøberg, and Daniela S. Cruzes. What works for whom, where, when, and why? In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '12*, page 19, New York, New York, USA, 2012. ACM Press. ISBN 9781450310567. doi: 10.1145/2372251.2372256. 189

[70] Tore Dybå, Vigdis By Kampenes, and Dag I.K. Sjøberg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745–755, 2006. ISSN 09505849. 11, 12, 13, 66, 71, 72, 102, 114, 132, 142, 188, 189

[71] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting Empirical Methods for Software Engineering Research. In Forrest Shull, Janice Singer, and Dag I. K. Sjø berg, editors, *Selecting Empirical Methods for Software Engineering*

Research, chapter 11, pages 285–311. Springer London, London, 1 edition, 2008. doi: 10.1007/978-1-84800-044-5_11. 24, 183, 185

[72] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. DevOps. *IEEE Software*, 33(3):94–100, may 2016. ISSN 0740-7459. doi: 10.1109/MS.2016.68. 179, 180

[73] Jorge Ejarque, Javier Alvarez, Raul Sirvent, and Rosa M. Badia. A Rule-based Approach for Infrastructure Providers' Interoperability. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 272–279, Athens, November 2011. IEEE. ISBN 978-1-4673-0090-2. doi: 10.1109/CloudCom.2011.44. 33, 40, 41

[74] Mahmoud O. Elish and Karim O. Elish. Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 69–78, Kaiserslautern, 2009. IEEE. ISBN 978-1-4244-3755-9. doi: 10.1109/CSMR.2009.57. 192, 195

[75] Mahmoud O. Elish, Hamoud Aljamaan, and Irfan Ahmad. Three empirical studies on predicting software maintainability using ensemble methods. *Soft Computing*, 19(9):2511–2524, September 2015. ISSN 1432-7643. doi: 10.1007/s00500-014-1576-2. 179

[76] E. Elmroth and J. Tordsson. An Interoperable, Standards-Based Grid Resource Broker and Job Submission Service. In *First International Conference on e-Science and Grid Computing (e-Science'05)*, pages 212–220, Melbourne, 2005. IEEE. ISBN 0-7695-2448-6. doi: 10.1109/E-SCIENCE.2005.17. 30

[77] Joe Masters Emison. State of Cloud Survey. Technical report, InformationWeek, 2014. 21, 31, 32, 37, 55, 59, 93

[78] Joe Masters Emison. Cloud Convergence: 6 Standards That Matter. Technical report, InformationWeek, 2014. 180

[79] Patricia Takako Endo, Glauco Estácio Goncalves, Djamel Fawzi Hadj Sadok, and Judith Kelner. A Survey on Open-source Cloud Computing Solutions. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 3–16, Gramado, 2010. Sociedade Brasileira de Computação (SBC). 21

[80] Tilen Faganel and Matjaz B. Juric. KumuluzEE: Building Microservices with Java EE. *Java magazine*, pages 80–88, 2016. 55, 57

[81] Stephen Farrell. API Keys to the Kingdom. *IEEE Internet Computing*, 13(5):91–93, 2009. ISSN 1089-7801. doi: 10.1109/MIC.2009.100. 95

[82] Janet Feigenspan, Christian Kastner, Jorg Liebig, Sven Apel, and Stefan Hanenberg. Measuring programming experience. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 73–82, Passau, 2012. IEEE. ISBN 978-1-4673-1216-5. doi: 10.1109/ICPC.2012.6240511. 24, 103, 104, 131

[83] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *Proceedings of the IEEE Sixth International Conference on Cloud Computing*, pages 887–894, Santa Clara Marriott, CA, 2013. IEEE. doi: 10.1109/CLOUD.2013.133. 32, 35, 54, 152

[84] A. Field, J. Miles, and Z. Field. *Discovering Statistics Using R*. SAGE Publications, 2012. ISBN 9781446200469. 25, 85, 89, 123, 188, 193, 198, 199, 200

[85] Huber Flores, Satish Narayana Srirama, and Carlos Paniagua. A generic middleware framework for handling process intensive hybrid cloud services from mobiles. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia - MoMM '11*, pages 87–94, New York, New York, USA, 2011. ACM Press. ISBN 9781450307857. doi: 10.1145/2095697.2095715. 33, 40

[86] Teodor-Florin Fortis, Victor Ion Munteanu, and Viorel Negru. Towards an Ontology for Cloud Services. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 787–792, Palermo, July 2012. IEEE. ISBN 978-1-4673-1233-2. doi: 10.1109/CISIS.2012.138. 29, 31, 33, 39, 40

[87] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion mmre. *IEEE Transactions on Software Engineering*, 29(11):985–995, nov 2003. ISSN 0098-5589. doi: 10.1109/TSE.2003.1245300. 194

[88] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001. ISSN 1094-3420. doi: 10.1177/109434200101500302. 30

[89] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *2008 Grid Computing Environments Workshop*, pages 1–10,

Austin, TX, 2008. IEEE. ISBN 978-1-4244-2860-1. doi: 10.1109/GCE.2008.4738445. 28, 30

[90] Gordon L. Freeman and Stephen R. Schach. The task-dependent nature of the maintenance of object-oriented programs. *Journal of Systems and Software*, 76(2):195–206, May 2005. ISSN 01641212. doi: 10.1016/j.jss.2004.05.010. 51

[91] Fabrizio Gagliardi and Silvana Muscella. Cloud Computing – Data Confidentiality and Interoperability Challenges. In Nick Antonopoulos and Lee Gillam, editors, *Cloud Computing*, volume 0 of *Computer Communications and Networks*, pages 257–270. Springer London, London, 2010. ISBN 978-1-84996-240-7. doi: 10.1007/978-1-84996-241-4_15. 36, 46

[92] Fermín Galán, Americo Sampaio, Luis Rodero-Merino, Irit Loy, Victor Gil, and Luis M. Vaquero. Service specification in cloud environments based on extensions to open standards. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE - COMSWARE '09*, page 1. ACM Press, 2009. ISBN 9781605583532. doi: 10.1145/1621890.1621915. 33

[93] Matthias Galster and Eva Bucherer. A Taxonomy for Identifying and Specifying Non-Functional Requirements in Service-Oriented Development. In *2008 IEEE Congress on Services - Part I*, pages 345–352, Honolulu, HI, 2008. IEEE. ISBN 978-0-7695-3286-8. doi: 10.1109/SERVICES-1.2008.51. 50, 52

[94] M. Genero, M. Piattini, E. Manso, and G. Cantone. Building UML class diagram maintainability prediction models based on early metrics. In *Ninth International Software Metrics Symposium, 2003*, pages 263–275, Sydney, 2003. IEEE Comput. Soc. ISBN 0-7695-1987-3. doi: 10.1109/METRIC.2003.1232473. 117, 178, 190, 193

