Factors Affecting Cloud Infra-Service Development Lead Times: A Case Study at ING

Huijgens, Hennie; Greuter, Eric; Brons, Jerry; Doorn, Evert A. van; Papadopoulos, Ioannis; Martinez, Francisco Morales; Aniche, Maurício; Visser, Otto; Deursen, Arie van

DOI
10.1109/ICSE-SEIP.2019.00033

Publication date
2019

Document Version
Submitted manuscript

Published in
Proceedings of the International Conference on Software Engineering (ICSE)

Citation (APA)
Huijgens, H., Greuter, E., Brons, J., Doorn, E. A. V., Papadopoulos, I., Martinez, F. M., Aniche, M., Visser, O., & Deursen, A. V. (2019). Factors Affecting Cloud Infra-Service Development Lead Times: A Case Study at ING. In *Proceedings of the International Conference on Software Engineering (ICSE): Software Engineering in Practice (SEIP)* (pp. 233-242). IEEE. https://doi.org/10.1109/ICSE-SEIP.2019.00033

Delft University of Technology
Software Engineering Research Group
Technical Report Series

Factors Affecting Cloud Infra-Service Development Lead Times: A Case Study at ING

Hennie Huijgens, Eric Greuter, Jerry Brons,
Evert A. van Doorn, Ioannis Papadopoulos,
Francisco Morales Martinez, Maurício Aniche,
Otto Visser, Arie van Deursen

Report TUD-SERG-2018-003

TUD-SERG-2018-003

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
`https://se.ewi.tudelft.nl/tr.html`

For more information about the Software Engineering Research Group:
`https://se.ewi.tudelft.nl/`

Factors Affecting Cloud Infra-Service Development Lead Times: A Case Study at ING

Hennie Huijgens, Eric Greuter, Jerry Brons, Evert A. van Doorn
ING
Amsterdam, The Netherlands
hennie.huijgens,eric.greuter, jerry.brons,evert-jan.van.doorn@ing.com

Ioannis Papadopoulos*, Francisco Morales Martinez*
Delft University of Technology
Delft, The Netherlands
i.papadopoulos-1,f.j.moralesmartinez @student.tudelft.nl

Mauricio Aniche, Otto Visser, Arie van Deursen
Delft University of Technology
Delft, The Netherlands
m.f.aniche,o.w.visser, arie.vandeursen@tudelft.nl

***Abstract*—*Background*: The development of Cloud Infra-Services has shifted over the past decade in the direction of a software code development process, also known as infrastructure as code (IaC). *Objective*: Contemporary continuous delivery settings in industry require fast feedback. As a consequence, companies need metrics that can be used to steer on improvements of time to (internal) market, and to benchmark the performance of their Cloud Infra-Services with peer groups. *Method*:We benchmark Cloud Infra-Services, and explore which factors affect their lead time, within ING. For that purpose we examine a series of 28 Cloud Infra-Services. *Results*: We observe that an initial perception among several stakeholders, that Cloud Infra-Services within ING take longer than those in peer companies, is not confirmed by our benchmark. Development team members identified the time to internal market of Cloud Infra-Services to be affected negatively by the Consumer Ordering Interface (the IPC-portal) and the Orchestration Workflows. This perception is supported by additional metrics. *Conclusions*: We propose that promising ways to reduce lead time include reducing the complexity of the ING environment, by treating Cloud Infra-Services like regular software deliveries and by reducing the dependencies between teams in terms of tooling and collaboration.**

***Index Terms*—Cloud Infra-Services, Infrastructure as Code, Virtual Machine, SaaS, PaaS, IaaS, Continuous Delivery, ING**

I. Introduction

Cloud computing is a widely used 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). This model can be rapidly provisioned and released with minimal management effort or service provider interaction [1] [2]. Cloud environments commonly consist of services, which are increasingly being developed entirely as code. In this study, we apply a software development approach to these deliveries. We focus on Cloud Infra-Services: services that enable the automated deployment of infrastructure [1]. If an organization wants to release Cloud Infra-Services rapidly, it is crucial that it knows which factors affect the time needed to develop these services. However few, if any, studies provide guidance on this subject. In this paper, we explore factors that affect the time to internal market and the development time of services related to infrastructure.

*Work completed during an internship at ING.

A cloud may contain various types of services. These services may include pieces of infrastructure, that users may order in the cloud (e.g. a database, a virtual machine with an OS, a network component). Such services can be developed as code using ways of working like continuous delivery, test-driven development, Dev/Ops and build/deployment automation, in order to automate as many of the parts of their life-cycle as possible [3] [4]. They are generally referred to as *infrastructure as code* (IaC) [5] [6]. IaC services can be divided into three types [1]: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

For *Software as a Service* services, the capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser, or a program interface (e.g. Oracle Database as a Service).

For *Platform as a Service (PaaS)* services, the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider (e.g. Microsoft SQL 2016 server stacks, Linux Redhat server stacks, or GlusterFS patterns).

For *Infrastructure as a Service* services, the capability provided to the consumer is to provision processing, storage, networks and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications (e.g. network components, or private networks).

As limited data was available for IaaS cloud infra services, the focus of our study is an analysis and benchmark of a series of SaaS and PaaS Cloud Infra-Services. To explore the time to market of such Cloud Infra-Services and the factors that affect it, we examine such services developed in the private cloud platform of ING, a large, globally operating bank based in the Netherlands.

A. Background

ING is in the midst of a shift from finance-oriented to engineering-driven company. The infrastructure department of ING - ING Tech Infra - delivers the global digital self-service

IT Infra platforms, to enable the bank to unite and operate as one. For the services that Tech Infra provides, virtualization of environments and infrastructure play a decisive role in providing information to customers and employees.

In recent years, ING implemented a fully automated release engineering pipeline for its software engineering activities. This pipeline facilitates more than 600 teams, that perform more than 2500 deployments to production each month on over 750 different applications. The pipeline is based on the model described by Humble and Farley [7] - and is known within ING as *CDaaS*, an abbreviation of *Continuous Delivery as a Service*. Within CDaaS, ING created two pipelines for their main technology platforms Windows and Linux.

One main goal of CDaaS is to support teams in maximizing the benefits of shared use of tools. The mindset behind CDaaS is to go to production as fast as possible, while maintaining or improving quality, so teams get fast feedback, and know they are on the right track. It forms the core of an ongoing transition within ING towards BizDevOps, a model were software developers, business staff, and operations staff work together in one small, agile team. The idea behind this is that such teams can develop software more quickly, be more responsive to user demand, and ultimately maximize revenue.

ING Tech Infra delivers its infrastructure products through a private cloud platform known as ING Private Cloud (IPC). ING has decided to build its own private cloud, to comply with regulations in the financial sector. Private cloud refers to a model of cloud computing where IT services are provisioned over private IT infrastructure for the dedicated use of a single organization [1].

With IPC, ING controls the global pipeline of its infra-deliveries through four stages: Development, Test, Acceptance, and Production. In this study, we focus on the Cloud Infra-Services that are currently in Production. This means that an engineer in a BizDevOps team can order a Cloud Infra-Service from a web portal known as the IPC portal. By doing so, a part of the cloud infrastructure specifically developed to deploy a Cloud Infra-Service automatically deploys an instance of the service that is ready for use. We explore which factors affect the time to internal market and development time of the full Cloud Infra-Service, including these automated deployment processes.

B. Problem Statement

Because ways of working like continuous delivery and Dev/Ops specifically require short iteration times, we are interested in examining how long it takes for a Cloud Infra-Service to be developed, from the moment a vendor releases it as a product to the moment customers can order it within ING.

Our exploration resolves around the following questions:

RQ1: How does development time of the examined Cloud Infra-Services compare to other companies?

RQ2: What factors affect the time to internal market and development time of Cloud Infra-Services in continuous delivery settings?

RQ3: What actions can be taken to decrease time to internal market and development time of Cloud Infra-Services?

We use converging methods too answer these research questions, and aim to make the following contributions:

1) We propose a lightweight measuring technique of Cloud Infra-Services in a continuous delivery setting, based on a proven model for benchmarking software delivery portfolios.
2) We gather data on 28 deployed Cloud Infra-Services, and map these deliveries on a model for internal and external benchmarking purposes in order to identify good and bad deliveries.
3) We report a set of additional metrics related to usage, complexity and reliability of services once they have been deployed, to explore if they correlate with time to internal market and development time of the Cloud-Infra Service.
4) We survey stakeholders in the Cloud Infra-Service development process, to identify factors that influenced IPC PaaS and SaaS Cloud Infra-Services' internal time to market and development time.

The remainder of this paper is structured as follows. In Section II related work is described. Section III outlines the research design. The results of the study are described in Section IV. We discuss the results in Section V, and finally, in Section VI we make conclusions and outline future work.

II. Related Work

Cloud computing is a paradigm to deliver IT services as computing utilities, which run on data centers. It is enabled by advances in virtualization in computing, storage, and networking [8] [9]. Clouds provide users with services. Among other things, such services can be used to construct highly customized, software-defined environments that can support dynamic and data-driven applications. To the extent that they support deployments of services to consumers, such services can provide infrastructure [10]. Cloud computing and service oriented computing have a number of challenges. For example, Wei and Blake [11] identify maintaining high service availability, providing end-to-end secure solutions, and managing longer-standing service workflows. They also mention opportunities, such as service discovery through federated clouds, rapid service deployment, and agent-mediated ontology generation from co-located information.

To address challenges related to cloud computing, authors have proposed benchmarks at several levels of abstraction. For example, focusing on the deployment process, benchmarks have been proposed for *deployment methods* and *management platforms* for cloud services (e.g. [12] [13] [14] [15] [16]). Focusing on development, Palesandro et al. describe how the Infrastructure as Code (IaC) paradigm is emerging as a key enabler for cloud services, to develop and manage infrastructure configurations. However, the complexity of the infrastructure life-cycle, the diverse resources that infrastructure configurations consist of, and demand for user-customizations complicate application of their approach [17]. More importantly, both methods fail to distinguish *build* and *delivery* phases of infrastructure services.

In other publications benchmarks are explored specifically for Cloud Infra-Services. Scheuner et al. developed a benchmarking approach for IaaS deliveries [12], and introduced Cloud WorkBench (CWB) in [13]. They presented their results of a large-scale cloud evaluation analyzing more than 33,000 measurements in [14]. Bhattacharjee et al. developed CloudCAMP, a Model-driven Generative Approach for Automating Cloud Application Deployment and Management [18]. Additionally, Scheuner and Leitner describe a new execution methodology that combines micro and application benchmarks into a benchmark suite called RMIT Combined [15]. Although more specific, these benchmarks do not distinguish infra services form non-infra services, limiting their usefulness for our current exploration.

To benchmark the performance of SaaS and PaaS Cloud Infra-Services within IPC to a representative set of available data, we chose to use a software development-based model, known as the *Evidence-Based Software Portfolio Management-model* (EBSPM-model) [19] [20]. The EBSPM-model focuses on benchmarking software delivery portfolios. It is built on a repository of more than 500 finalized software deliveries in four different companies (two banking companies, one telecom company, and one billing software company). Using this model allows us to view the entire development cycle of a Cloud Infra-Service, and compare with similar deliveries in other companies on three key metrics.

III. Research Design

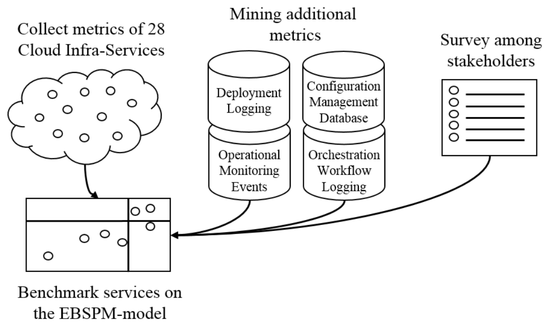
To better understand which factors affect lead time of Cloud Infra-Services, we use an exploratory mixed case study design consisting of the four steps depicted in figure 1. We will first describe our sample, and then discuss each step in turn.

A. Experimental Context

At the time of writing there are 38 Cloud Infra-Services available in Production in the IPC-portal. Most services are based on a vendor product (e.g. Red Hat Enterprise Linux 7 v1.0.8 for the RHEL7 delivery). They are also characterized by a platform (Linux or Windows). Upon deployment, an instance of the Cloud Infra-Service is automatically created in IPC by its cloud infrastructure, and registered as a configuration item in the configuration management database (CMDB). Such instances can have a variety of types (pattern, virtual machine with or without operating system, physical machine) and may have relations with middleware and / or applications as needed.

Within ING, teams are responsible for the delivery of each Cloud Infra-Service. These teams work agile, led by a Product Owner (a person responsible for the business value of the team). Teams usually work in close collaboration with other teams to create a service. We focused on a subset of 28 SaaS and PaaS Cloud Infra-Services that can be ordered directly in the IPC-portal (we excluded IaaS services from our scope due to the limited availability of data). A full overview of the services in scope and in the portal can be found in the technical report [21].

Fig. 1. Overview of the Research Approach.


Collect metrics of 28 Cloud Infra-Services
Mining additional metrics
Survey among stakeholders
Deployment Logging
Configuration Management Database
Operational Monitoring Events
Orchestration Workflow Logging
Benchmark services on the EBSPM-model

B. Collection of Metrics for Cloud Infra-Services

In order to plot each Cloud Infra-Service into the EBSPM model [19] [20], we collected three metrics: (1) lead time, (2) effort (e.g. man hours spent, cost of a delivery), and (3) functional size (the latter being included as a normalizer). We did so by conducting open interviews with the Product Owner for each Cloud Infra-Service, asking them to provide (1) and (2). Point (3) was measured by one of the principal researchers, by counting function points in the IPC portal environment. We counted functional size based on functionality delivered by the IPC portal itself, according to IFPUG guidelines [22].

C. Benchmark Cloud Infra-Services

We plotted the Cloud Infra-Services collected in the former step on the EBSPM-model. The results of this step are (1) a *research repository* with basic metrics of the services over time, and (2) an inventory of *good practice Cloud Infra-Services* (services that performed better than average on both Development Time and Cost) and *bad practice Cloud Infra-Services* (services that performed worse than average on both Development Time and Cost). The resulting plot and metrics will be discussed in the next section.

D. Mining of Additional Metrics

The benchmarking metrics discussed above relate to the time necessary to build cloud *infra* services, services that enable the automated deployment of each cloud service. We sought to explore whether *post-deployment* characteristics of the Cloud Infra-Services, particularly *usage*, *complexity* and *reliability*, could be related to cost, development time and functional size (as used in our benchmarking procedure). Given the exploratory nature of this study, we did not have specific hypotheses with regards to influence of these metrics on the performance of Cloud Infra-Services.