[95] Sushant Goel, Hema Sharda, and David Taniar. Message-Oriented-Middleware in a Distributed Environment. In Thomas Böhme, Gerhard Heyer, and Herwig Unger, editors, *Innovative Internet Community Systems*, pages 93–103. Springer Berlin Heidelberg, 2003. doi: 10.1007/978-3-540-39884-4_8. 57, 60

[96] Kelsey Goings and Paul Abel. Consumer Perceptions of Online Registration and Social Login. Technical report, Blue Research, 2012. URL `http://janrain.com/resources/industry-research/`

2011-consumer-research-perceptions-of-online-registration-and-social-login/. 95

[97] F. Gonidis, I. Paraskakis, and D. Kourtesis. Addressing the Challenge of Application Portability in Cloud Platforms. In *Proceedings of the 7th South East European Doctoral Student Conference (DSC 2012)*, pages 565–576, Thessaloniki, Greece, 2012. RELATE FP7 Marie Curie ITN. 35, 53

[98] Anand Govindarajan and Lakshmanan. Overview of Cloud Standards. In Nick Antonopoulos and Lee Gillam, editors, *Cloud Computing*, volume 0 of *Computer Communications and Networks*, pages 77–89. Springer London, London, 2010. ISBN 978-1-84996-240-7. doi: 10.1007/978-1-84996-241-4_5. 22, 32, 38, 49, 180

[99] Andrew R. Gray and Stephen G. MacDonell. A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology*, 39(6):425–437, January 1997. ISSN 09505849. doi: 10.1016/S0950-5849(96)00006-7. 192, 193, 198, 199

[100] Stein Grimstad, Magne Jørgensen, and Kjetil Moløkken-Østvold. Software effort estimation terminology: The tower of Babel. *Information and Software Technology*, 48(4): 302–310, April 2006. ISSN 09505849. doi: 10.1016/j.infsof.2005.04.004. 191

[101] J.a Guillén, J.b Miranda, and J.M.b Murillo. Decoupling cloud applications from the source: A framework for developing cloud agnostic software. In *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, pages 70–75, Porto, 2012. SciTePress. 32, 55

[102] Julie Hallmark and C. Rebecca Garcia. System Migration: Experiences from the Field, 1992. 44, 45, 48

[103] M Hamdaqa, T Livogiannis, and L Tahvildari. A reference model for developing cloud applications. In *CLOSER 2011 - Proceedings of the 1st International Conference on Cloud Computing and Services Science*, pages 98–103. SciTePress, 2011. 31, 32, 36

[104] Jo Hannay, Dag Sjoberg, and Tore Dyba. A Systematic Review of Theory Use in Software Engineering Experiments. *IEEE Transactions on Software Engineering*, 33(2):87–107, February 2007. ISSN 0098-5589. doi: 10.1109/TSE.2007.12. 74

[105] Piyush Harsh, Florian Dudouet, Roberto G Cascella, Yvon Jégou, and Christine Morin. Using Open Standards for Interoperability - Issues, Solutions, and Challenges facing Cloud Computing. *CoRR (6th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud (2012))*, abs/1207.5, 2012. 31, 36

[106] W. Hasselbring, R. Reussner, H. Jaekel, J. Schlegelmilch, T. Teschke, and S. Krieghoff. The Dublo architecture pattern for smooth migration of business information systems: an experience report. In *Proceedings. 26th International Conference on Software Engineering*, pages 117–126, Scotland, UK, 2004. IEEE Computer Society. ISBN 0-7695-2163-0. doi: 10.1109/ICSE.2004.1317434. 34, 49

[107] Leonard Heilig and Stefan VoB. A Scientometric Analysis of Cloud Computing Literature. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2014. ISSN 2168-7161. doi: 10.1109/TCC.2014.2321168. 21

[108] Zach Hill and Marty Humphrey. CSAL: A Cloud Storage Abstraction Layer to Enable Portable Cloud Applications. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 504–511, Indianapolis, nov 2010. IEEE. ISBN 978-1-4244-9405-7. doi: 10.1109/CloudCom.2010.88. 33

[109] Paul Hofmann and Dan Woods. Cloud Computing: The Limits of Public Clouds for Business Applications. *IEEE Internet Computing*, 14(6):90–93, 2010. ISSN 1089-7801. doi: 10.1109/MIC.2010.136. 21, 28, 31, 32, 36

[110] IEEE. Draft Standard for Intercloud Interoperability and Federation (SIIF). Technical report, 2012. URL `https://www.oasis-open.org/committees/download.php/46205/p2302-12-0002-00-DRFT-intercloud-p2302-draft-0-2.pdf`. 41

[111] Marcelo Alexandre da Cruz Ismael, Cesar Alberto da Silva, Gabriel Costa Silva, and Reginaldo Re. An Empirical Study for Evaluating the Performance of jclouds. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 115–122, Vancouver, 2015. IEEE. ISBN 978-1-4673-9560-1. doi: 10.1109/CloudCom.2015.61. 52, 178, 190

[112] Ronald Jabangwe, Jürgen Börstler, Darja Šmite, and Claes Wohlin. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic

literature review. *Empirical Software Engineering*, 20(3):640–693, March 2014. ISSN 1382-3256. doi: 10.1007/s10664-013-9291-7. 51, 72, 185

[113] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York, New York, NY, 2013. ISBN 978-1-4614-7137-0. doi: 10.1007/978-1-4614-7138-7. 25, 194, 198, 199

[114] M Jonnalagedda, M C Jaeger, U Hohenstein, and G Kaefer. Application portability for public and private clouds. In *CLOSER 2011 - Proceedings of the 1st International Conference on Cloud Computing and Services Science*, pages 484–493, Noordwijkerhout, 2011. SciTePress. 22, 29, 55

[115] M. Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1-2):37–60, feb 2004. ISSN 01641212. doi: 10.1016/S0164-1212(02)00156-5. 192

[116] Magne Jørgensen. What We Do and Don't Know about Software Development Effort Estimation. *IEEE Software*, 31(2):37–40, 2014. ISSN 0740-7459. doi: 10.1109/MS.2014.49. 191, 192, 193

[117] Magne Jørgensen and Martin Shepperd. A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, January 2007. ISSN 0098-5589. doi: 10.1109/TSE.2007.256943. 192, 198

[118] Magne Jørgensen, Tore Dybå, Knut Liestø l, and Dag I.K. Sjøberg. Incorrect results in software engineering experiments: How to improve research practices. *Journal of Systems and Software*, March 2015. ISSN 01641212. doi: 10.1016/j.jss.2015.03.065. 185, 187

[119] F Jrad, Jie Tao, and A Streit. SLA based service brokering in intercloud environments. In *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, pages 76–81, Porto, 2012. SciTePress. 33, 39