We measured usage as the number of deployments of configuration items with a specific Cloud Infra-Service within

IPC overall and within the past year, and the total amount of configuration items that were active during the past year (configuration items which were decommissioned were included if their time of decommissioning within the past year). Complexity refers to the duration of the deployment workflow of the Cloud Infra-Service, the number of deployment steps needed for that service in the main orchestration layer, and the number of workflow orchestration tools used in deploying the service. Reliability refers to the number of monitoring events registered by ING's automated event monitoring per Cloud Infra-Service, averaged over configuration items.

Because monitoring data proved incomplete, we did not count numbers of events in isolation. Instead, we focused on the impact of events for ING by counting the events per configuration item that were acknowledged by an operator after being generated by an automated monitoring tool, events assigned an incident number, and events assigned a severity number ranging from 0 (least severe) to 5 (most severe). The choice for these metrics was made by project stakeholders, together with subject matter experts on monitoring. The relevance of the metric for ING and availability of data were used as criteria.

To derive the metrics above, we combined data from the *deployment registry* (all configuration items that were deployed within IPC since its launch), the *configuration management database*, the *event monitoring datawarehouse* (registrations of events on configuration items generated by automated monitoring and logging processes) and the *deployment orchestration logging* (all of the workflow steps invoked by the central orchestration layer within ING Infra). We deduplicated entries for configuration items in both registries, and checked assignment of configuration items to Cloud Infra-Services with subject matter experts within ING. We then subsetted monitoring and workflow logging data when appropriate (e.g. to select succesful deploys or events with certain severity). We also used timestamp data to infer whether a configuration item had been active during the time period or not, and to isolate the deployment steps in the logged workflow data. Refer to [21] for a more detailed description of the steps taken, and the R scripts used.

E. Survey Among Cloud Infra-Service Stakeholders

We wanted to measure which factors the members of the teams that develop the Cloud Infra-Services identified as affecting the time to internal market of these deliveries. To that end we conducted a survey, which focused on three parts: the duration of the development of the Cloud Infra-Service, idle time prior to the start of development, and the perceived complexity of a Cloud Infra-Service.

In the survey, we first asked which Cloud Infra-Service the stakeholder was most involved in developing, and what his or her role in the development process was. We then asked about 11 aspects of the development process that could affect internal time to market, as depicted below in Table II. These 11 aspects derived from discussion sessions with Product Owners of a variety of Cloud Infra-Services within IPC, which were aimed at identifying a typology of steps that can generically be said to be taken in the development of IPC Cloud Infra-Services.

Each of the 11 aspects were addressed in a survey question that asks to what extent a respondent agrees with a statement, on a 1 to 5 point Likert-scale (strongly agree - agree - neutral - disagree - strongly disagree - don't know). Each survey question was accompanied by the follow-up question "Can you please explain the choice you made to us?" See the technical report [21] for a detailed overview of the survey questions.

We sent the electronic survey to 275 members of ING Tech Infra squads that were involved in one or more Cloud Infra-Services in scope of this study. We did not offer any reward to increase the participation in the survey. Based on the responses, we calculated several indicators in order to interpret the results of the survey. Note that the first three are measures of the central tendency, CV is a measure of variability.

1) *Percent Agree* or *Top-2-Box*; the percentage respondents that agreed or strongly agreed.
2) *Top-Box*: the percentage respondents that strongly agreed.
3) *Net-Top-2-Box*; the percentage respondents that chose the top 2 bottom responses subtracted from the top-2 top responses.
4) *Coefficient of Variation* (CV); also known as relative standard deviation; the standard deviation divided by the mean. Higher CV-values indicate higher variability.

We also coded the free format text from the surveys to examine whether the provided responses confirmed observations from the survey analysis. We did so using an open card sort [23] with three phases. In the *preparation phase*, we created cards for each survey question commented on by the respondents. In the *execution phase*, cards were sorted into meaningful groups with a descriptive title. Finally, in the *analysis phase*, abstract hierarchies were formed in order to deduce general categories and themes. Our card sort was open, meaning we had no predefined groups; instead, we let the groups emerge and evolve during the sorting process. We applied a number of sub-sequential steps in the card sort. The fifth author tagged the first half of answers. The sixth author tagged the second half. Results were reviewed and discussed in a group discussion with the other authors.

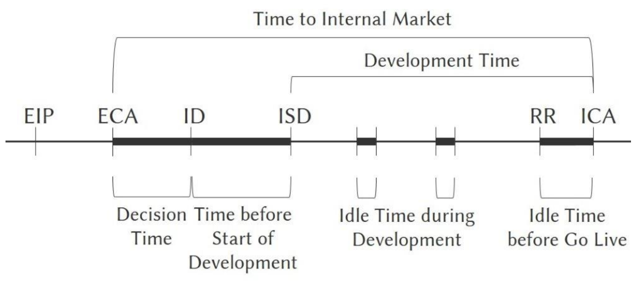
IV. Results

We report results from 1) the analysis of collected Cloud Infra-Services, 2) the benchmarking of the services, 3) the analysis of additional metrics, and 4) the survey performed among stakeholders of the services in scope. A summary of all metrics collected, including the various key moments on the Cloud Infra-Service timeline, is included in the technical report [21].

A. Inventory of Cloud Infra-Services

We recorded the collected data as described in Section III in a repository. Figure 2 gives an overview of applicable time

Fig. 2. Overview of Timelines in Cloud Infra-Services.



Explanation of abbreviations: *EIP* (External Information on Product); the date when the first information of a product is made available by a vendor. *ECA* (External Consumer Availability); the date when a product is made generally available for consumers by a vendor. *ID* (Internal Decision); the date when a decision was made to start developing a Cloud Infra-Service. *ISD* (Internal Start of Development); the date when a Cloud Infra-Service development team put the first user story in the backlog management system into a sprint. *RR* (Ready for Release); the date when a complete productized build was ready according to its Definition of Done. *ICA* (Internal Customer Availability); the date when a Cloud Infra-Service became generally available for internal consumers on the IPC portal.

lines for infra-services within ING, such as Time to Internal Market and Development Time.

TABLE I
TIMELINE STATISTICS

	Dev.Time	Decision Time	Time before SoD
Count	28	12	14
Max	16	26	17
Mean	6.96	5.58	2.64
Median	6	3	1
Min	3	0	0
Standard Deviation	3.45	4.49	7.75

As can be seen in Table I, the Development Time of Cloud Infra-Services varied from 3 to 16 months, with an average duration of 6.96 months. Two types of idle time occur. First, Decision Time - the time between a product being available for consumers and the internal decision taken within ING to start a project - varied from 0 to 26 months, with an average of 5.58 months. Second, Time before Start of Development - the time between the a decision by ING to start a project and the actual start of development - varied from 0 to 17 months, with an average of 2.64 months. The other expected two types of idle time (during development and before go live) could in theory also occur, but interviewees found it difficult to provide accurate information on them.

B. Benchmark Cloud Infra-Services

We mapped the 28 services on the EBSPM-model [19] [20], with Development Time projected on the vertical axis, and cost projected on the horizontal axis, as shown in Figure 3. Revisiting RQ1 - How does lead time of the examined Cloud Infra-Services compare to other companies? - shows that on average the subset of 28 services performed 17% better on duration and 41% better on cost than the average of the EBSPM-repository, based on a repository of 500 finalized software deliveries in four comparable companies.

Observation 1: Our study does not confirm the initial perception among many ING-stakeholders that Cloud Infra-Services within ING take more time than those in peer companies, instead ING services show on average a 17% shorter Development Time than software deliveries in the EBSPM-repository.

The colors of the different Cloud Infra-Services in Figure 3 indicate that services with a longer-than-average Development Time (indicated on the vertical axis) also tend to have a longer Time to Internal Market (indicated by the color range from blue at the top to red at the bottom). Longer Decision Time and Time before Start of Development seem to go together with longer Development Time.

Observation 2: Taken together, average Decision Time and average Time before Start of Development exceed average Development Time. This suggests that examining the preliminary stage of service development in more detail may yield improvements in lead time.

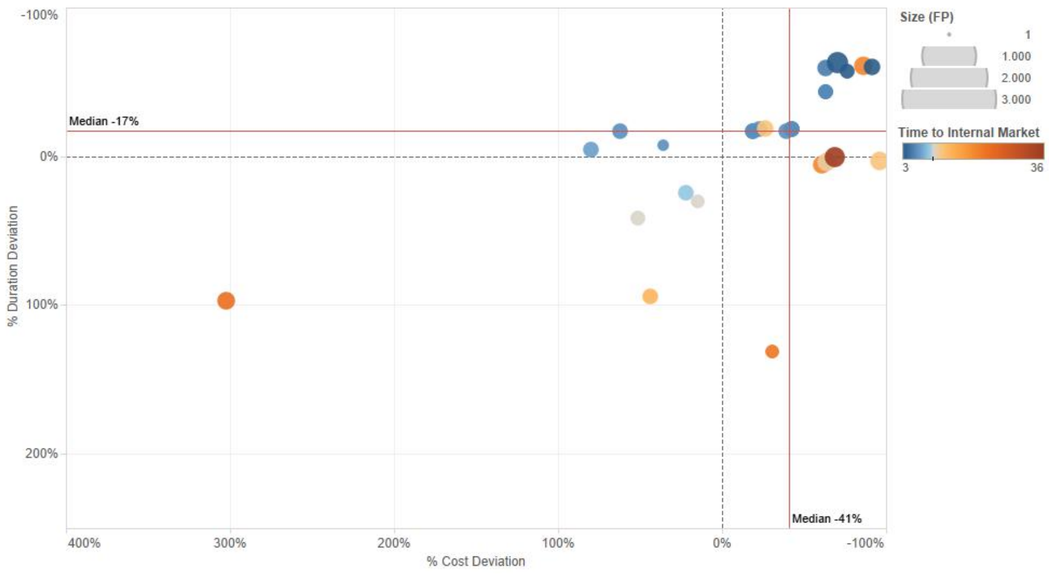
C. Mining of Additional Metrics

To assess what factors affect the lead time of Cloud Infra-Services in continuous delivery settings (RQ2), we derived metrics for usage, complexity, and reliability of Cloud Infra-Services in the context of IPC. We explored whether correlations exist between usage, complexity, reliability, and benchmarking metrics time to internal market, cost and functional size. For each of these categories of metrics, descriptive statistics are included in the technical report [21].

1) Correlations between metrics: Given our relatively small sample size (26 data points), calculation of a correlation matrix may suffer from low statistical power, which may lead to inflated correlation coefficients [25]. However, since our research is exploratory in nature, we report the correlations as possible directions to explore. Due to redundancy of indicators for usage and complexity, we report only the most relevant dimensions here. A more extensive correlation table is included in the technical report [21]. The various reliability metrics correlate highly with each other. Cloud Infra-Services with a higher average of acknowledged events per configuration item also have a higher average of incidents per configuration item and a higher number of severity 5 events per production configuration item, all $r(26) = .59$, all p-values $< .05$. This pattern of results matched what one would expect based on monitoring, where more sever events are formally recognized more often.

Two correlations between metrics for usage, reliability and complexity are noteworthy. First, the more configuration items were active during the past year, the more components were included in the deployment workflow of the main orchestration tool, $r(26) = .69$, p-value $= .002$. This shows that more frequently used Cloud Infra-Services incorporate more tools

Fig. 3. The Cloud Infra-Services in scope of this exploratory study mapped on the EBSPM-model.



The EBSPM-model is based on a subset of more than 500 finalized software deliveries from five different companies. The figure above shows only the 28 Cloud Infra-Services in scope of this study. Each service is shown as a circle. The larger the circle, the larger the service is (in functional size). Color indicates a longer Time to Internal Market (the more red, the longer; this varies from 3 to 36 months). The position of each service in the matrix represents the Cost and Development Time deviation of the service relative to the benchmark, expressed as percentages. The horizontal and vertical 0%-lines represent zero deviation, i.e. services that are exactly consistent with the benchmark. A service at (0%, 0%) would be one that behaves exactly in accordance with the benchmark; a service at (-100%, -100%) would cost nothing and be ready immediately; and a service at (+100%, +100%) would be twice as expensive and take twice as long as expected from the benchmark.

in their workflows. Second, the longer a deployment of a Cloud Infra-Service in the main orchestration tool took on average, the more incidents were registered on configuration items for that Cloud Infra-Service, r = .71, p-value = .002. This shows that Cloud Infra-Services with, on average, longer deployments have more post-deployment incidents registered. We observed no correlations between the benchmarking metrics and usage, complexity and reliability.

Observation 3: Our study shows that more popular Cloud Infra-Services have workflows that consist of a greater number of components, and Cloud Infra-Services with a longer deployment time register more incidents per configuration instance, on average.

D. Survey Results

Our survey on factors that affected development time of Cloud Infra-Services was active during two weeks. During this time, 10.2% of the 275 who were invited to participate responded, yielding 28 completed questionnaires. The respondents include 22 Cloud Infra-Service Engineers (78.6%), 5 Product Owners (17.9%), and 1 Chapter Lead (3.6%) .

The respondents indicated their level of agreement or disagreement towards 11 statements (questions Q03 through Q13 in the survey). They did so on 1 to 5 point Likert-scales, or resorted to an *"I don't know"* option if they were unsure whether the aspect mentioned in the question affected the time to delivery of their infra cloud service. See Table II for descriptive statistics and bar-charts depicting the spread of scores for each survey question.

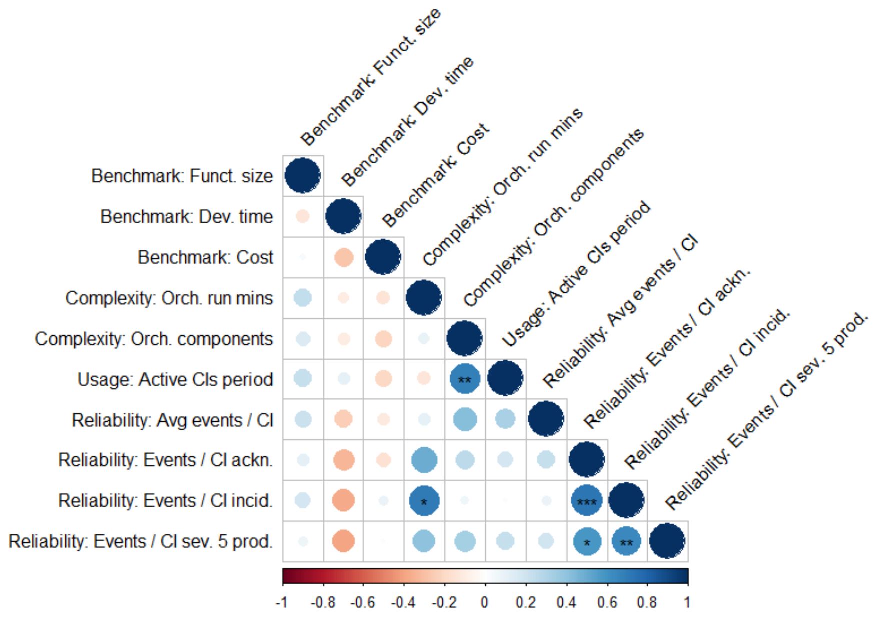
1) Consumer Ordering Interface and Orchestration Workflows: A relatively high number of respondents agreed with the statements that the Consumer Ordering Interface (Q03) and Orchestration Workflows (Q04) were obstacles for the delivery of the Cloud Infra-Service they worked on (67 and 68 percent agreement, respectively). As one of the respondents put it: "The portal is working very slow and it is annoying" [P07]. On the other hand, Service Delivery aspects (Q09) were considered the least hindering of all measured aspects (13 percent agreement).