[120] Vigdis By Kampenes, Tore Dybå, Jo E. Hannay, and Dag I.K. Sjøberg. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11-12):1073–1086, 2007. ISSN 09505849. doi: 10.1016/j.infsof.2007.02.015. 72, 73, 85, 114, 123, 142, 150, 188, 189

[121] Vigdis By Kampenes, Tore Dybå, Jo E. Hannay, and Dag I. Dag. A systematic review of quasi-experiments in software engineering. *Information and Software Technology*, 51(1): 71–82, 2009. ISSN 09505849. doi: 10.1016/j.infsof.2008.04.006. 153, 185

[122] R. Kazman, S.G. Woods, and S.J. Carriere. Requirements for integrating software architecture and reengineering models: CORUM II. In *Proceedings Fifth Working Conference on Reverse Engineering*, pages 154–163, Honolulu, HI, 1998. IEEE Comput. Soc. ISBN 0-8186-8967-6. doi: 10.1109/WCRE.1998.723185. 58

[123] Ali Khajeh-Hosseini, David Greenwood, and Ian Sommerville. Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 450–457, Miami, FL, July 2010. IEEE. ISBN 978-1-4244-8207-8. doi: 10.1109/CLOUD.2010.37. 47

[124] Ali Khajeh-Hosseini, Ian Sommerville, and Ilango Sriram. Research Challenges for Enterprise Cloud Computing. *CoRR*, abs/1001.3, 2010. 29

[125] Ali Khajeh-Hosseini, David Greenwood, James W. Smith, and Ian Sommerville. The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise. *Software: Practice and Experience*, 42(4):447–465, 2012. ISSN 00380644. doi: 10.1002/spe.1072. 28, 29, 46, 50

[126] B.A. Kitchenham and S.L. Pfleeger. Personal Opinion Surveys. In Forrest Shull, Janice Singer, and DagI.K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London, London, 2008. ISBN 978-1-84800-043-8. doi: 10.1007/978-1-84800-044-5_3. 25, 183

[127] B.A. Kitchenham, L.M. Pickard, S.G. MacDonell, and M.J. Shepperd. What accuracy statistics really measure. *IEE Proceedings - Software*, 148(3):81, 2001. ISSN 14625970. doi: 10.1049/ip-sen:20010506. 193, 194, 195

[128] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002. ISSN 0098-5589. doi: 10.1109/TSE.2002.1027796. 185, 187, 190

[129] Barbara Kitchenham. What's up with software metrics? – A preliminary mapping study. *Journal of Systems and Software*, 83(1):37–51, 2010. ISSN 01641212. doi: 10.1016/j.jss.2009.06.041. 51, 72

[130] 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):316–329, May 2007. ISSN 0098-5589. doi: 10.1109/TSE.2007.1001. 193

[131] Andrew J. Ko, Thomas D. LaToza, and Margaret M. Burnett. A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering*, 20(1):110–141, 2013. ISSN 1382-3256. doi: 10.1007/s10664-013-9279-3. 96, 103, 146, 152, 185, 187

[132] AndrewJ. Ko, ThomasD. LaToza, and MargaretM. Burnett. A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering*, 20(1):110–141, 2015. ISSN 1382-3256. doi: 10.1007/s10664-013-9279-3. 75

[133] Ekrem Kocaguneli and Tim Menzies. Software effort models should be assessed via leave-one-out validation. *Journal of Systems and Software*, 86(7):1879–1890, July 2013. ISSN 01641212. doi: 10.1016/j.jss.2013.02.053. 194

[134] Stefan Kolb, Jorg Lenhard, and Guido Wirtz. Application migration effort in the cloud - the case of cloud platforms. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 41–48, New York City, NY, June 2015. IEEE. ISBN 978-1-4673-7287-9. doi: 10.1109/CLOUD.2015.16. 53, 152

[135] Shyam Kotecha, Minal Bhise, and Sanjay Chaudhary. Query translation for cloud databases. In *2011 Nirma University International Conference on Engineering*, pages 1–4, Ahmedabad, Gujarat, December 2011. IEEE. ISBN 978-1-4577-2168-7. doi: 10.1109/NUiConE.2011.6153249. 33

[136] Maurizio Lancia, Roberto Puccinelli, and Flavio Lombardi. Feasibility and benefits of migrating towards JEE. In *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java - PPPJ '07*, page 13, New York, New York, USA,

2007. ACM Press. ISBN 9781595936721. doi: 10.1145/1294325.1294328. 31, 34, 45, 47, 49

[137] C. Larman. *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*. Prentice Hall PTR, 2002. ISBN 9780130925695. 63

[138] Bu Sung Lee, Shixing Yan, Ding Ma, and Guopeng Zhao. Aggregating IaaS Service. In *2011 Annual SRII Global Conference*, pages 335–338, San Jose, CA, March 2011. IEEE. ISBN 978-1-61284-415-2. doi: 10.1109/SRII.2011.44. 22, 32, 55

[139] Barry Leiba. OAuth Web Authorization Protocol. *IEEE Internet Computing*, 16(1): 74–77, 2012. ISSN 1089-7801. doi: 10.1109/MIC.2012.11. 94, 95

[140] Lydia Leong, Douglas Toombs, Bob Gill, Gregor Petri, and Tiny Haynes. Magic Quadrant for Cloud Infrastructure as a Service. Technical report, Gartner, Inc, 2013. URL `http://www.gartner.com/technology/reprints.do?id=1-1IMDMZ5&ct=130819&st=sb`. 144, 154

[141] Grace Lewis. The Role of Standards in Cloud-Computing Interoperability. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2012. URL `http://www.sei.cmu.edu/library/abstracts/reports/12tn012.cfm`. 31, 32

[142] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. CloudCmp: comparing public cloud providers. In *Proceedings of the 10th Annual Conference on Internet Measurement - IMC '10*, pages 1–14, Melbourne, Australia, 2010. ACM Press. ISBN 9781450304832. doi: 10.1145/1879141.1879143. 134, 154

[143] Y. F. Li, M. Xie, and T. N. Goh. Bayesian Inference Approach for Probabilistic Analogy Based Software Maintenance Effort Estimation. In *2008 14th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 176–183, Taipei, December 2008. IEEE. doi: 10.1109/PRDC.2008.21. 192, 193

[144] Zheng Li, Liam O'Brien, He Zhang, and Rainbow Cai. On a Catalogue of Metrics for Evaluating Commercial Cloud Services. In *2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 164–173, Beijing, 2012. IEEE. ISBN 978-1-4673-2901-9. doi: 10.1109/Grid.2012.15. 21

[145] Joa Sang Lim, Seung Ryul Jeong, and Stephen R. Schach. An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software. *Journal of Systems and Software*, 77(2):131–138, 2005. ISSN 01641212. doi: 10.1016/j.jss.2004.11.004. 51

[146] Fang Liu, Jin Tong, Jian Mao, Robert B. Bohn, John V. Messina, Mark L. Badger, and Dawn M. Leaf. NIST Cloud Computing Reference Architecture. Technical report, National Institute of Standards and Technology (NIST), 2011. URL `http://www.nist.gov/manuscript-publication-search.cfm?pub{_}id=909505`. 22

[147] Nikolaos Loutas, Vassilios Peristeras, Thanassis Bouras, Eleni Kamateri, Dimitrios Zeginis, and Konstantinos Tarabanis. Towards a Reference Architecture for Semantically Interoperable Clouds. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 143–150, Indianapolis, November 2010. IEEE. ISBN 978-1-4244-9405-7. doi: 10.1109/CloudCom.2010.38. 22, 28, 30, 33, 41

[148] Nikolaos Loutas, Eleni Kamateri, Filippo Bosi, and Konstantinos Tarabanis. Cloud Computing Interoperability: The State of Play. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 752–757, Athens, November 2011. IEEE. ISBN 978-1-4673-0090-2. doi: 10.1109/CloudCom.2011.116. 22, 38, 49

[149] Nikolaos Loutas, Eleni Kamateri, and Konstantinos Tarabanis. A Semantic Interoperability Framework for Cloud Platform as a Service. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 280–287, Athens, nov 2011. IEEE. ISBN 978-1-4673-0090-2. doi: 10.1109/CloudCom.2011.45. 33

[150] L. Lynch. Inside the Identity Management Game. *IEEE Internet Computing*, 15(5): 78–82, 2011. ISSN 1089-7801. doi: 10.1109/MIC.2011.119. 95

[151] C. Mair and M. Shepperd. The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In *2005 International Symposium on Empirical Software Engineering, 2005.*, pages 491–500, Noosa Heads, 2005. IEEE. ISBN 0-7803-9507-7. doi: 10.1109/ISESE.2005.1541858. 191, 192, 198

[152] Giuliano Manno, Waleed W. Smari, and Luca Spalazzi. FCFA: A semantic-based federated cloud framework architecture. In *2012 International Conference on High Perfor-*

mance Computing & Simulation (HPCS), pages 42–52, Madrid, July 2012. IEEE. ISBN 978-1-4673-2362-8. doi: 10.1109/HPCSim.2012.6266889. 33

[153] Simon Maple. Java Tools and Technologies Landscape 2016. Technical report, Rebellabs by ZeroTurnaround, 2016. URL `http://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016/`. 104

[154] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing – The business perspective. *Decision Support Systems*, 51(1):176–189, 2011. ISSN 01679236. doi: 10.1016/j.dss.2010.12.006. 28, 31

[155] Peter Mell and Timoty Grance. NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology (Special Publication 800-145), Gaithersburg, 2011. URL `http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909616`. 21, 28, 29, 37

[156] D.A. Menasce. MOM vs. RPC: Communication Models for Distributed Applications. *IEEE Internet Computing*, 9(2):90–93, March 2005. ISSN 1089-7801. doi: 10.1109/MIC.2005.42. 55, 59

[157] Andre Merzky, Katerina Stamou, and Shantenu Jha. Application Level Interoperability between Clouds and Grids. In *2009 Workshops at the Grid and Pervasive Computing Conference*, pages 143–150, Geneva, May 2009. IEEE. ISBN 978-0-7695-3677-4. doi: 10.1109/GPC.2009.17. 22, 33, 34, 38

[158] James Miller. Statistical significance testing – a panacea for software technology experiments? *Journal of Systems and Software*, 73(2):183–192, October 2004. ISSN 01641212. doi: 10.1016/j.jss.2003.12.019. 188

[159] James Miller. Replicating software engineering experiments: a poisoned chalice or the Holy Grail. *Information and Software Technology*, 47(4):233–244, mar 2005. ISSN 09505849. doi: 10.1016/j.infsof.2004.08.005. 178, 190

[160] James Miller, John Daly, Murray Wood, Marc Roper, and Andrew Brooks. Statistical power and its subcomponents – missing and misunderstood concepts in empirical software engineering research. *Information and Software Technology*, 39(4):285–295, 1997. ISSN 09505849. doi: 10.1016/S0950-5849(96)01139-1. 189

[161] Javier Miranda, Juan Manuel Murillo, Joaquín Guillén, and Carlos Canal. Identifying adaptation needs to avoid the vendor lock-in effect in the deployment of cloud SBAs. In *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups on - WAS4FI-Mashups '12*, page 12, New York, New York, USA, 2012. ACM Press. ISBN 9781450315661. doi: 10.1145/2377836.2377841. 55

[162] I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering*, 31(5):380–391, May 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005.58. 191, 192, 193, 194, 195

[163] Balakrishna Narasimhan and Ryan Nichols. State of Cloud Applications and Platforms: The Cloud Adopters' View. *Computer*, 44(3):24–28, March 2011. ISSN 0018-9162. doi: 10.1109/MC.2011.66. 21, 28, 32

[164] V. Nelson and V. Uma. Semantic based Resource Provisioning and scheduling in inter-cloud environment. In *2012 International Conference on Recent Trends in Information Technology*, pages 250–254, Chennai, Tamil Nadu, April 2012. IEEE. ISBN 978-1-4673-1601-9. doi: 10.1109/ICRTIT.2012.6206823. 33, 40

[165] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015. ISBN 1491950358, 9781491950357. 179

[166] Reason Baathuli Nfila, Motumi Nini Dintwe, and K.N. Rao. Experience of systems migration at the University of Botswana Library: a case study. *Program: electronic library and information systems*, 39(3):248–256, 2005. ISSN 0033-0337. doi: 10.1108/00330330510610582. 35, 44, 45, 47, 48

[167] BinhMinh Nguyen, Viet Tran, and Ladislav Hluchy. High-Level Abstraction Layers for Development and Deployment of Cloud Services. In Rachid Benlamri, editor, *Networked Digital Technologies*, volume 293 of *Communications in Computer and Information Science*, pages 208–219. Springer Berlin Heidelberg, Dubai, 2012. ISBN 978-3-642-30506-1. doi: 10.1007/978-3-642-30507-8_19. 33

[168] Vu Nguyen, Barry Boehm, and Phongphan Danphitsanuphan. A controlled experiment in assessing and estimating software maintenance tasks. *Information and Software Technology*, 53(6):682–691, 2011. ISSN 09505849. doi: 10.1016/j.infsof.2010.11.003. 75, 110

[169] NorthBridge and GigaOM. 2013 Future of Cloud Computing 3rd Annual Survey Results, 2013. URL `http://www.northbridge.com/2013-cloud-computing-survey`. 21, 32, 47, 48, 93

[170] Harshad Oak. Build with NetBeans IDE, Deploy to Oracle Java Cloud Service. *Java magazine*, pages 67–72, 2014. URL `http://oracle.com/javamagazine`. 54, 152