Observation 4: The Consumer Ordering Interface (the IPC-portal) and the Orchestration Workflows were seen by a large percentage of respondents as negatively affecting time to internal market.

2) Second Day Operations: Although answers of both Q06 and Q07 are scattered, respondents say that second day operations are time consuming in general, and a lack of unified CMDB models is perceived as an obstacle: "The optional software capabilities should be part of the CDaaS (application) workflow and not of the Cloud capabilities" [P05].

3) Security, Risk, Compliance, and Governance: The process with regard to risk, security and compliance (Q08) is perceived as complex by many respondents: "It takes weeks to

Fig. 4. Correlation Matrix of selected Benchmarking, Usage, Complexity and Reliability Metrics.



This correlation matrix depicts correlations between the most important benchmarking, usage, complexity, and reliability metrics for each Cloud Infra-Service in our study. The size of the circle represents the magnitude of the correlation. The *, ** and *** superscripts represent *p*-values associated with the correlation, of > .05, > .01 and > .005, respectively. The color of the circle represents the direction (blue for positive, red for negative) of the correlation. The correlations depicted are a subset of a correlation matrix containing all metrics. This matrix was corrected for multiple comparisons using a Benjamini Hochberg correction [24]. See the technical report [21] for more details.

set up security scans, pentests, get approval for documents..." [P11]. Governance related aspects (e.g. decision-making, rules & regulations) are not perceived as impediments, as indicated by a Net-Top-2-Box score of 0%.

Observation 5: The process with regard to risk, security and compliance is perceived as complex by a large number of respondents.

4) Service Delivery: Documentation, service component description, service specification, and training are not perceived as obstacles, as indicated by a Net-Top-2-Box score of -52%.

5) Finance and Governance: Financial (Q10) and Governance (Q13) related aspects received more *"I don't know"* answers than the other statements (10 each, in total). It might be the case that respondents are less aware or familiar with these aspects, given their less-technological nature.

V. Discussion

In this study we identify five topics that we see as relevant to establishing the time to internal market of Cloud Infra-Services: data quality considerations, using appropriate benchmarks for projects, reduction of decision time, reduction of dependencies between teams and tools, and assessing the implementation of security measures. We discuss these topics below for each of our research questions, giving attention to considerations with regards to validity where necessary.

A. How does lead time of the examined Cloud Infra-Services compare to other companies? (RQ1)

Our results indicate that with regards to development time, the Cloud Infra-Services within IPC perform 17% better than other deliveries in the benchmark repository. This shows that in terms of cost and development time, ING is doing well. Though ING internal customers may experience the development process as slow, our benchmark suggests other organizations with projects of comparable size do about as well, if not worse in terms of development time.

We based our benchmarks on interview data, which introduces several issues with regards to validity. First, we relied on the memory of the Product Owner to obtain timepoints and estimates of cost and effort. These data were not adequately administered for several projects, and team composition (including the Product Owner role) may have changed during or after development. Our metrics should therefore be seen as rough estimates. Additionally, interviews with Product Owners were conducted by the investigators. This may have affected interview results. We attempted to minimize such effects by asking for factual information and followed a standardized protocol for the interview.

TABLE II
OVERVIEW OF THE SURVEY ANALYSIS.

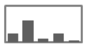
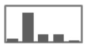
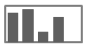
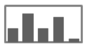
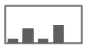
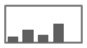
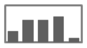
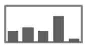
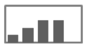
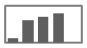
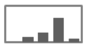
Interview Question	Likert Distribution	Number of Respondents	Percent Agree	Top-Box	Net-Top-2-Box	CV
Q4. Orchestration Workflows related aspects (e.g. Workflows/Automation for Virtual Machine, Operating System, Network, System Accounts, Storage, Configuration Registration) hindered the delivery of <Cloud Infra-Service choice>.		25	68%	20%	44%	31%
Q3. Consumer ordering interface related aspects (e.g. setting up IPC portal to consume new service) hindered the delivery of <Cloud Infra-Service choice>.		27	67%	7%	48%	27%
Q11. Team dynamics related aspects (e.g. dependencies on other teams, cultural differences, many team changes, age of teams, difference in expertise) hindered the delivery of <Cloud Infra-Service choice>.		27	63%	30%	37%	31%
Q12. Service Verification and Testing related aspects (e.g. Optimization, Bug Fixing, Test Resources, Test Automation) hindered the delivery of <Cloud Infra-Service choice>.		24	50%	17%	17%	36%
Q7. Operations related aspects (e.g. Monitoring, Configuration Scanning, CMDB Model) hindered the delivery of <Cloud Infra-Service choice>.		22	45%	9%	0%	35%
Q6. Second Day Operations related aspects (e.g. Install optional software, SelfService capabilities) hindered the delivery of <Cloud Infra-Service choice>.		24	42%	13%	0%	35%
Q8. Security, Risk & Compliance related aspects (e.g. OSG, BIA, Risk Assessment, SEM-I, TSCM-I, Vulnerability Scanning, Penetration Testing, Certificate Management) hindered the delivery of <Cloud Infra-Service choice>.		23	39%	13%	4%	36%
Q10. Financial related aspects (e.g. Procurement, License Metering, Pricing & Charging) hindered the delivery of <infra-delivery choice>.		18	39%	17%	-6%	40%
Q13. Governance related aspects (e.g. Decision-making, Rules & Regulations) hindered the delivery of <Cloud Infra-Service choice>.		18	33%	11%	0%	32%
Q5. Stack Definition related aspects (e.g. Capabilities for Backup, APIs, Agents) hindered the delivery of <Cloud Infra-Service choice>.		22	32%	5%	-5%	30%
Q9. Service Delivery related aspects (e.g. Documentation, Service Component Description, Service Description, Service Specification, Training + Instruction Movies) hindered the delivery of <infra-delivery choice>.		23	13%	0%	-52%	34%

Table sorted on percentage agreed. Column 'Likert Distribution' shows a graph of the distribution on a 1-5 point Likert scale for each question with from left to right the values 'Strongly Agree', 'Agree', 'Neutral', 'Disagree', and 'Strongly Disagree'. See the Technical Report for an extended overview of the survey setup and the survey questions.

TABLE III
MOST MENTIONED CODES PER QUESTION

Question	Code Description
Q3	Setting up IPC portal to consume the new service (9)
	Dependencies on other teams (5)
	Issues with orchestration tools (4)
Q4	Issues with orchestration tools (5)
	Dependencies on other teams (3)
	Complexity of the infra delivery (3)
Q5	Backup capabilities / miss-alignments in requirements (4)
Q6	Second day operations are time consuming (5)
	Complexity of the infra delivery (4)
Q7	Not unified CMDB Models and other issues with them (8)
Q8	The process of risk and security is too complex (9)
Q9	Documentation issues (3)
Q10	Pricing/Charging/License (6)
Q11	Dependencies on other teams (12)
Q12	Bug fixing and testing (10)
	Test Resources (3)
Q13	Decision making (4)
Q14	Pricing/Charging/Licence (4)

B. What factors affect the lead time of Cloud Infra-Services in continuous delivery settings? (RQ2)

Reviewing our benchmarking data, we saw that a relatively long period of decision time precedes the decision to start developing a solution. On average, decision time spans half a year, with the most extreme case recorded spanning over two years. Development time is equally long but has a smaller spread, suggesting that there is value in examining the decision making process in more detail.

Although we found no significant correlations with deployment time, cost, or functional size, our additional metrics did show that the more a Cloud Infra-Service was used over the past year, the more complex it gets. In itself, this does not say much about development time. However, Cloud Infra-Services with a longer deployment duration (an indicator of complexity) had more incidents occur per CI over the past year. Taken together with the first finding, this suggests an increase in the complexity of Cloud Infra-Services may lead to less reliable Cloud Infra-Services after deployment.

A possible explanation for this problem is that an eventual larger number of configuration items means a greater (anticipated) demand for custom functionality, which complicates Cloud Infra-Service development. Such a complexity-based explanation matches the results of the survey, in which problems with workflow tools are prominently mentioned as factors that impede development time. A second prominent factor mentioned in the survey is collaboration between teams; apparently increased dependencies in tooling go together with increased dependencies in collaboration. In an organization that aims to implement infrastructure as code, what effects such dependencies have seems like an important topic to investigate.

As we have seen in our discussion of RQ1, our benchmarking data suffers from non-response by product owners. With regards to our data mining efforts, we were unable to conclusively verify completeness of data for each system we mined. We could not tie deployments, event monitoring or orchestration logging to specific configuration items for Cassandra Keyspace or Oracle DBaaS resulting in missing data for these infra deliveries. In addition, our sample size was small, making any conclusions based on correlations tentative at best. Moreover, we learnt monitoring is optional for certain classes of configuration item, meaning our event data is likely incomplete. We were also not able to conclusively verify whether monitoring for all Cloud Infra-Services was stored in the datawarehouse we mined. Severity categories do not seem to be used systematically for monitoring. For these reasons, we resolved not to report counts of monitored events without any classification of organizational relevance. Finally, our monitoring data only went back one year, while some infra deliveries were more mature than others. In sum, although we are confident that the conclusions we draw are optimal given the available data, a more systematic approach of data storage with regards to Cloud Infra-Services in both development and deployment would greatly increase ING's ability to draw conclusions regarding the IPC environment.

The survey we sent out suffers from two main issues. First, we sought a representative, stratified sample of ING engineers who worked on the Cloud Infra-Services in our study. This was complicated by staffing changes within teams, leading us to e-mail all employees of the Infra department at ING. A list of who worked on which delivery when would have made it easier to target a representative, stratified sample. Additionally, we built our survey to gather information on categories, of which team leaders indicated they were process steps in developing an infra delivery. The extent to which these categories where adequately understood by our respondents may vary from category to category. We were unable to verify the extent to which this was the case.

C. What actions can be taken to decrease lead time of Cloud Infra-Services? (RQ3)

Our results are largely specific to IPC, and do not generalize well to other environments within or outside ING. Yet, based on our answers to RQ1 and RQ2, we identify four general take-away messages that may be of general benefit in reducing time-to-market and development time of Cloud Infra-Services:

1) Reduce the complexity of the environment by treating Cloud Infra-Services just like regular software deliveries; e.g. make the use of standardized, automated delivery pipelines (such as CDaaS) mandatory.
2) Do follow-up research into the possibilities to reduce the dependencies of other teams (e.g.: security, workflow orchestration), since this is mentioned by many stakeholder as the biggest obstacle for time-to-market.
3) Ensure good process data quality as a precondition for well-informed decision-making; make the use of a standardized backlog management tool, mandatory from the start of a service (e.g. the creation of an epic) and beyond, and formally track decision moments.
4) Examine the decision-making process more closely; the greatest impact on the time-to-market of Cloud Infra-Services can be realized in the decision-making phase and the period prior to the start of the development.

D. Threats to Validity

Like many applied researchers, we have had to sacrifice experimental control for studying an in vivo phenomenon. In doing so, several factors impacted the validity of our results. We have already discussed several points related to construct validity and internal validity above, in summarizing answers to our research questions.

1) External Validity: The results of this study are based on the current situation with ING Infra. Because of the complexity of the environment and the relatively low levels of standardization in processes and tooling, conclusions from the current study have limited external validity. At the same time, this study yields a number of concepts that were shown to be related to the time of internal deployments. These can be mapped onto other organizations with cloud-based infra services.

2) Study Reliability: As a general note, the infra deliveries we examined were developed over a period of years. This development period spanned several large organizational changes and efforts at restructuring, efforts which were ongoing at the time of this study. Teams changed, with members being reassigned or leaving, and the structure of the infra environment changed. Additionally, changes in the various data sources (particularly event data stored in the monitoring datawarehouse), and the necessity for stakeholder management in a project of this scope make it difficult to repeat this process exactly. However, by scripting our analyses and making them repeatable and documenting all our efforts in detail in the technical report, we have made every effort to enable others to replicate the steps we followed.

VI. Conclusions

We performed an exploratory case study on 28 Paas and SaaS Cloud Infra-Services deployed at ING Tech Infra, in order to examine how time-to-market of such services can be shortened. We benchmarked 28 Cloud Infra-Services with peer group software deliveries, mining additional metrics from four data sources, and from a survey among stakeholders. Based on these, we propose that time to internal market may benefit from reducing the complexity within which development teams operate, both in terms of tools and dependencies between teams, from a more detailed consideration of the time necessary to reach a decision to start developing, and from more structural registration of Cloud Infra-Service related data.

A. Directions for Future Research

We see this study as offering several interesting directions for future work. First, the software development-based perspective we applied in benchmarking IPC PaaS and SaaS Cloud Infra-Services provides a straightforward way of quantifying the full lead time of an automatically deployed cloud service. We aim to incorporate a more diverse range of metrics into this model in the future, including data on IaaS components, agile team performance, decision making, and idle time. This will lead to a more fine-grained model, that should be generically applicable across organizations.

Second, we see merit in exploring the differences between the development of applications and infrastructure components. In this study, we have assumed that function points are a useful proxy for the functionality of a service. However, Cloud Infra-Services can involve a more dependencies between cloud infrastructure components than applications normally have. Such dependencies may not be countable as function points. Future examination could test an adapted version of our benchmarking model, in which functionality indicators are matched to Cloud Infra-Service complexity.

Finally, we have conducted this study in a banking environment. Such environments can be expected to have regulations that go beyond those in other sectors. We hope to use our benchmarking model for Cloud Infra-Services in other sectors, so as to provide a standardized comparison within a more homogeneous population. This will enable more confident conclusions with regards to the performance of Cloud Infra-Service development processes.