[171] Sunday Olusanya Olatunji and Ali Selamat. Type-2 Fuzzy Logic Based Prediction Model of Object Oriented Software Maintainability. In Hamido Fujita and Ali Selamat, editors, *Intelligent Software Methodologies, Tools and Techniques*, pages 329–342. Springer International Publishing, 2015. doi: 10.1007/978-3-319-17530-0_23. 195

[172] Eric Olden. Architecting a Cloud-Scale Identity Fabric. *Computer*, 44(3):52–59, March 2011. ISSN 0018-9162. doi: 10.1109/MC.2011.60. 95

[173] Justice Opara-Martins, Reza Sahandi, and Feng Tian. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, 5(1):4, December 2016. ISSN 2192-113X. doi: 10.1186/s13677-016-0054-z. 21, 22, 37, 43

[174] Sofia Ouhbi, Ali Idri, Jose Luis Fernandez Aleman, and Ambrosio Toval. Evaluating Software Product Quality: A Systematic Mapping Study. In *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, pages 141–151, Rotterdam, October 2014. IEEE. doi: 10.1109/IWSM.Mensura.2014.30. 50

[175] Michael Papazoglou. *Web Services: Principles and Technology*. Pearson-Prentice Hall, Harlow, 1 edition, 2007. ISBN 978-0321155559. 30, 57, 59

[176] Michael Papazoglou. Cloud Blueprints for Integrating and Managing Cloud Federations. In Maritta Heisel, editor, *Software Service and Application Engineering*, volume 7365 of *Lecture Notes in Computer Science*, pages 102–119. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-30834-5. doi: 10.1007/978-3-642-30835-2. 22, 32

[177] Michael P. Papazoglou and Benedikt Kratz. Web services technology in support of business transactions. *Service Oriented Computing and Applications*, 1(1):51–63, 2007. ISSN 1863-2386. doi: 10.1007/s11761-007-0002-3. 30

[178] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.400. 30

[179] Fawaz Paraiso, Nicolas Haderer, Philippe Merle, Romain Rouvoy, and Lionel Seinturier. A Federated Multi-cloud PaaS Infrastructure. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 392–399, Honolulu, HI, 2012. IEEE. ISBN 978-1-4673-2892-0. doi: 10.1109/CLOUD.2012.79. 41

[180] A.V. Parameswaran and A. Chaddha. Cloud Interoperability and Standardization. *Infosys Technology Limited/SETLabs Briefings*, 7(7):19–26, 2009. 32

[181] Russell Pavlicek. *Unikernels*. O'Reilly Media, Inc., 1st edition, 2016. ISBN 9781491959244. 179

[182] Przemyslaw Pawluk, Bradley Simmons, Michael Smit, Marin Litoiu, and Serge Mankovski. Introducing STRATOS: A Cloud Broker Service. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 891–898, Honolulu, HI, June 2012. IEEE. ISBN 978-1-4673-2892-0. doi: 10.1109/CLOUD.2012.24. 40

[183] Mikhail Perepletchikov and Caspar Ryan. A Controlled Experiment for Evaluating the Impact of Coupling on the Maintainability of Service-Oriented Software. *IEEE Transactions on Software Engineering*, 37(4):449–465, 2011. ISSN 0098-5589. doi: 10.1109/TSE.2010.61. 52, 57, 58, 62, 75

[184] Dana Petcu. Portability and interoperability between clouds: Challenges and case study. *Towards a Service-Based Internet*, 6994 LNCS:62–74, 2011. doi: 10.1007/978-3-642-24755-2_6. 30, 32, 36, 37, 38, 49, 53, 54

[185] Dana Petcu and Athanasios V. Vasilakos. Portability in clouds: approaches and research opportunities. *Scalable Computing: Practice and Experience*, 15(3):251–270, October 2014. ISSN 1895-1767. doi: 10.12694/scpe.v15i3.1019. 22, 53

[186] Dana Petcu, Georgiana Macariu, Silviu Panica, and Ciprian Crăciun. Portable Cloud applications – From theory to practice. *Future Generation Computer Systems*, 29(6):1417–1430, 2012. ISSN 0167739X. doi: 10.1016/j.future.2012.01.009. 22, 29, 31, 32, 38, 39, 40, 41, 50, 53, 55, 57, 58

[187] Dana Petcu, Beniamino Di Martino, Salvatore Venticinque, Massimiliano Rak, Tamás Máhr, Gorka Esnal Lopez, Fabrice Brito, Roberto Cossu, Miha Stopar, Svatopluk Šperka, and Vlado Stankovski. Experiences in building a mOSAIC of clouds. *Journal of Cloud Computing*, 2(1):1–22, 2013. doi: 10.1186/2192-113X-2-12. 152

[188] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic Mapping Studies in Software Engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swinton, UK, UK, 2008. British Computer Society. 37

[189] Madhukara Phatak and V.N Kamalesh. On cloud computing deployment architecture. In *2010 International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 11–14, Colombo, September 2010. IEEE. ISBN 978-1-4244-9041-7. doi: 10.1109/ICTER.2010.5643276. 32

[190] Lesley M. Pickard, Barbara A. Kitchenham, and Peter W. Jones. Combining empirical results in software engineering. *Information and Software Technology*, 40(14):811–821, 1998. ISSN 09505849. doi: 10.1016/S0950-5849(98)00101-3. 76

[191] Ajith Ranabahu and Amit P. Sheth. Semantics Centric Solutions for Application and Data Portability in Cloud Computing. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 234–241, Indianapolis, 2010. IEEE. ISBN 978-1-4244-9405-7. doi: 10.1109/CloudCom.2010.48. 22, 31, 32, 33, 34

[192] Ajith Ranabahu, Eugene Michael Maximilien, Amit P. Sheth, and Krishnaprasad Thirunarayan. A domain specific language for enterprise grade cloud-mobile hybrid applications. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11 - SPLASH '11 Workshops*, pages 77–84, New York, New York, USA, oct 2011. ACM Press. ISBN 9781450311830. doi: 10.1145/2095050.2095064. 31, 32, 36

[193] M. A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004. ISSN 0018-8670. doi: 10.1147/sj.431.0032. 28

[194] Mehwish Riaz, Emilia Mendes, and Ewan Tempero. A systematic review of software maintainability prediction and metrics. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, pages 367–377, Lake Buena

Vista, FL, 2009. IEEE. ISBN 9781424448418. doi: 10.1109/ESEM.2009.5314233. 51, 52, 57, 72

[195] Martin P. Robillard. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 26(6):27–34, 2009. ISSN 0740-7459. doi: 10.1109/MS.2009.193. 117

[196] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, Ignacio M. Llorente, Rubén S. Montero, Y. Wolfsthal, Erik Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and Fermín Galán. The Reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4:1–4:11, 2009. ISSN 0018-8646. doi: 10.1147/JRD.2009.5429058. 30, 33, 38, 39, 41