Our study provides an exploration into the development of the infrastructure of the ING Private Cloud. We have seen that the development speed of PaaS and SaaS Cloud Infra-Services is on par with a sample of other software deliveries. We have also identified several promising directions that ING can explore to further accelerate the time needed to go from vendor release to customer ready Cloud Infra-Service. We hope our findings will help build the better clouds of tomorrow.

Acknowledgments

The authors would like to thank ING Tech Infra and its employees for their willing contributions to this project.

References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *NIST Special Publication 800-145, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930*, 2011.

[2] OpenGroup, "The Open Group Cloud Ecosystem Reference Model - The Cloud Ecosystem Reference Model," 2018. [Online]. Available: http://www.opengroup.org/

[3] D. Spinellis, "Don't install software by hand," *IEEE Software*, vol. 29, no. 4, pp. 86–87, July 2012.

[4] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, no. 3, pp. 94–100, May 2016.

[5] M. Hüttermann, *Infrastructure as Code*. Berkeley, CA: Apress, 2012.

[6] A. Wittig and M. Wittig, *Amazon Web Services in Action*. Manning Press, 2016.

[7] J. Humble and D. Farley, *Continuous Delivery, reliable software releases through build, test and deployment automation*. Addison-Wesley, 2010.

[8] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, November 2013.

[9] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in *2008 10th IEEE International Conference on High Performance Computing and Communications*, Sept 2008, pp. 5–13.

[10] M. Abdelbaky, J. Diaz-Montes, M. Unuvar, M. Romanus, I. Rodero, M. Steinder, and M. Parashar, "Enabling Distributed Software-Defined Environments Using Dynamic Infrastructure Service Composition," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2017, pp. 274–283.

[11] Y. Wei and M. B. Blake, "Service-Oriented Computing and Cloud Computing: Challenges and Opportunities," *IEEE Internet Computing*, vol. 14, no. 6, pp. 72–75, Nov 2010.

[12] J. Scheuner, J. Cito, P. Leitner, and H. Gall, "Cloud workbench: Benchmarking iaas providers based on infrastructure-as-code," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15 Companion. New York, NY, USA: ACM, 2015, pp. 239–242.

[13] J. Scheuner, P. Leitner, J. Cito, and H. Gall, "Cloud work bench – infrastructure-as-code based cloud benchmarking," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, Dec 2014, pp. 246–253.

[14] P. Leitner and J. Cito, "Patterns in the Chaos&Mdash;A Study of Performance Variation and Predictability in Public IaaS Clouds," *ACM Trans. Internet Technol.*, vol. 16, no. 3, pp. 15:1–15:23, Apr. 2016.

[15] J. Scheuner and P. Leitner, "A Cloud Benchmark Suite Combining Micro and Applications Benchmarks," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: ACM, 2018, pp. 161–166.

[16] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun, "Benchmarking in the Cloud: What It Should, Can, and Cannot Be," in *Selected Topics in Performance Evaluation and Benchmarking*, R. Nambiar and M. Poess, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 173–188.

[17] A. Palesandro, M. Lacoste, N. Bennani, C. Ghedira-Guegan, and D. Bourge, "Mantus: Putting Aspects to Work for Flexible Multi-Cloud Deployment," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, June 2017, pp. 656–663.

[18] A. Bhattacharjee, Y. Barve, A. Gokhale, and T. Kuroda, "Technical Report - CloudCAMP: A Model-driven Generative Approach for Automating Cloud Application Deployment and Management," 2018.

[19] H. Huijgens, A. van Deursen, and R. van Solingen, "The effects of perceived value and stakeholder satisfaction on software project impact," *Information and Software Technology*, vol. 89, pp. 19 – 36, 2017.

[20] H. Huijgens, R. van Solingen, and A. van Deursen, "How to Build a Good Practice Software Project Portfolio?" in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 64–73.

[21] H. Huijgens, E. Greuter, J. Brons, E. A. van Doorn, I. Papadopoulos, F. M. Martinez, M. Aniche, O. Visser, and A. van Deursen, "TUD-SERG-2018-003 - Factors Affecting Cloud Infra-Services Development Lead Times: A Case Study at ING," 2018. [Online]. Available: https://se.ewi.tudelft.nl/tr.html

[22] IFPUG, "IFPUG FSM Method: ISO/IEC 20926 - Software and systems engineering - Software measurement - IFPUG functional size measurement method," 2009.

[23] S. Fincher and J. Teneberg, "Making sense of card sorting data," *Expert Systems*, vol. 22, no. 3, pp. 89–93, 2005.

[24] "Controlling the false discovery rate: A practical and powerful approach to multiple testing." *Journal of the Royal Statistical Society Series B*, vol. 57, no. 1, pp. 289–300, 1995.

[25] T. Dybå, V. By Kampenes, and D. I. K. Sjøberg, "A systematic review of statistical power in software engineering experiments," *Information and Software Technology*, vol. 48, pp. 745–755, 2006.

Technical Report

This technical report contains methodological and statistical supplements for the paper 'Factors affecting Cloud Infra-Service development lead times: A case study at ING'. The report is organized into three sections, corresponding to the three main topics of analysis. In order, benchmarking, mining of additional metrics and survey will be discussed.

VII. Benchmarking

For the benchmarking part of this study, we report an overview of the Cloud Infra-Services included in this study. We then provide an overview of descriptive data for the timeline measurements, as they were measured in our interviews.

A. *Overview of Infra Cloud Services*

TABLE IV
Infra deliveries contained in study

Cloud Infra-Service
Apache Web Server
Cassandra Keyspace
Datalake Datawarehouse
Datalake Hadoop
Datalake Landing Zone
GlusterFS
IBM InfoSphere
JBoss
JBoss(2)
Linux Developer Workstation
Microsoft SQL Server 2016
Microsoft SQL Analysis Server
Microsoft SQL
Docker
Oracle
Oracle DataGuard
Oracle DBaaS
RabbitMQ
Redis
Red Hat Enterprise Linux Atomic
Red Hat Enterprise Linux
Red Hat Enterprise Linux (2)
Tomcat
Microsoft Windows Citrix 2012
Microsoft Windows Native
Microsoft Windows Robotics
Microsoft Windows Server 2012 DotNet

B. Summary of benchmarking metrics

TABLE V
OVERVIEW OF MEASUREMENTS AND METRICS INCLUDED IN THE EXPLORATORY STUDY

Metric	Source	Type	Metrics Definition
External Information on Product (EIP)	Vendor website	Date	Date when the first information of a product is made available by a vendor.
External Beta Availability (EBA)	Vendor website	Date	Date when a beta version of a product is made general available for consumers by a vendor.
External Customer Availability (ECA)	Vendor website	Date	Date when a product is made general available for consumers by a vendor.
External End of Support (EES)	Vendor website	Date	Date when a product is not supported by a vendor anymore, including extended support, excluding third party support.
Internal Information on Product (IIP)	EBA	Date	Date when the organization knows about the upcoming product. We assume that we cannot properly measure this metric, therefore we approximate it with EBA as a replacement.
Internal Decision (ID)	Product Owner	Date	Date when a decision was made to start developing an internal product. Proxy: date of the decision in the QBR.
Internal Customer Development (ICD)	Product Owner	Date	Date when an internal customer started developing a dedicated version of an internal product (ask the product owner).
Internal Start of Development (ISD)	ServiceNow	Date	Date when the first user story in ServiceNow related to an internal product was put in a sprint.
Internal Customer Availability (ICA)	Infra Portal	Date	Date when a product is made general available for internal consumers on the Infra portal.
Internal End of Support (IES)	Product Owner	Date	Date when an internal product is not supported anymore by ING Tech Infra.
Time to Internal Market	(ICA - ECA)	Months	The Internal Customer Availability minus External Customer Availability; expressed in months: ((ICA - ECA)/30.43056).
Development Time	(ICA - ISD)	Months	The Internal Customer Availability minus Internal Start of Development; expressed in months: ((ICA - ISD)/30.43056).
Idle Time	(ISD - ID)	Months	The Internal Start of Development minus Internal Decision; expressed in months: ((ISD - ID)/30.43056).
Decision Time	(ID - ECA)	Months	The Internal Decision minus External Customer Availability; expressed in months: ((ID - ECA)/30.43056).
Story Points Delivered	ServiceNow	Ratio	The Number of Story Points delivered in a sprint; expressed in a ratio: ?
Effort		Days	Effort spent to develop a Cloud Infra-Service; as delivered by the Product Owner of a specific service
Cost		Euros	Actual Cost of a Cloud Infra-Service based on Effort * 94 euro.
Functional Size		FPs	Functional size based on the IPC Portal web-functionality, according to IFPUG FSM Method [22]
Complexity - Deployment Time	Workflow Logging	Duration	Average orchestration workflow deployment time in minutes.
Complexity - Number of Deploys	Workflow Logging	Number	Count of orchestration workflow deployments registered.
Complexity - Number of Workflow Steps	Workflow Logging	Number	Average number of workflow steps in orchestration deployment workflow.
Complexity - Number of Workflow Tools	Workflow Logging	Number	Count of orchestration tools in orchestration deployment workflow.
Usage - Overall Deploys IPC	CMDB	Number	Count of all deployments registered within IPC.
Usage - Deploys IPC Past Year	CMDB	Number	Count of all deployments registered within IPC during past year.
Usage - Active CIs IPC Past Year	CMDB	Number	Count of active CIs within IPC during last year.
Events - CIs With Events Past Year	Event Bus	Number	Count of CIs with monitored events during past year.
Events - Average Events per CI	Event Bus	Number	Average number of events per CI during past year.
Events - Average Events per CI Severity 0	Event Bus	Number	Average number of events per CI over past year with severity 0.
Events - Average Events per CI Severity 2	Event Bus	Number	Average number of events per CI over past year with severity 2.
Events - Average Events per CI Severity 3	Event Bus	Number	Average number of events per CI over past year with severity 3.
Events - Average Events per CI Severity 4	Event Bus	Number	Average number of events per CI over past year with severity 4.
Events - Average Events per CI Severity 5	Event Bus	Number	Average number of events per CI over past year with severity 5.
Events - Average Acknowledged Events per CI	Event Bus	Number	Average number of acknowledged events per CI over past year.
Events - Average Incidents per CI	Event Bus	Number	Average number of events with incident number per CI over past year.
Events - Average Events per Production CI with	Event Bus	Number	Average number of events per production CI over past

VIII. Mining of Additional Metrics

We used various data sources to mine for additional metrics. We pre-processed and cleaned these data sources, combined them into datasets suitable for analyses, and then computed metrics for each of the cloud infra services in our study. We then aggregated over these cloud infra deliveries in order to report statistics across them in our paper. This process translates to four sections of extra information included in this technical report. First, we will provide a detailed overview of the steps that were taken in data preparation and processing. Then, we share an overview of the relations between all metrics we mined, in a correlation matrix and a scatter matrix. Unfortunately, we are not able to share the datasets used, as they are proprietary to ING. We can provide parts of the scripts we used, to provide as much detail concerning our analyses as possible. Finally, we include an overview of the ING Private Cloud environment level usage, complexity and reliability metrics (i.e. the tables underlying the main conclusions drawn in the paper).

The starting point for our analysis was an overview of all configuration items deployed within ING Private Cloud, that could uniquely be related to registry entries in the configuration management database. Several configuration items were associated with multiple entries (34 in our deployment data and 39 in the ING Private Cloud configuration management database). We removed such duplicates by keeping the most recent entry for each instance. Additionally, a number of configuration items had multiple distinct infrastructure components associated to them (e.g. a virtual machine with an operating system and a middleware component). We resolved to collapse all relevant component information onto one line for each configuration item, thus creating a dataset consisting of unique configuration items.

Next, we resolved any ambiguity in infra delivery labels through discussion with subject matter experts within ING, resulting in clear labels for each of the deliveries. We subsequently merged the deployment registry data to the configuration management database containing registrations of all successful ING Private Cloud deployments. The resulting table

Interestingly, 267 of these deploys could be matched to our event monitoring data based on server numbers, suggesting they were active while not being registered in the ING Private Cloud configuration management database. Although establishing the reasons for this strange state of affairs goes well beyond the scope of this paper, we do need to note that we decided to exclude these cases due to data from the ING Private Cloud configuration management database registry missing. Based on the component type from the deployment data, we could see that several cloud infra services missed small numbers of configuration items as a result (specifically: Microsoft Citrix missed 3 Configuration Items, NGINX Load Balancer missed 4, Microsoft Robotics missed 3). We were also able to use the merged table to derive a timestamp for each deploy, representing the date the first deploy of a cloud infra delivery was registered in configuration management database.

Next, we merged our overview of ING Private Cloud infrastructure deliveries with the event data contained in configuration item events. This enabled us to quantify reliability by deriving counts and averages of configuration item and events per configuration item, distributed over various categories when appropriate.

To quantify usage, we excluded all configuration items that were deployed into the ING Private Cloud tenants for development of quality assurance (as we are interested in deploys by internal customers, rather than development teams).