[197] Luis Rodero-Merino, Luis M. Vaquero, Victor Gil, Fermín Galán, Javier Fontán, Rubén S. Montero, and Ignacio M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, October 2010. ISSN 0167739X. doi: 10.1016/j.future.2010.02.013. 22, 30, 32

[198] Jarrett Rosenberg. Statistical Methods and Measurement. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 155–184. Springer London, London, 2008. ISBN 978-1-84800-043-8. 188

[199] Americo Sampaio and Nabor Mendonça. Uni4Cloud: An approach based on open standards for deployment and management of multi-cloud applications. In *Proceeding of the 2nd International Workshop on Software Engineering for Cloud Computing - SECLOUD '11*, page 15, New York, New York, USA, May 2011. ACM Press. ISBN 9781450305822. doi: 10.1145/1985500.1985504. 33, 41, 54, 152

[200] Juliana Saraiva. A roadmap for software maintainability measurement. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1453–1455, San Francisco, CA, May 2013. IEEE. ISBN 978-1-4673-3076-3. doi: 10.1109/ICSE.2013.6606742. 52, 57, 72

[201] Juliana de A.G. Saraiva, Micael S. de França, Sérgio C.B. Soares, Fernando J.C.L. Filho, and Renata M.C.R. de Souza. Classifying Metrics for Assessing Object-Oriented Software Maintainability: A Family of Metrics' Catalogs. *Journal of Systems and Software*, 103: 85–101, 2015. ISSN 01641212. doi: 10.1016/j.jss.2015.01.014. 51, 58

[202] Benjamin Satzger, Waldemar Hummer, Christian Inzinger, Philipp Leitner, and Schahram Dustdar. Winds of Change: From Vendor Lock-In to the Meta Cloud. *IEEE Internet Computing*, 17(1):69–73, 2013. ISSN 1089-7801. doi: 10.1109/MIC.2013.19. 21, 22, 31, 32, 35

[203] Carolyn B. Seaman. Qualitative Methods. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, chapter Section I, pages 35–62. Springer London, 2008. doi: 10.1007/978-1-84800-044-5_2. 24

[204] Yeong-Seok Seo and Doo-Hwan Bae. On the value of outlier elimination on software effort estimation research. *Empirical Software Engineering*, 18(4):659–698, 2013. ISSN 1382-3256. doi: 10.1007/s10664-012-9207-y. 153, 188, 193, 195

[205] Sam Sepassi. Scalable Microservices Using Modern Java-Based Frameworks, 2016. URL `https://dzone.com/articles/scalable-microservices-using-kumuluz-ee-framework?edition=201745{&}utm{_}source=Spotlight{&}utm{_}medium=email{&}utm{_}campaign=integration2016-08-25`. 55, 57

[206] W R Shadish, T D Cook, and D T Campbell. *Experimental and Quasi-experimental Designs for Generalized Causal Inference*. Houghton Mifflin, Boston, 2002. ISBN 9780395615560. 85, 123, 199

[207] Chen Shan, Chang Heng, and Zou Xianjun. Inter-cloud operations via NGSON. *IEEE Communications Magazine*, 50(1):82–89, 2012. ISSN 0163-6804. doi: 10.1109/MCOM.2012.6122536. 32

[208] Mahdi Negahi Shirazi, Ho Chin Kuan, and Hossein Dolatabadi. Design Patterns to Enable Data Portability between Clouds' Databases. In *2012 12th International Conference on Computational Science and Its Applications*, pages 117–120, Salvador, June 2012. IEEE. ISBN 978-1-4673-1691-0. doi: 10.1109/ICCSA.2012.29. 22, 32

[209] Forrest Shull and Raimund L. Feldmann. Building Theories from Multiple Evidence Sources. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, chapter 13, pages 337–364. Springer London, London, 1 edition, 2008. doi: 10.1007/978-1-84800-044-5_13. 74, 178, 189, 190

[210] Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. A Systematic Review of Cloud Lock-In Solutions. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, pages 363–368, Bristol, UK, 2013. IEEE. ISBN 978-0-7695-5095-4. doi: 10.1109/CloudCom.2013.130. 22, 37, 49, 115

[211] Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. Cloud DSL: A Language for Supporting Cloud Portability by Describing Cloud Entities. In Richard Paige, Jordi Cabot, Marco Brambilla, Louis Rose, and James H. Hil, editors, *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud*, pages 36–45, Valencia, Spain, 2014. 22, 115

[212] Dag I. K. Sjøberg, Tore Dyba, and Magne Jørgensen. The Future of Empirical Methods in Software Engineering Research. In *Future of Software Engineering (FOSE '07)*, pages 358–378, Minneapolis, MN, May 2007. IEEE. ISBN 0-7695-2829-5. doi: 10.1109/FOSE.2007.30. 25, 183

[213] Dag I. K. Sjøberg, Tore Dybå, Bente C. D. Anda, and Jo E. Hannay. Building Theories in Software Engineering. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, chapter 12, pages 312–336. Springer London, London, 1 edition, 2008. doi: 10.1007/978-1-84800-044-5_12. 24, 74, 183

[214] D.I.K. Sjøberg, B. Anda, E. Arisholm, T. Dyba, M. Jørgensen, A. Karahasanovic, E.F. Koren, and M. Vokac. Conducting realistic experiments in software engineering. In *Proceedings International Symposium on Empirical Software Engineering*, pages 17–26, Nara, 2002. IEEE Comput. Soc. ISBN 0-7695-1796-X. doi: 10.1109/ISESE.2002.1166921. 96, 103, 185, 187

[215] D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A.C. Rekdal. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 31(9):733–753, 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005.97. 74, 75, 76, 115, 152, 154, 187, 190

[216] Harry M. Sneed and Katalin Erdoes. Migrating AS400-COBOL to Java: A Report from the Field. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 231–240, Genova, mar 2013. IEEE. ISBN 978-0-7695-4948-4. doi: 10.1109/CSMR.2013.32. 22, 31, 34, 44, 45, 46, 47, 48, 49

[217] H.M. Sneed. Risks involved in reengineering projects. In *Sixth Working Conference on Reverse Engineering (Cat. No.PR00303)*, pages 204–211, Atlanta, GA, 1999. IEEE Computer Society. ISBN 0-7695-0303-9. doi: 10.1109/WCRE.1999.806961. 52

[218] Associação para Promoção da Excelência do Software Brasileiro SOFTEX. *MPS.BR-Guia de Aquisição*. Associação para Promoção da Excelência do Software Brasileiro – SOFTEX, 2009. ISBN 978-85-99334-14-0. URL `http://www.softex.br`. 45

[219] Thamarai Selvi Somasundaram, Kannan Govindarajan, M.r Rajagopalan, and S. Madhusudhana Rao. An architectural framework to solve the interoperability issue between private clouds using semantic technology. In *2012 International Conference on Recent Trends in Information Technology*, pages 162–167, Chennai, Tamil Nadu, April 2012. IEEE. ISBN 978-1-4673-1601-9. doi: 10.1109/ICRTIT.2012.6206764. 33, 39, 40

[220] Ian Sommerville. *Software Engineering*. Addison-Wesley, Harlow, 8th edition, 2007. ISBN 0321313798. 49, 50, 52, 58, 180

[221] Arthur E. C. da Silva Souza, José A. Medeiros de Lima, Renato Gondim, Thomas Diniz, Nelio Cacho, Frederico Lopes, and Thais Batista. Avaliando o Aprisionamento entre Várias Plataformas de Computação em Nuvem. In *31 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 775–788, Brasília, 2013. 29, 33

[222] Bruno Souza and Edson Yanaga. 7 Open Source Tools for Java Deployment. *Java magazine*, pages 05–13, 2014. URL `http://oracle.com/javamagazine`. 54, 152

[223] Diomidis Spinellis. Portability: Goodies vs. the Hair Shirt. *IEEE Software*, 30(4):22–23, July 2013. ISSN 0740-7459. doi: 10.1109/MS.2013.82. 51

[224] E. Stensrud, T. Foss, B. Kitchenham, and I. Myrtveit. An empirical validation of the relationship between the magnitude of relative error and project size. In *Proceedings Eighth IEEE Symposium on Software Metrics*, pages 3–12, Ottawa, Canada, 2002. IEEE Comput. Soc. ISBN 0-7695-1339-5. doi: 10.1109/METRIC.2002.1011320. 193, 194

[225] Roman Suvorov, Meiyappan Nagappan, Ahmed E. Hassan, Ying Zou, and Bram Adams. An empirical study of build system migrations in practice: Case studies on KDE and the Linux kernel. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 160–169, Trento, September 2012. IEEE. ISBN 978-1-4673-2312-3. doi: 10.1109/ICSM.2012.6405267. 44, 45, 48

[226] Ladan Tahvildari, Kostas Kontogiannis, and John Mylopoulos. Quality-driven software re-engineering. *Journal of Systems and Software*, 66(3):225–239, 2003. ISSN 01641212. doi: 10.1016/S0164-1212(02)00082-1. 52, 58

[227] D. Talia. The Open Grid Services Architecture: where the grid meets the Web. *IEEE Internet Computing*, 6(6):67–71, 2002. ISSN 1089-7801. doi: 10.1109/MIC.2002.1067739. 30

[228] A.S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, 2007. ISBN 9780132392273. 30, 57, 60, 94

[229] Jie Tao, Holger Marten, David Kramer, and Wolfgang Karl. An Intuitive Framework for Accessing Computing Clouds. *Procedia Computer Science*, 4(1):2049–2057, January 2011. ISSN 18770509. doi: 10.1016/j.procs.2011.04.224. 32, 40

[230] Werner Teppe. The ARNO Project: Challenges and Experiences in a Large-Scale Industrial Software Migration Project. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 149–158, Kaiserslautern, 2009. IEEE. ISBN 978-1-4244-3755-9. doi: 10.1109/CSMR.2009.64. 34, 35, 44, 45, 46, 47, 48

[231] Mie Mie Thet Thwin and Tong-Seng Quah. Application of neural networks for software quality prediction using object-oriented metrics. *Journal of Systems and Software*, 76(2): 147–156, May 2005. ISSN 01641212. doi: 10.1016/j.jss.2004.05.001. 179, 192

[232] Van Tran, Jacky Keung, Anna Liu, and Alan Fekete. Application migration to cloud: A Taxonomy of Critical Factors. In *Proceeding of the 2nd International Workshop on Software Engineering for Cloud Computing - SECLOUD '11*, pages 22–28, New York, New York, USA, 2011. ACM Press. ISBN 9781450305822. doi: 10.1145/1985500.1985505. 28, 34, 46, 47, 48, 49

[233] Van T.K. Tran, Kevin Lee, Alan Fekete, Anna Liu, and Jacky Keung. Size Estimation of Cloud Migration Projects with Cloud Migration Point (CMP). In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 265–274, Banff, AB, September 2011. IEEE. ISBN 978-1-4577-2203-5. doi: 10.1109/ESEM.2011.35. 192

[234] Adam Trendowicz, Jürgen Münch, and Ross Jeffery. State of the Practice in Software Effort Estimation: A Survey and Literature Review. In Zbigniew Huzar, Radek Koci,

Bertrand Meyer, Bartosz Walter, and Jaroslav Zendulka, editors, *Software Engineering Techniques*, pages 232–245. Springer Berlin Heidelberg, Berlin, 2011. ISBN 978-3-642-22385-3. doi: 10.1007/978-3-642-22386-0_18. 191, 192

[235] Wei-Tek Tsai, Xin Sun, and Janaka Balasooriya. Service-Oriented Cloud Computing Architecture. In *2010 Seventh International Conference on Information Technology: New Generations*, pages 684–689, Las Vegas, NV, 2010. IEEE. ISBN 978-1-4244-6270-4. doi: 10.1109/ITNG.2010.214. 22, 33, 39

[236] Milan K. Vachhani and Kishor H. Atkotiya. Globus Toolkit 5 (GT5): Introduction of a tool to develop Grid Application and Middleware. *International Journal of Emerging Technology and Advanced Engineering*, 2(7):174–178, 2012. 30

[237] S.S. Vadhiyar and J.J. Dongarra. A performance oriented migration framework for the grid. In *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.*, pages 130–137, Tokyo, Japan, 2003. IEEE. ISBN 0-7695-1919-9. doi: 10.1109/CCGRID.2003.1199361. 34

[238] Frank van der Linden. Porting NetBSD to the AMD x86-641: a case study in OS portability. In *Proceedings of the BSDCon '02 Conference on File and Storage Technologies*, pages 1–10, San Francisco, California, 2002. USENIX, the Advanced Computing Systems Association. 22, 34, 44, 45, 46, 47, 48

[239] C. van Koten and A.R. Gray. An application of Bayesian network for predicting object-oriented software maintainability. *Information and Software Technology*, 48(1):59–67, January 2006. ISSN 09505849. doi: 10.1016/j.infsof.2005.03.002. 192, 195

[240] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds. *ACM SIGCOMM Computer Communication Review*, 39(1):50, 2008. ISSN 01464833. doi: 10.1145/1496091.1496100. 29

[241] Jinesh Varia. Cloud Architectures. Technical report, Amazon Web Services, Inc., 2008. URL `http://media.amazonwebservices.com/AWS_Cloud_Architectures.pdf`. 28

[242] Jinesh Varia. Architecting for the Cloud: Best Practices. Technical report, Amazon Web Services, Inc., 2011. URL `http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf`. 53, 55