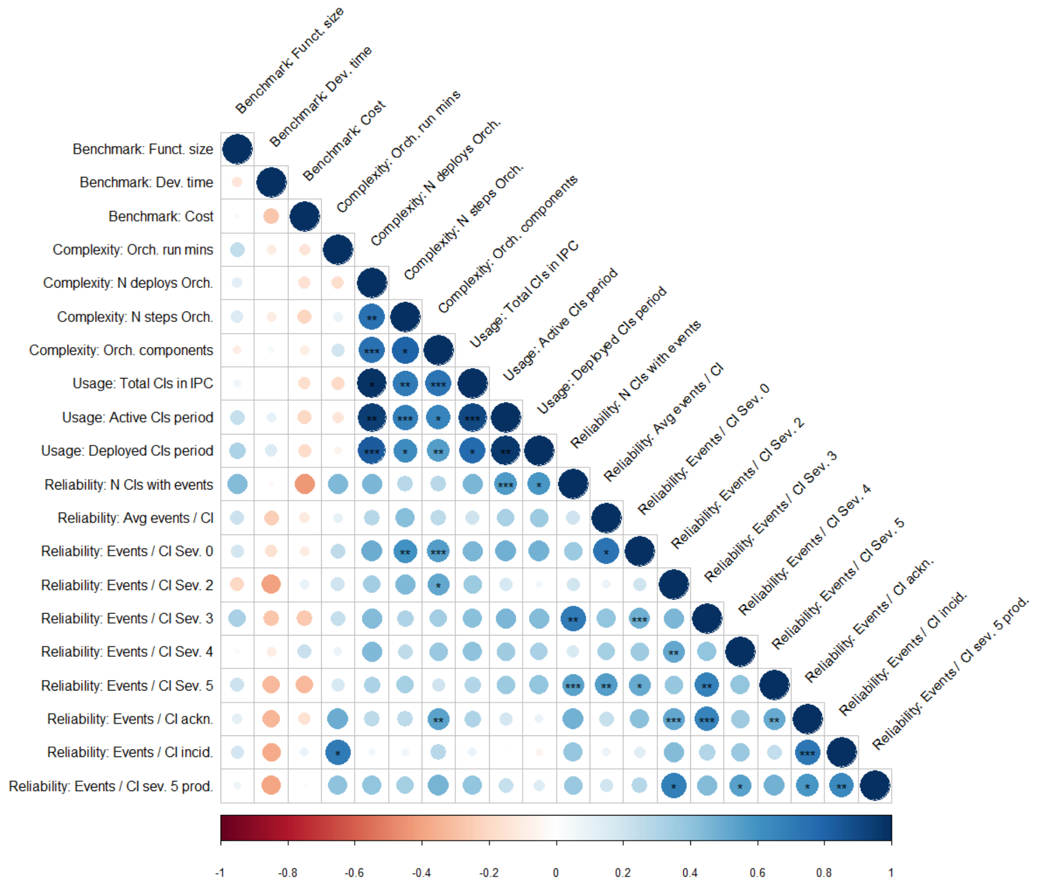
We also examined the number of configuration items which were active during the last year. To do so, we calculated a decommissioning date based on a timestamp recording the date at which retired servers received their last update (which, in this case, is always the moment at which it was retired according to domain experts). We then counted all servers which were retired after 31-08-2017, or which were still active.

Finally, we merged the overview of ING Private Cloud services with our orchestration logging. We filtered the logging information to represent only successful deploys, and used the R bupaR process mining package to construct process maps for each infra delivery. We derived common process steps by selecting the process steps that occurred in more than 50 percent of the deployments for each delivery.

Because the resulting process maps included data for both deployment and decommissioning of an infra delivery, we identified cutoff points per delivery to isolate the deployment steps. We filtered the process logs on these steps, and calculated their average duration. We then counted the number of steps in each infra delivery's deployment process, and counted the number of workflow orchestration components within each workflow. To get an overview of data related to each infra delivery, we combined information related to the benchmarking process and metrics related to usage, complexity and reliability in a central data repository.

A. Extensive correlation matrix

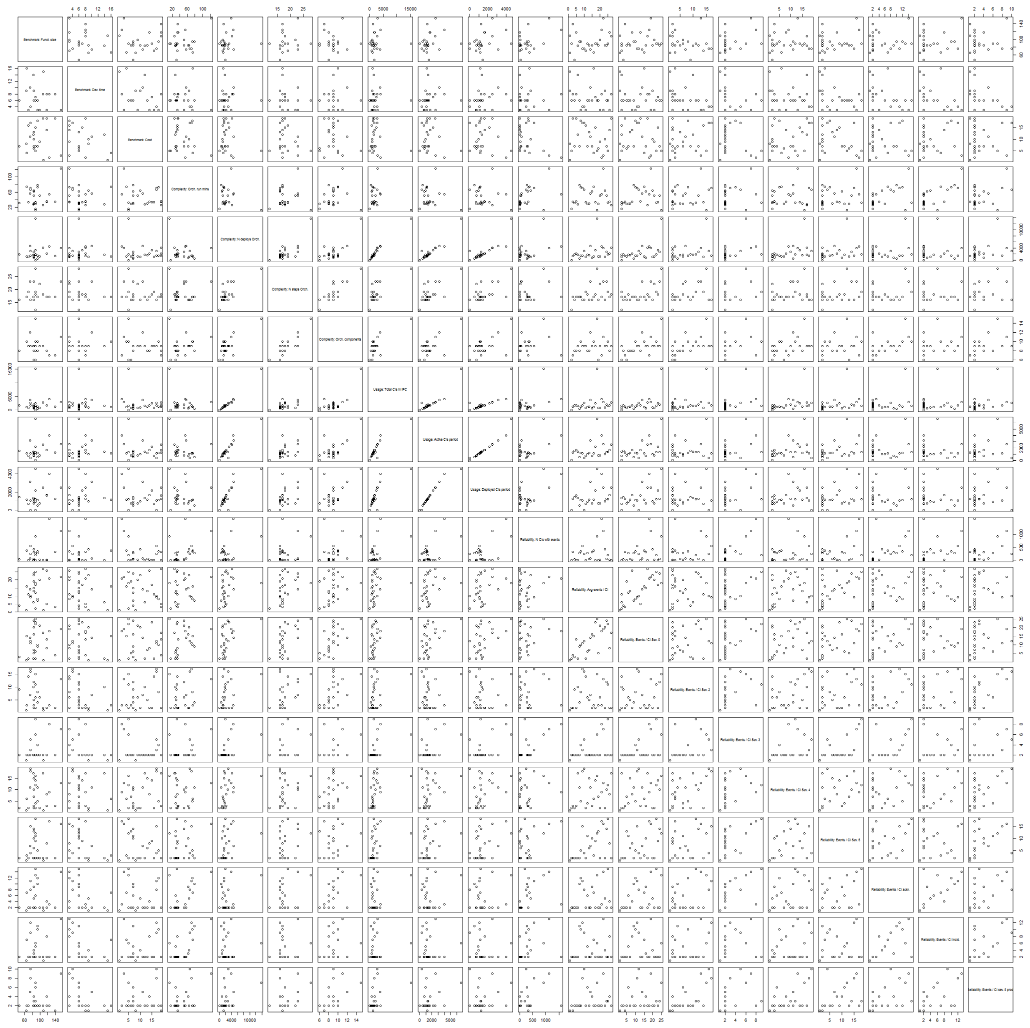
Fig. 5. Correlations between all benchmarking, usage, complexity and reliability metrics.



This correlation matrix depicts correlations between all benchmarking, usage, complexity, and reliability metrics for each Cloud Infra-Service in our study. The size of the circle represents the magnitude of the correlation. The *, ** and *** superscripts represent p-values associated with the correlation, of $> .05$, $> .01$ and $> .005$, respectively. The color of the circle represents the direction (blue for positive, red for negative) of the correlation. The correlations depicted are a subset of a correlation matrix containing all metrics. This matrix was corrected for multiple comparisons using a Benjamini Hochberg correction [24].

B. Scatter Matrix of Metrics

Fig. 6. Scatterplots for all combinations of benchmarking, usage, complexity and reliability metrics.



This scatter matrix depicts scatterplots for all combinations of metrics in the full sample of out study. It can be combined with the correlation matrix above to get an idea of the distribution of data points for each significant combination.

C. Data processing scripts - Versioning information

We used four R scripts to conduct the analyses reported in our paper. Edited versions of these scripts are printed below. The scripts below have been redacted. Information that has been edited out has been replaced by a meta-level description between angle brackets, like so: <description>. The general version information for all scripts is printed directly below. The four scripts used in our study are subsequently printed, each with their own respective header.

```
# Set working directory to the local directory you want to work from:
getwd()
setwd('C:\\Data\\')
getwd()

# System specifications:
# * Windows 7 Enterprise edition (64-bit)
# * 2.4 GHZ processor: Intel i5-6300
# * 16 GB RAM
# * R version: 3.5.1 "Feather Spray"
# * RStudio version: 1.1.453

# Installed packages:
#Package    Version
#assertthat        assertthat     0.2.0
#base64enc          base64enc     0.1-3
#BH                        BH  1.66.0-1
#bindr                  bindr     0.1.1
#bindrcpp            bindrcpp     0.2.2
#bit                      bit    1.1-14
#bitops                bitops     1.0-6
#blob                    blob     1.1.1
#brew                    brew     1.0-6
#bupaR                  bupaR     0.4.1
#caTools              caTools  1.17.1.1
#cli                      cli     1.0.0
#colorspace        colorspace     1.3-2
#covr                    covr     3.1.0
#crayon                crayon     1.3.4
#crosstalk          crosstalk     1.0.0
#curl                    curl       3.2
#data.table        data.table    1.11.4
#devtools            devtools    1.13.6
#DiagrammeR        DiagrammeR     1.0.0
#DiagrammeRsvg  DiagrammeRsvg       0.1
#dichromat          dichromat     2.0-0
#digest                digest    0.6.15
#downloader        downloader       0.4
#dplyr                  dplyr     0.7.6
#edeaR                  edeaR     0.8.1
#evaluate            evaluate      0.11
#eventdataR        eventdataR     0.2.0
#fansi                  fansi     0.2.3
#forcats              forcats     0.3.0
#gapminder          gapminder     0.3.0
#ggplot2              ggplot2     3.0.0
#ggthemes            ggthemes     4.0.0
#git2r                  git2r    0.23.0
#glue                    glue     1.3.0
#gmailr                gmailr     0.7.1
```

```
#gridExtra              gridExtra      2.3
#gtable                    gtable    0.2.0
#hexbin                    hexbin   1.27.2
#highr                      highr      0.7
#hms                          hms    0.4.2
#htmltools              htmltools    0.3.6
#htmlwidgets          htmlwidgets      1.2
#httpuv                    httpuv    1.4.5
#httr                        httr    1.3.1
#igraph                    igraph    1.2.2
#influenceR            influenceR    0.1.0
#jsonlite                jsonlite      1.5
#knitr                      knitr     1.20
#labeling                labeling      0.3
#later                      later    0.7.3
#lazyeval                lazyeval    0.2.1
#lubridate              lubridate    1.7.4
#magrittr                magrittr      1.5
#markdown                markdown      0.8
#memoise                  memoise    1.1.0
#mime                        mime      0.5
#miniUI                    miniUI  0.1.1.1
#munsell                  munsell    0.5.0
#openssl                  openssl    1.0.2
#packrat                  packrat  0.4.9-3
#petrinetR              petrinetR    0.2.0
#pillar                    pillar    1.3.0
#pkgconfig              pkgconfig    2.0.1
#plogr                      plogr    0.2.0
#plotly                    plotly    4.8.0
#plyr                        plyr    1.8.4
#praise                    praise    1.0.0
#prettyunits          prettyunits    1.0.2
#processmapR          processmapR    0.3.2
#processmonitR      processmonitR    0.1.0
#processx                processx    3.1.0
#promises                promises    1.0.1
#purrr                      purrr    0.2.5
#R6                            R6    2.2.2
#RColorBrewer        RColorBrewer    1.1-2
#Rcpp                        Rcpp  0.12.18
#readr                      readr    1.1.1
#reshape2                reshape2    1.4.3
#rex                          rex    1.1.2
#rgexf                      rgexf   0.15.3
#rJava                      rJava   0.9-10
#rlang                      rlang    0.2.1
#rmarkdown              rmarkdown     1.10
#RODBC                      RODBC   1.3-15
#Rook                        Rook    1.1-1
#rprojroot              rprojroot    1.3-2
#rstudioapi            rstudioapi      0.7
#rsvg                        rsvg      1.3
#scales                    scales    0.5.0
#shiny                      shiny    1.1.0
#shinyTime              shinyTime    0.2.1
```

```
#sourcetools     sourcetools     0.1.7
#stringi             stringi     1.1.7
#stringr             stringr     1.3.1
#testthat           testthat     2.0.0
#tibble               tibble     1.4.2
#tidyr                 tidyr     0.8.1
#tidyselect       tidyselect     0.2.4
#tinytex             tinytex       0.6
#utf8                   utf8     1.1.4
#V8                       V8       1.5
#viridis             viridis     0.5.1
#viridisLite     viridisLite     0.3.0
#visNetwork       visNetwork     2.0.4
#whisker             whisker     0.3-2
#withr                 withr     2.1.2
#xesreadR           xesreadR     0.2.2
#xfun                   xfun       0.3
#xlsx                   xlsx     0.6.1
#xlsxjars           xlsxjars     0.6.1
#XML                     XML  3.98-1.12
#xml2                   xml2     1.2.0
#xtable               xtable     1.8-2
#yaml                   yaml     2.2.0
#zoo                     zoo     1.8-3
#translations   translations     3.5.1
```

D. Data processing scripts - Data preparation script.R

```
# Read in relevant libraries:

library("dplyr", lib.loc=<Path to R library folder>)
library("tidyr", lib.loc=<Path to R library folder>))
library("lubridate", lib.loc=<Path to R library folder>))
library("reshape2", lib.loc=<Path to R library folder>))
library("RODBC", lib.loc=<Path to R library folder>))
library("ggplot2", lib.loc=<Path to R library folder>)

# Function definitions:

# NOT USED: Read event bus data from datawarehouse (included for reference to SQL
    query, if desired)
<Function which executes SQL query through ODBC connection with database>
  <Event bus name><-sqlQuery(cnxn, query)
  ## View(<Event bus name>)
  <Event bus name>$FirstOccurrence <- as.POSIXct(<Event bus name>$FirstOccurrence,
      format = '%Y-%m-%d %h:%m:%s')
  #ggplot(data=<Event bus name>, aes(<Event bus name>$FirstOccurrence, <Event bus
      name>$Count, fill=<Event bus name>$AlertGroup)) + geom_col() + scale_x_date()
  return(<Event bus name>)
}


# USED functions:


DEPLOYS_clean_configinstance_name <- function(df){
  # Takes dataframe that includes a field with server name and component description,
      splits this field, and adds both components to dataset.
  df$name <- as.character(df$name)
  tmp <- do.call(rbind, strsplit(df$name, '\\/'))
  df$server <- tmp[,1]
  df$component <- tmp[,2]
  return(df)
}


DEPLOYS_select_most_recent <- function(multiple_servers){
  # Takes dataframe with configuration instances that have multiple entries, and
      selects the most recent deploy as the reference deploy.
  result <- multiple_servers %>%
    group_by(name)%>%
    slice(which.max(sys_created_on)) # In case of duplicates, keep the most recent
        server entry
  return(result)
}


DEPLOYS_count_servers <- function(df){
  dfServers <- df[(grepl("\\/", df$name)==FALSE),] # Not unique, as some servers
      recreated several times
  dfServers$id <- 1:nrow(dfServers)
  dfServers <- dfServers %>%
    group_by(name) %>%
    mutate(unique_servers = n_distinct(id))
  return(dfServers)
}


DEPLOYS_list_unique_server_instances <- function(df){
  # Takes dataframe of deploy-related data, checks if configuration items (server
      names) are unique, and creates list of those instances that have multiple
      entries
  dfServers <- DEPLOYS_count_servers(df)
  multiple_servers <- dfServers[(dfServers$unique_servers > 1),]
  most_recent_of_multiple_servers <- DEPLOYS_select_most_recent(multiple_servers)
  deployments_unique <- rbind((dfServers[(dfServers$unique_servers == 1),]),
      most_recent_of_multiple_servers)
  return(deployments_unique)
}


COMPONENTS_get_components <- function(df){
  dfComponents <- df[(grepl("\\/", df$name)==TRUE),] # Examine how many components
      are on one and the same configuration item, maximum. Note that these are all of
      the extra components that had a backslash in their name field, as such they're
      independent of the multiple server entries addressed above.
  return(dfComponents)
  }


COMPONENTS_extract_component_info <- function(component_df){
  component_info <- component_df[,c("name", "install_status", "sys_class_name", "
      sys_created_on", "sys_created_by", "server")]
  component_info$id <- 1:nrow(component_info)
  return(component_info)
}


COMPONENTS_get_uniques_per_server <- function(component_info){
  unique_components_per_server <- component_info %>%
    group_by(server) %>%
    mutate(unique_components = n_distinct(id))
  return(unique_components_per_server)
}


COMPONENTS_extract_single_component_data <- function(unique_components_per_server){
  singlecomponent <- unique_components_per_server[(
      unique_components_per_server$unique_components <= 1),]
  names(singlecomponent)<-c("c1_name", "c1install_status", "c1_class_name", "
      c1_sys_created_on", "c1_sys_created_by", "server", "throw1", "throw2")
  return(singlecomponent)
}


COMPONENTS_extract_multiple_component_data <- function(unique_components_per_server){
  multiple_components <- unique_components_per_server[(
      unique_components_per_server$unique_components > 1),]
  return(multiple_components)
}


COMPONENTS_get_first_component <- function(multiple_components){
  firstofmultiple <- multiple_components %>% #Get first of two
    group_by(server)%>%
    slice(which.max(id))
  return(firstofmultiple)
}


COMPONENTS_get_second_component <- function(multiple_components){
  secondofmultiple <- multiple_components %>% #Get second of two
    group_by(server)%>%
    slice(which.min(id))
    return(secondofmultiple)
}


COMPONENTS_merge_cleanup <- function(singlecomponent, firstofmultiple,
    secondofmultiple){
  multiples_restructured = merge(x=firstofmultiple, y=secondofmultiple, by="server")
  names(multiples_restructured) <- c("server", "c1_name", "c1install_status", "
      c1_class_name", "c1_sys_created_on", "c1_sys_created_by", "throw1", "throw2", "
      c2_name", "c2install_status", "c2_class_name", "c2_sys_created_on", "
      c2_sys_created_by", "throw3", "throw4")
  components_restructured <- bind_rows(multiples_restructured, singlecomponent)
  return(components_restructured)
}


COMPONENTS_cleanup <- function(components){
  drops <- c("throw1","throw2", "throw3", "throw4", "X", "pk_id", "id", "
      unique_servers")
  components_restructured <- components[ , !(names(components) %in% drops)]
  components_restructured$id <- 1:nrow(components_restructured)
  return(components_restructured)
}


DEPLOYS_create_full_set <- function(deployments_unique, components_restructured){
  deployments_full <- merge(x = deployments_unique, y = components_restructured, by.x
       = "name", by.y = "server", all.x=TRUE)
  deployments_full$sys_created_on <- as.POSIXct(deployments_full$sys_created_on,
      format = '%d-%m-%Y %H:%M:%S')
  deployments_full$sys_updated_on <- as.POSIXct(deployments_full$sys_updated_on,
      format = '%d-%m-%Y %H:%M:%S')
  deployments_full$sys_created_on <- as_datetime(deployments_full$sys_created_on)
  deployments_full$sys_updated_on <- as_datetime(deployments_full$sys_updated_on)
  return(deployments_full)
}


<Event bus>_parseserver <- function(fullSet_use){
  # Take first column of datset, and parse out server name
  fullSet_use$CI_Name <- as.character(fullSet_use$CI_Name)
  out_1 <- do.call(rbind, strsplit(fullSet_use$CI_Name, '\\['))
  # View(out_1)
  out_2 <- do.call(rbind, strsplit(out_1[,2], '\\]'))
  # View(out_2)
  fullSet_use$name <- out_2[,1]
  fullSet_use$name <- as.character(fullSet_use$name)
  return(fullSet_use)
}



# Data preparation script - Configuration item event analysis

# This script describes how we prepared a dataset containing data on events relevant
   to infra deliveries. This dataset contains data from three sources:
# * Data regarding the deployment of configuration items within the in-company cloud
   computing environment of a major bank;
# * Data regarding the registration of these configuration items in a configuration
   management database, which links the configuration instance to one or more cloud
   infra services;
# * Data regarding events that were recorded on each of these configuration items by
   monitoring tools within the bank's infra environment over the period between 31-08
    2017 and 31-08 2018.

# The preparatory steps prior to analysis involve:
# * Reading the appropriate data sets from .csv dumps of the relevant data sources (
   deployment registry, configuration management database and event monitoring
   database)
# * Cleaning and describing each of these data sets
# * Merging these sources, assessing inconsistencies, and reporting these

# Each of these steps will be documented below, with comments specifying any choices
   we made and the line of reasoning behind them.

# This script provides input for the script 'data analysis script'.

# Note that all 'View' and 'print' statements are commented out in the script. Remove
    the '#' to view or print the appropriate outcomes.
# Also, note that when reading this script you can adapt it by including a View() or
   print() call on any of the created variables, to inspect its contents for yourself
   .






# Step 1: Reading source data
deployments <- read.csv(<Path to file>, sep = ",", header=TRUE)
cmdb <- read.csv(<Path to file>, sep = ",", header = TRUE)
<Event bus> <- read.csv(<Path to file>, sep = ",", header = TRUE)


# Step 2: Cleaning source data
# Deployment data

# View(deployments)
# View(unique(deployments$sys_class_name))
# View(count(deployments, name))
# A number of configuration item deployments has the same configuration item number.
# We want to relate our event data to unique configuration items.
# In addition, a number of deploys has a configuration item name consisting of a
    server name, with a / and a description of the component appended.
# These seem to be multiple components related to the same server.
#
# To resolve the ambiguities introduced by multiple components sharing the same name,
     we need to take two steps:
# * Map various components, which each receive their own registry on the same
    configuration item, onto one configuration item;
# * Deduplicate multiple entries for the same component for the same configuration
    item. To ensure recency, we take the last entry to be leading.
#
# First, we split the configuration item on the '/' character, and extract the
    resulting two parts as "server" and "component"
deploy_config_cleaned <- DEPLOYS_clean_configinstance_name(deployments)
# Then, we can see that some configuration items have had multiple deploys on the
    same item, because there are multiple lines with no components, but the same
    server number.
# We resolve the ambiguity by selecting the most recent deploy for each server, and
    excluding the others from our dataset. The reasoning behind this is that the most
    recent deploy was the last attempt, giving it the most relevance. We verified that
     in all cases, the most recent deploy was the only one that was potentially still
    active.
deploy_single_servers <- DEPLOYS_list_unique_server_instances(deploy_config_cleaned)


#Next, we subset those components from configuration items that have a '/' in their
    configuration item name
deploy_components_added <- COMPONENTS_get_components(deploy_config_cleaned)

# We can now get the information from these deployments that we want to save
deploy_components <- COMPONENTS_extract_component_info(deploy_components_added)

# We can then count how many components are in use on a server
unique_components_per_server <- COMPONENTS_get_uniques_per_server(deploy_components)

# Then, we divide the dataframe by subsetting configuration items with one component
    and configuration items with multiple components
single_components <- COMPONENTS_extract_single_component_data(
    unique_components_per_server)
multiple_components <- COMPONENTS_extract_multiple_component_data(
    unique_components_per_server)
View(unique(multiple_components$server)) #<Number> servers have multiple components
    running on them

# For the configuration instances with multiple components, we can extract each
    component, and merge them together based on the configuration instance name
first_component <- COMPONENTS_get_first_component(multiple_components)
second_component <- COMPONENTS_get_second_component(multiple_components)
merge_components_into_single_server_df <- COMPONENTS_merge_cleanup(single_components,
    first_component, second_component)

# Finally, we can clean up the resulting data frame. Note that we have not lost any
   data other than duplicated data for the multiple components per configuration item
   , which we collapsed into single lines.
components <- COMPONENTS_cleanup(merge_components_into_single_server_df)
cleaned_components <- COMPONENTS_cleanup(components)
deployments_processed <- DEPLOYS_create_full_set(deploy_single_servers,
   cleaned_components)
# View(deployments_processed)



# CMDB data

# View(cmdb)
# View(unique(cmdb$u_pattern_type))
# View(count(cmdb, name))
# A number of configuration item deployments has the same configuration item number.
# We want to relate our event data to unique configuration items.
# In addition, the u_pattern_type field contains entries that are synonymous with one
    another, and blank values. We want this field to include a unique descriptor per
   cloud infra service.
#
# To resolve the ambiguities introduced by multiple instances sharing the same name,
   we can deduplicate multiple entries following the logic also used for deployments
   above.
# To standardize cloud infra service names, we need to attempt to rename
   u_pattern_type into consistent indicators, and fill in the blanks for as far as
   possible.
# Filling in the blanks can be done by using values from the OS field if the
   u_pattern_type field is empty, and later on by merging with the deploy data (which
    may have a sys_class_name that is indicative of a certain cloud infra service)

# First, let's assign the data an id to group by later.
cmdb$id <- 1:nrow(cmdb)

# Next, let's assess the number of unique configuration items within the dataset. To
   check this, we can group entries by server name and count unique identifiers
   associated with each server.
cmdb_tmp <- cmdb %>%
  group_by(name) %>%
  mutate(unique_components = n_distinct(id))

# Given that there are several servers with multiple entries, we can now subset for
   them, select the most recent one, and bind the result.
cmdb_singleserver <- cmdb_tmp[cmdb_tmp$unique_components == 1,]
cmdb_multiserver <- cmdb_tmp[cmdb_tmp$unique_components > 1,]
View(unique(cmdb_multiserver$name)) # <Number> servers have multiple deploys in CMDB
cmdb_selection <- DEPLOYS_select_most_recent(cmdb_multiserver)
cmdb_uniques <- bind_rows(cmdb_singleserver, cmdb_selection)

# We can then set the date time fields to the appropriate type, and complete our CMDB
    data

cmdb_uniques$sys_created_on <- dmy_hms(cmdb_uniques$sys_created_on)
```

```
cmdb_uniques$sys_updated_on <- dmy_hms(cmdb_uniques$sys_updated_on)

# With that settled, we can rename the entries in the u_pattern_type field to be
    consistent with one another.
# First, we need to create a temporary merge between the deploy and cmdb datasets
Deploys_CMDB_ok <- merge(x = deployments_processed, y = cmdb_uniques, by = 'name',
    all.x=TRUE)

# We can check if sys_class_name and the deployment pattern names can help fill out
    missing u_pattern_type values, by examining cases in which the former is filled
    while the latter is not
View(Deploys_CMDB_ok[((is.na(Deploys_CMDB_ok$u_pattern_type))&(!is.na(
    Deploys_CMDB_ok$sys_class_name))),]) # These appear to be only the non-matched
    deployments.
View(cmdb[is.na(cmdb$u_pattern_type),]) # Verify: Non of the entries in the CMDB are
    missing pattern type



# Then, we can plot counts of the sys_class_name variable from the deployment data to
     counts of the u_pattern_type variable from the cmdb data, in order to assess
    where mismatches occur
# We can create views for any of these tables on the command line interface, to use
    them to cross-reference entries.
#n_deploys_vms <- table(Deploys_CMDB_ok$u_pattern_type,
    Deploys_CMDB_ok$sys_class_name)
#n_deploys_components1 <- table(Deploys_CMDB_ok$u_pattern_type,
    Deploys_CMDB_ok$c1_class_name)
#n_deploys_components2 <- table(Deploys_CMDB_ok$u_pattern_type,
    Deploys_CMDB_ok$c2_class_name)

#n_deploys_vms_os <- table(Deploys_CMDB_ok$u_os, Deploys_CMDB_ok$sys_class_name)
#n_deploys_components1_os <- table(Deploys_CMDB_ok$u_os,
    Deploys_CMDB_ok$c1_class_name)
#n_deploys_components2_os <- table(Deploys_CMDB_ok$u_os,
    Deploys_CMDB_ok$c2_class_name)

#n_deploys_vms_overall <- table(Deploys_CMDB$u_pattern_type,
    Deploys_CMDB$sys_class_name)
#n_deploys_components1_overall <- table(Deploys_CMDB$u_pattern_type,
    Deploys_CMDB$c1_class_name)
#n_deploys_components2_overall <- table(Deploys_CMDB$u_pattern_type,
    Deploys_CMDB$c2_class_name)

#n_deploys_vms_os_overall <- table(Deploys_CMDB$u_os, Deploys_CMDB$sys_class_name)
#n_deploys_components1_os_overall <- table(Deploys_CMDB$u_os,
    Deploys_CMDB$c1_class_name)
#n_deploys_components2_os_overall <- table(Deploys_CMDB$u_os,
    Deploys_CMDB$c2_class_name)

#Subset_Deploys_CMDB_active <- Deploys_CMDB_ok[((Deploys_CMDB_ok$operational_status
    == 'Operational')|(Deploys_CMDB_ok$sys_updated_on.x > as.Date('2017-08-31', format
     = '%Y-%m-%d'))),]

#n_deploys_active_vms <- table(Subset_Deploys_CMDB_active$u_pattern_type,
    Subset_Deploys_CMDB_active$sys_class_name)
```

```
#n_deploys_active_components1 <- table(Subset_Deploys_CMDB_active$u_pattern_type,
    Subset_Deploys_CMDB_active$c1_class_name)
#n_deploys_active_components2 <- table(Subset_Deploys_CMDB_active$u_pattern_type,
    Subset_Deploys_CMDB_active$c2_class_name)

#n_deploys_active_vms_os <- table(Subset_Deploys_CMDB_active$u_os,
    Subset_Deploys_CMDB_active$sys_class_name)
#n_deploys_active_components1_os <- table(Subset_Deploys_CMDB_active$u_os,
    Subset_Deploys_CMDB_active$c1_class_name)
#n_deploys_active_components2_os <- table(Subset_Deploys_CMDB_active$u_os,
    Subset_Deploys_CMDB_active$c2_class_name)


# Then, based on the content of these tables, we can rename the entries to match
    between deployments and cmdb.
<Series of If-Then-Else statements renaming synonymous cloud infra service names to
    unique indicators>
# View(cmdb_uniques %>%
#        group_by(u_pattern_type) %>%
#        summarise(n()))
# Note that there are still an number of values in u_pattern_type we are not using
    now, but which could become relevant at some other point in time.

cmdb_processed <- cmdb_uniques



# <Event bus> data

# View(<Event bus>)
# View(count(<Event bus>, name))
# The <Event bus> monitoring data looks good, but has multiple entries for many
    servers. Some of these include the server names between square brackets, meaning
    we'll have to parse them.
# As the SQL query that yielded this data was loosely based on the list of
    deployments within IPC, and we have no way of verifying whether each of the
    configuration items listed in deploys are being logged, the event monitoring data
    may include only an approximation of the total set of IPC deliveries. We will need
     to keep a close eye on the numbers of servers matched when we merge the files.
#
# For <Event bus>, we need to parse the configuration item names to pull out the
    servers
<Event bus>_parsed <- <EVENT BUS>_parseserver(<Event bus>)
<Event bus>$CI_Name <- tolower(<Event bus>$CI_Name)
# Then, we can set the fields containing datetimes to their appropriate types.
<Event bus>$StateChange <- as_datetime(<Event bus>$StateChange)
<Event bus>$FirstOccurrence <- as_datetime(<Event bus>$FirstOccurrence)
<Event bus>$LastOccurrence <- as_datetime(<Event bus>$LastOccurrence)
<Event bus>$DeleteDat <- as_datetime(<Event bus>$DeleteDat)
<Event bus>$LastUpdated <- as_datetime(<Event bus>$LastUpdated)
<Event bus>_processed <- <Event bus>
# Future work: Verify representativeness of data more conclusively, as data is now
    based on server number grep
```

```
# Step 3: Merging source data

# Count all unique servers in each file
# View(length(unique(deployments_processed$name))) # <Number> unique deployed servers
    within IPC
# View(length(unique(cmdb_processed$name))) # <Number> unique servers registered in
   CMDB as deployed within IPC
# View(length(unique(<Event bus>_processed$CI_Name))) # <Number> unique servers with
    registered events over the past year within IPC (roughly)


# Deployments: Missing values and counts
# View(deployments_processed %>%
#         group_by(<status>) %>%
#         summarise(n()))
# View(unique(deployments_processed[(deployments_processed$ <status>),"server"])) # <
   Number> deployments listed as <status> at time of dump
# View(deployments_processed %>%
#         group_by(<status>) %>%
#         summarise(n()))
# View(unique(deployments_processed[(deployments_processed$<status>),"server"])) # <
   Number> deployments listed as <status> at time of dump
# View(deployments_processed %>%
#         group_by(sys_updated_on) %>%
#         summarise(n()))
# View(unique(deployments_processed[((deployments_processed$<status>  == <status>)|(
    deployments_processed$sys_updated_on > as.Date('2017-08-31', format = '%Y-%m-%d'))
    ),"server"])) # <Number> unique servers updated over the last year


# Cmdb: Missing values and descriptives
# View(cmdb_processed %>%
#       group_by(state) %>%
#       summarise(n()))
# <Number> without value, <Number> decommissioned, <Number> active servers
# <Number> deployments done - <Number> active servers = an estimated <Number> failed
    deployments
# Does not match servers between files. Merge these tables below for a more complete
    picture.




# Merge deployments and cmdb
deployments_cmdb <- merge(x = deployments_processed, y = cmdb_processed, by.x = "
    server", by.y = "name", all.x=TRUE)
server_nonmatch <- deployments_cmdb[(is.na(deployments_cmdb$sys_created_on.y)),] # <
   Number> servers were deployed, but never entered into the cmdb. Possibly, servers
   are missing from the cmdb dump, as it is based on an export of <Type1> and <Type2>
     servers.
server_match <- deployments_cmdb[(is.na(deployments_cmdb$sys_created_on.y)==FALSE),]
   # <Number> matched: All servers from the cmdb were also in the deployment data.
deployments_cmdb_processed <- deployments_cmdb[(is.na(deployments_cmdb$sys_created_on
    .y)==FALSE),]

cmdb$u_pattern_type <- ifelse(((cmdb$u_pattern_type == "none")|(cmdb$u_pattern_type
    == "")),as.character(cmdb$u_os), as.character(cmdb$u_pattern_type))
cmdb$u_pattern_type <- ifelse(is.na(cmdb$u_pattern_type), as.character(cmdb$u_os), as
    .character(cmdb$u_pattern_type))

list_os_only_servers <- deployments_cmdb_processed[((is.na(
    deployments_cmdb_processed$c1_class_name)|deployments_cmdb_processed$c1_class_name
    == <Class name>) & (is.na(deployments_cmdb_processed$c2_class_name)|
    deployments_cmdb_processed$c2_class_name==<Class name>)),"server"]
# Future work: This should be repeated with a full dump of deliveries included in the
    IPC CMDB

# Server nonmatch composition: Which kinds of server did not match to IPC CMDB?
# View(server_nonmatch%>%
#   group_by(sys_class_name) %>%
#   summarise(nonmatch_components = n()))

# Were any of the no-matching servers showing up in <Event bus>?
# View(server_nonmatch %>%
#          mutate(indicateit = name %in% <Event bus>$CI_Name) %>%
#          filter(indicateit==TRUE) %>%
#          group_by(c1_class_name) %>%
#          summarise(count_components = n()))
# Turns out some were, making our data not entirely consitent. Numbers are small
    though, so it is likely not a problem.



# Get counts of deployments registered in CMDB per solution:
# For full time period
# Full set no <DEVELOPMENT TENANT>
 View(deployments_cmdb_processed %>%
        filter((ref_cmdb_ci_vmware_instance.u_tenant!=<Development tenant>)&(
            ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)) %>%
        group_by(u_pattern_type) %>%
        summarise(n())) #REPORTED ANALYSIS
# OS only no <DEVELOPMENT TENANT>
 View(deployments_cmdb_processed %>%
        filter(server %in% list_os_only_servers) %>%
        filter((ref_cmdb_ci_vmware_instance.u_tenant!=<Development tenant>)&(
            ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)) %>%
        group_by(u_pattern_type) %>%
        summarise(n())) #REPORTED ANALYSIS



# Created CIs since 31-08-2017
# Full set no <DEVELOPMENT TENANT>
 View(deployments_cmdb_processed %>%
        filter(sys_created_on.x > '2017-08-31') %>%
        filter((ref_cmdb_ci_vmware_instance.u_tenant!=<Development tenant>)&(
            ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)) %>%
        group_by(u_pattern_type) %>%
        summarise(n())) #REPORTED ANALYSIS
# OS only no <DEVELOPMENT TENANT>
 View(deployments_cmdb_processed %>%
        filter(sys_created_on.x > '2017-08-31') %>%
        filter(server %in% list_os_only_servers) %>%
        filter((ref_cmdb_ci_vmware_instance.u_tenant!=<Development tenant>)&(
           ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)) %>%
        group_by(u_pattern_type) %>%
        summarise(n())) #REPORTED ANALYSIS



# Active CIs since 31-08-2017
# Full set no <DEVELOPMENT TENANT>
View(deployments_cmdb_processed %>%
        filter((install_status != <Status>)|((install_status == <Status>)&(
           sys_updated_on.x > '2017-08-31'))) %>%
        filter((ref_cmdb_ci_vmware_instance.u_tenant!=<Development tenant>)&(
           ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)) %>%
        group_by(u_pattern_type) %>%
        summarise(n())) #REPORTED ANALYSIS
# OS only no <DEVELOPMENT TENANT>
 View(deployments_cmdb_processed %>%
        filter((install_status != <Status>)|((install_status == <Status>)&(
           sys_updated_on.x > '2017-08-31'))) %>%
        filter(server %in% list_os_only_servers) %>%
        filter((ref_cmdb_ci_vmware_instance.u_tenant!=<Development tenant>)&(
           ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)) %>%
        group_by(u_pattern_type) %>%
        summarise(n())) #REPORTED ANALYSIS




# First deployment timestamps

# Get first deploy date per delivery from deploy data
# Without <DEVELOPMENT TENANT>
first_dates_prod_noIPC <- deployments_cmdb_processed %>%
  filter((ref_cmdb_ci_vmware_instance.u_tenant!=<Development tenant>)&(
     ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)) %>%
  group_by(u_pattern_type) %>%
  summarise(min(sys_created_on.y))

first_dates_prod_noIPC_os <- deployments_cmdb_processed %>%
  filter(server %in% list_os_only_servers) %>%
  filter((ref_cmdb_ci_vmware_instance.u_tenant!=<Development tenant>)&(
     ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)) %>%
  group_by(u_pattern_type) %>%
  summarise(min(sys_created_on.y))


# Merge deployments_cmdb and <Event bus>
deployments_cmdb_<Event bus> <- merge(x = deployments_cmdb_processed, y = <Event bus>
   _processed, by.x = "name", by.y = "CI_Name", all.y=TRUE)
# Remove reports on servers that were not in CMDB
deployments_cmdb_<Event bus>_processed <- deployments_cmdb_<Event bus>[(is.na(
   deployments_cmdb_<Event bus>$sys_created_on.x)==FALSE),] # <Number> logged events
   for IPC servers
# View(deployments_cmdb_<Event bus>_processed %>%
#   group_by(u_pattern_type) %>%
#   summarise(n()))

# We can examine which servers from the Deployment and CMDB data did not have any
    events
matched_<Event bus> <- unique(deployments_cmdb_<Event bus>_processed$name)
non_matched_<Event bus> <- deployments_cmdb_processed[!(
    deployments_cmdb_processed$name %in% matched_<Event bus>),]

# Most deploys will have an OS installed on them. We are interested in the events
    related specifically to OS, so we will have to partial out the deploys which have
    just an OS.
# Select the OS-related events for specific analysis
deployments_cmdb_<Event bus>_processed_os <- deployments_cmdb_<Event bus>_processed[(
    deployments_cmdb_<Event bus>_processed$name %in% list_os_only_servers)&((
    deployments_cmdb_<Event bus>_processed$u_pattern_type<OS name>)|(deployments_cmdb_
    <Event bus>_processed$u_pattern_type==<OS name 2>)|(deployments_cmdb_<Event bus>
    _processed$u_pattern_type<OS name 3>)),] # <Number> OS only machines


# Clean up dataset: Further descriptives on full data set and OS only data set
categorical_variables <- <Concatenation of categorical type variable names>
free_entry_variables <- <Concatenation of free text type variable names>
timestamp_variables <- <Concatenation of timestamp type variable names>
continuous_variables <- <Concatenation of continuous type variable names>

# Descriptives on full data set and OS only data set
#for (var in categorical_variables){
  #print(var)
  #View(deployments_cmdb_<Event bus>_processed %>%
  #   group_by_(.dots = var) %>%
  #   summarise(count = n(), proportion_total_events = n()/<Number of total events>))
  # browser()
#}


#for (var in categorical_variables){
#print(var)
#View(deployments_cmdb_<Event bus>_processed_os %>%
#   group_by_(.dots = var) %>%
#   summarise(count = n(), proportion_total_events = n()/<Number of total events>))
# browser()
#}

for(var in continuous_variables){
  deployments_cmdb_<Event bus>_processed[,var] <- as.integer(deployments_cmdb_<Event
      bus>_processed[,var])
  deployments_cmdb_<Event bus>_processed_os[,var] <- as.integer(deployments_cmdb_<
      Event bus>_processed_os[,var])
}
<Descriptives for continuous variables: Average and SD>
<Descriptives for continuous variables: Average and SD for OS only dataset>

for(var in timestamp_variables){
  # print(var)
  # View(min(deployments_cmdb_<Event bus>_processed[,var]))
  # View(max(deployments_cmdb_<Event bus>_processed[,var]))
  # print('os:')
  # View(min(deployments_cmdb_<Event bus>_processed_os[,var]))
  # View(max(deployments_cmdb_<Event bus>_processed_os[,var]))
}


# install_status.x and install_status.y are two products from a merge, that provide
    complementary information. Merge them.
# merge install_status.x and install_status.y if they are complementary, do for all
    remaining x - y pairs
names_with_.x <- (grep("\\.x$", names(deployments_cmdb_<Event bus>_processed), value=
    TRUE))
names_with_.y <- (grep("\\.y$", names(deployments_cmdb_<Event bus>_processed), value=
    TRUE))
names_with_.y <- names_with_.y[names_with_.y!="server.y"]
#deployments_cmdb_<Event bus>_processed$server <- ifelse(is.na(deployments_cmdb_<
    Event bus>_processed$server), deployments_cmdb_<Event bus>_processed$server.y,
    deployments_cmdb_<Event bus>_processed$server)
#deployments_cmdb_<Event bus>_processed_os$server <- ifelse(is.na(deployments_cmdb_<
    Event bus>_processed_os$server), deployments_cmdb_<Event bus>_processed_os$server.
    y, deployments_cmdb_<Event bus>_processed_os$server)
# Merge x and y columns
for (i in names_with_.x){
  name_y <- gsub("\\.x", ".y", i)
  replace_index <- (is.na(deployments_cmdb_<Event bus>_processed[,i]) & (!is.na(
      deployments_cmdb_<Event bus>_processed[,name_y])))
  if(length(replace_index[replace_index==TRUE])>0){
    deployments_cmdb_<Event bus>_processed[replace_index,i] <- deployments_cmdb_<
        Event bus>_processed[replace_index,name_y]
  }
  replace_index_os <- (is.na(deployments_cmdb_<Event bus>_processed_os[,i]) & (!is.na
      (deployments_cmdb_<Event bus>_processed_os[,name_y])))
  if(length(replace_index_os[replace_index_os==TRUE])>0){
    deployments_cmdb_<Event bus>_processed_os[replace_index_os,i] <-
        deployments_cmdb_<Event bus>_processed_os[replace_index_os,name_y]
  }}
replace_index <- (is.na(deployments_cmdb_<Event bus>_processed[,i]) & (!is.na(
    deployments_cmdb_<Event bus>_processed[,name_y])))


# Fill list of columns to delete based on descriptive information (too many missing
    values, irrelevant information, etc)
delete <- <Concatenated list of columns to be deleted>

dataframe_full <- deployments_cmdb_<Event bus>_processed[, (!(names(deployments_cmdb_
    <Event bus>_processed) %in% delete)&(!(names(deployments_cmdb_<Event bus>
    _processed) %in% names_with_.y)))]
dataframe_full_os <- deployments_cmdb_<Event bus>_processed_os[, (!(names(
    deployments_cmdb_<Event bus>_processed_os) %in% delete)&!((names(deployments_cmdb_
    <Event bus>_processed_os) %in% names_with_.y)))]
names(dataframe_full) <- gsub("\\.x", "", names(dataframe_full))
names(dataframe_full_os) <- gsub("\\.x", "", names(dataframe_full_os))
```