[243] C. Verhoef and A.A. Terekhov. The realities of language conversions. *IEEE Software*, 17 (6):111–124, 2000. ISSN 07407459. doi: 10.1109/52.895180. 52

[244] Gil Vernik, Alexandra Shulman-Peleg, Sebastian Dippl, Ciro Formisano, Michael C. Jaeger, Elliot K. Kolodner, and Massimo Villari. Data On-boarding in Federated Storage Clouds. In *Proceedings of the IEEE Sixth International Conference on Cloud Computing*, pages 244–251, Santa Clara Marriott, CA, 2013. IEEE. 41

[245] S. Vinoski. Where is middleware. *IEEE Internet Computing*, 6(2):83–85, 2002. ISSN 10897801. doi: 10.1109/4236.991448. 55, 64

[246] Stefan Wagner. Quality Models. In *Software Product Quality Control*, pages 29–89. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38570-4. doi: 10.1007/978-3-642-38571-1_2. 50, 51

[247] Dong Wang, Jinlei Jiang, Yongwei Wu, and Guangwen Yang. CampusWare: An Easy-to-Use, Efficient and Portable Grid Middleware for Compute-Intensive Applications. In *2009 Fourth ChinaGrid Annual Conference*, pages 36–43, Yantai, Shandong, 2009. IEEE. ISBN 978-0-7695-3818-1. doi: 10.1109/ChinaGrid.2009.25. 30

[248] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In *2012 IEEE Symposium on Security and Privacy*, pages 365–379, San Francisco, CA, May 2012. IEEE. ISBN 978-1-4673-1244-8. doi: 10.1109/SP.2012.30. 94, 95

[249] Yongjian Wang, D'Ippolito Roberto, Mike Boniface, Depei Qian, Degang Cui, and Jiyun Jiang. Cross-Domain Middlewares Interoperability for Distributed Aircraft Design Optimization. In *2008 IEEE Fourth International Conference on eScience*, pages 485–492, Indianapolis, IN, December 2008. IEEE. ISBN 978-1-4244-3380-3. doi: 10.1109/eScience.2008.176. 33, 34

[250] Raul Sidnei Wazlawick. *Metodologia de pesquisa para ciência da computação*. Elsevier, Rio de Janeiro, RJ, 2009. 22, 178, 181

[251] Yi Wei and M. Brian Blake. Service-Oriented Computing and Cloud Computing: Challenges and Opportunities. *IEEE Internet Computing*, 14(6):72–75, 2010. ISSN 1089-7801. doi: 10.1109/MIC.2010.147. 21, 28, 30, 32

[252] Christof Weinhardt, Arun Anandasivam, Benjamin Blau, Nikolay Borissov, Thomas Meinl, Wibke Michalk, and Jochen Stößer. Cloud Computing - A Classification, Business Models, and Research Directions. *Business & Information Systems Engineering*, 1(5): 391–399, 2009. ISSN 1867-0202. doi: 10.1007/s12599-009-0071-2. 21, 30

[253] Aaron Weiss. Computing in the clouds. *netWorker*, 11(4):16–25, 2007. ISSN 10913556. doi: 10.1145/1327512.1327513. URL `http://dx.doi.org/10.1007/s11576-009-0192-8http://portal.acm.org/citation.cfm?doid=1327512.1327513`. 21, 28, 31

[254] Maurice Wilson. Talis at Nene: an experience in migration in a college library. *Program: electronic library and information systems*, 28(3):239–251, 1994. ISSN 0033-0337. doi: 10.1108/eb047170. 35, 44, 47, 48

[255] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*, pages 1–10, New York, New York, USA, 2014. ACM Press. ISBN 9781450324762. doi: 10.1145/2601248.2601268. 195

[256] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-29043-5. doi: 10.1007/978-3-642-29044-2. 10, 25, 61, 75, 96, 116, 130, 152, 154, 183, 185, 186, 187, 188, 189, 190

[257] Andreas Wolke and Gerhard Meixner. TwoSpot: A Cloud Platform for Scaling Out Web Applications Dynamically. In Elisabetta Nitto and Ramin Yahyapour, editors, *Towards a Service-Based Internet*, volume 6481 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin Heidelberg, Ghent, Belgium, 2010. ISBN 978-3-642-17693-7. doi: 10.1007/978-3-642-17694-4_2. 22, 31, 40, 41

[258] Daniel Wright, Andy Field, and Kamala London. Using Bootstrap Estimation and the Plug-in Principle for Clinical Psychology Data. *Journal of Experimental Psychopathology*, 2(2):252–270, May 2011. ISSN 20438087. doi: 10.5127/jep.013611. 142, 199

[259] Bingheng Yan, Zhongxin Wu, Dongbo Yang, and Depei Qian. Experiences with the EUChinaGrid Project - Implementing Interoperation between gLite and GOS. In *The 2nd IEEE Asia-Pacific Service Computing Conference (APSCC 2007)*, pages 224–231,

Tsukuba Science City, 2007. IEEE. ISBN 0-7695-3051-6. doi: 10.1109/APSCC.2007.15. 30, 33

[260] S.a b Yangui and S.a Tata. PaaS elements for hosting service-based applications. In *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, pages 476–479, Porto, 2012. SciTePress. 22, 33

[261] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010. ISSN 1867-4828. doi: 10.1007/s13174-010-0007-6. 21, 28, 29, 47

[262] Xulin Zhao, Foutse Khomh, and Ying Zou. Improving the Modifiability of the Architecture of Business Applications. In *2011 11th International Conference on Quality Software*, pages 176–185, Madrid, 2011. IEEE. ISBN 978-1-4577-0754-4. doi: 10.1109/QSIC.2011.36. 57, 62

[263] Liang Zhou. CloudFTP: A Case Study of Migrating Traditional Applications to the Cloud. In *2013 Third International Conference on Intelligent System Design and Engineering Applications*, pages 436–440, Hong Kong, January 2013. IEEE. ISBN 978-1-4673-4893-5. doi: 10.1109/ISDEA.2012.108. 34, 46, 47, 48, 49

[264] Minqi Zhou, Rong Zhang, Dadan Zeng, and Weining Qian. Services in the Cloud Computing era: A survey. *2010 4th International Universal Communication Symposium*, 51 (5):40–46, October 2010. ISSN 0937-6429. doi: 10.1109/IUCS.2010.5666772. 21, 28

[265] Yuming Zhou and Hareton Leung. Predicting object-oriented software maintainability using multivariate adaptive regression splines. *Journal of Systems and Software*, 80(8): 1349–1361, August 2007. ISSN 01641212. doi: 10.1016/j.jss.2006.10.049. 192, 195

[266] Zhenyun Zhuang and Yao-Min Chen. Optimizing JMS Performance for Cloud-Based Application Servers. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 828–835, Honolulu, HI, 2012. IEEE. ISBN 978-1-4673-2892-0. doi: 10.1109/CLOUD.2012.136. 60