E. Data processing scripts - Data analytics script.R

```
# Read in relevant libraries:

library("dplyr", lib.loc=<Path to R library folder>)
library("tidyr", lib.loc=<Path to R library folder>)
library("lubridate", lib.loc=<Path to R library folder>)
library("reshape2", lib.loc=<Path to R library folder>)
library("RODBC", lib.loc=<Path to R library folder>)
library("ggplot2", lib.loc=<Path to R library folder>)
library("lazyeval", lib.loc=<Path to R library folder>)
library("zoo", lib.loc=<Path to R library folder>)

# Define functions

pad <- function(daycount, patterns){
  for (i in seq(1,length(patterns))){
    pattern <- patterns[i]
    patternset <- daycount[(daycount$u_pattern_type==pattern),]
    alldays <- seq(min(daycount$DateOccurred),length=366,by="+1 day")
    tmp1 <- as.Date(alldays, format = '%Y-%m-%d')
    tmp2 <- as.Date(patternset$DateOccurred, format = '%Y-%m-%d')
    allcount <- data.frame(alldays)     # create table object from alldays.
    actindex <- tmp1 %in% tmp2
    allcount$actind <- actindex
    allcount[,"n"]<-0
    allcount[actindex==TRUE,"n"] <- patternset$n
    allcount$n <- ifelse(is.na(allcount$n), 0, allcount$n)
    avg_over_days <- allcount %>%
      summarise(nmean = mean(allcount$n),
                nsd = sd(allcount$n))
    print(pattern)
    print(avg_over_days)
  }
  return()
}


loop_over_levels <- function(df, loopvar, patternlist){
  looplist <- unlist(unique(df[,loopvar]))
  for (j in seq(1, length(looplist),1)){
    k <- looplist[j]
    tmpfilter <- df[,loopvar]==k
    subset_df <- df[tmpfilter,]
    count_event_days_subset <- subset_df %>%
      group_by(DateOccurred, u_pattern_type) %>%
      count()
    print(' ')
    print(' ')
    print(loopvar)
    print(k)
    print(' ')
    print(' ')
    View(subset_df %>%
           group_by(u_pattern_type) %>%
           count())
      if (nrow(subset_df) >= 1){
        pad(count_event_days_subset, patternlist)
      browser()
      }
    }
    return()
}

# This script builds on the 'data preparation script' by taking several variables
    from that script as input. Please run the âĂŸdata preparation scriptâĂŹ first,
    prior to running this script.

# Note that all 'View' and 'print' statements are commented out in the script. Remove
     the '#' to view or print the appropriate analysis outcomes.
# Also, note that when reading this script you can adapt it by including a View() or
    print() call on any of the created variables, to inspect its contents for yourself
    .


# Begin analyses
# View(dataframe_full)

# Examine descriptives

# Time-related variables
# View(dataframe_full[(dataframe_full$sys_updated_on)!=dataframe_full$LastUpdated,])
    #sys_update time from deploy data does not match lastupdated from <Event bus> data
    . Take deploy data as indicative of retirement when install_status = <Status>

# Create decom date and lifecycle
dataframe_full$deploy_decom_date <- dataframe_full$sys_updated_on
dataframe_full[(dataframe_full$install_status==<Status>),"deploy_decom_date"] <- NA
dataframe_full_os$deploy_decom_date <- dataframe_full_os$sys_updated_on
dataframe_full_os[(dataframe_full_os$install_status==<Status>),"deploy_decom_date"]
    <- NA

# Examine periods for sys_created_on, decom_date (full set)

# View(dataframe_full %>%
#        group_by(u_pattern_type) %>%
#        summarise(first_created = min(sys_created_on), last_created = max(
    sys_created_on), delivery_creation_period=difftime(max(sys_created_on), min(
    sys_created_on), units='days')))

dataframe_full <- dataframe_full %>%
       group_by(u_pattern_type, name) %>%
       mutate(lifecycle_time = as.numeric(difftime(deploy_decom_date, sys_created_on,
           units='days')))

# View(dataframe_full %>%
#        group_by(u_pattern_type) %>%
#        summarise(minimum_lifecycle_time = min(lifecycle_time),
    maximum_lifecycle_time = max(lifecycle_time), avg_lifecycle_time = mean(
    lifecycle_time, na.rm=TRUE), sd_lifecycle_time = sd(lifecycle_time, na.rm=TRUE)))
```

```
# Examine periods for sys_created_on, decom_date (Os only)

# View(dataframe_full_os %>%
#       group_by(u_pattern_type) %>%
#       summarise(first_created = min(sys_created_on), last_created = max(
    sys_created_on), delivery_creation_period=difftime(max(sys_created_on), min(
    sys_created_on), units='days')))

dataframe_full_os <- dataframe_full_os %>%
  group_by(u_pattern_type, name) %>%
  mutate(lifecycle_time = as.numeric(difftime(decom_date, sys_created_on, units='days
      ')))

# View(dataframe_full_os %>%
#       group_by(u_pattern_type) %>%
#       summarise(minimum_lifecycle_time = min(lifecycle_time),
    maximum_lifecycle_time = max(lifecycle_time), avg_lifecycle_time = mean(
    lifecycle_time, na.rm=TRUE), sd_lifecycle_time = sd(lifecycle_time, na.rm=TRUE)))



# Number of events per infra delivery

# View(dataframe_full %>%
#       group_by(u_pattern_type) %>%
#       summarise(number_of_events = n()))

# View(dataframe_full_os %>%
#       group_by(u_pattern_type) %>%
#       summarise(number_of_events = n()))

# Excluding the <DEVELOPMENT TENANT> tenant (i.e. excluding IPC development machines)

dataframe_noIPC <- dataframe_full[((dataframe_full$ref_cmdb_ci_vmware_instance.
    u_tenant!='<Development tenant>')&(dataframe_full$ref_cmdb_ci_vmware_instance.
    u_tenant!=<Testing tenant>)),]
dataframe_os_noIPC <- dataframe_full_os[((
    dataframe_full_os$ref_cmdb_ci_vmware_instance.u_tenant!='<Development tenant>')&(
    dataframe_full_os$ref_cmdb_ci_vmware_instance.u_tenant!=<Testing tenant>)),]

# View(dataframe_noIPC %>%
#         group_by(u_pattern_type) %>%
#         summarise(number_of_events = n()))

# View(dataframe_os_noIPC %>%
#      group_by(u_pattern_type) %>%
#       summarise(number_of_events = n()))


# Number of events per delivery

Counts_overall_Events_noIPC <- table(dataframe_noIPC$u_pattern_type,
    dataframe_noIPC$sys_class_name)
Counts_overall_Events_os_noIPC <- table(dataframe_os_noIPC$u_pattern_type,
    dataframe_os_noIPC$sys_class_name)
```

```
# Count number of CIs with events registered

# Extract the number of CIs with events
CIs_with_events_noIPC <- dataframe_noIPC %>%
  group_by(u_pattern_type) %>%
  summarise(n_CIs = length(unique(name))) # REPORTED ANALYSIS

# OS only excl <DEVELOPMENT TENANT>
CIs_with_events_os_noIPC <- dataframe_os_noIPC %>%
  group_by(u_pattern_type) %>%
  summarise(n_CIs = length(unique(name))) # REPORTED ANALYSIS


# Averages per day

# Extract date on which event occurred
dataframe_noIPC$DateOccurred <- as.Date(dataframe_noIPC$FirstOccurrence, format = '%Y
    -%m-%d')
dataframe_os_noIPC$DateOccurred <- as.Date(dataframe_os_noIPC$FirstOccurrence, format
    = '%Y-%m-%d')

#Note: the next two code blocks are included for reference, but not reported in our
    paper or repository
# Extract count of number of events per infra delivery
count_event_days_noIPC <- dataframe_noIPC %>%
  group_by(DateOccurred, u_pattern_type) %>%
  count()

count_event_days_os_noIPC <- dataframe_os_noIPC %>%
  group_by(DateOccurred, u_pattern_type) %>%
  count()

# Average the counts of events of infra delivery per day
avg_over_days_noIPC <- count_event_days_noIPC %>%
  group_by(u_pattern_type) %>%
  summarise(nmean = mean(n),
            nsd = sd(n))

avg_over_days_os_noIPC <- count_event_days_os_noIPC %>%
  group_by(u_pattern_type) %>%
  summarise(nmean = mean(n),
            nsd = sd(n))


# Weights are based on number of days on which events were observed, rather than
    total number of days in period.
# We want to construct a full, padded time series to calculate mean per passed day (
    rather than day with event registered),
# and create a rolling average.

# Create (padded) time series of numbers of events

patterns <- unique(dataframe_full$u_pattern_type)

# At the start of this script, we've defined two functions. Taken together, these:
```

```
# * turn the observed events of an indicated type into a time series for an entire
    year;
# * calculate the number of observed events for all configuration items of each cloud
     infra service;
# * weigh this number of observed events by the number of days in the year to arrive
    at an average number of events per infra delivery per day, with a standard
    deviation

pad(count_event_days_noIPC, unique(dataframe_noIPC$u_pattern_type))
pad(count_event_days_os_noIPC, unique(dataframe_os_noIPC$u_pattern_type))

# severity
loop_over_levels(dataframe_IPC_only, "Severity", patterns)
loop_over_levels(dataframe_IPC_only_os, "Severity", patterns)

# acknowledged
loop_over_levels(dataframe_noIPC, "Acknowledged", patterns)
loop_over_levels(dataframe_os_noIPC, "Acknowledged", patterns)

# incidentnr
dataframe_noIPC$IncidentYN <- ifelse(grepl('^I', dataframe_noIPC$IncidentNr), 1, 0)
loop_over_levels(dataframe_noIPC, "IncidentYN", patterns)
dataframe_os_noIPC$IncidentYN <- ifelse(grepl('^I', dataframe_os_noIPC$IncidentNr),
    1, 0)
loop_over_levels(dataframe_os_noIPC, "IncidentYN", patterns)

# used_for (type of configuration items)

# grouped by severity 5

# View(dataframe_IPC_only %>%
#       group_by(u_pattern_type, used_for, Severity) %>%
#       summarize(n()))

# View(dataframe_IPC_only_os %>%
#       group_by(u_pattern_type, used_for, Severity) %>%
#       summarize(n()))

# grouped by acknowledged

# View(dataframe_IPC_only %>%
#       group_by(u_pattern_type, used_for, Acknowledged) %>%
#       summarize(n()))

# View(dataframe_IPC_only_os %>%
#       group_by(u_pattern_type, used_for, Acknowledged) %>%
#       summarize(n()))

# grouped by IncidentYN

# View(dataframe_IPC_only %>%
#       group_by(u_pattern_type, used_for, IncidentYN) %>%
#       summarize(n()))

# View(dataframe_IPC_only_os %>%
#       group_by(u_pattern_type, used_for, IncidentYN) %>%
#       summarize(n()))



# Finally, we can look at the average amount of events per configuration items for
    each delivery, to know whether effects we are interested in are independent of the
     number of configuration items we have.

# n events
avg_events_CI <- dataframe_noIPC %>%
  group_by(u_pattern_type, name) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI) # REPORTED ANALYSIS

# n_events_os
avg_events_CI_os <- dataframe_os_noIPC %>%
  group_by(u_pattern_type, name) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_os) # REPORTED ANALYSIS


# severity
avg_events_CI_severity <- dataframe_noIPC %>%
  group_by(u_pattern_type, Severity, name) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type, Severity) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_severity) # REPORTED ANALYSIS

# severity_os
avg_events_CI_os_severity <- dataframe_os_noIPC %>%
  group_by(u_pattern_type, Severity, name) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type, Severity) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_os_severity) # REPORTED ANALYSIS


# Acknowledged
avg_events_CI_acknowledged <- dataframe_noIPC %>%
  group_by(u_pattern_type, Acknowledged, name) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type, Acknowledged) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_acknowledged) # REPORTED ANALYSIS

# Acknowledged_os
avg_events_CI_os_acknowledged <- dataframe_os_noIPC %>%
  group_by(u_pattern_type, Acknowledged, name) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type, Acknowledged) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_os_acknowledged) # REPORTED ANALYSIS


# Incidents
avg_events_CI_incidents <- dataframe_noIPC %>%
  group_by(u_pattern_type, IncidentYN, name) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type, IncidentYN) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_incidents) # REPORTED ANALYSIS

# Incidents_os
avg_events_CI_os_incidents <- dataframe_os_noIPC %>%
  group_by(u_pattern_type, IncidentYN, name) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type, IncidentYN) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_os_incidents) # REPORTED ANALYSIS


# used_for
avg_events_CI_used_for <- dataframe_noIPC %>%
  group_by(u_pattern_type, name, Severity, used_for) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type, Severity, used_for) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_used_for) # REPORTED ANALYSIS

# used_for_os
avg_events_CI_os_used_for <- dataframe_os_noIPC %>%
  group_by(u_pattern_type, name, Severity, used_for) %>%
  summarise(cnt = n()) %>%
  group_by(u_pattern_type, Severity, used_for) %>%
  mutate(avg = mean(cnt), sdev = sd(cnt))
View(avg_events_CI_os_used_for) # REPORTED ANALYSIS
```

F. Data processing scripts - Workflow mining script.R

```
# Read in relevant libraries:
library("dplyr", lib.loc=<Path to R library folder>)
library("tidyr", lib.loc=<Path to R library folder>)
library("lubridate", lib.loc=<Path to R library folder>)
library("reshape2", lib.loc=<Path to R library folder>)
library("RODBC", lib.loc=<Path to R library folder>)
library("ggplot2", lib.loc=<Path to R library folder>)
library("bupaR", lib.loc=<Path to R library folder>)
library("DiagrammeR", lib.loc=<Path to R library folder>)
library("DiagrammeRsvg", lib.loc=<Path to R library folder>)
library("rsvg", lib.loc=<Path to R library folder>)

# Define functions:

shift <- function(x, n){
  c(x[-(seq(n))], rep(NA, n))
}


# This script contains the analyses of metrics related to workflow mining.
# This datset comes from a dump of the main orchestration tool logs, as they have
    been collected for all deploys of IPC within ING.
# To conduct this analysis, we used bupaR, an R library for process mining. The steps
     followed are explained in the script below.

# Note that all 'View' and 'print' statements are commented out in the script. Remove
     the '#' to view or print the appropriate analysis outcomes.
# Also, note that when reading this script you can adapt it by including a View() or
    print() call on any of the created variables, to inspect its contents for yourself
    .

# First, we will read in the workflow logging data

df_wf <- read.csv(<File path>, sep=",", header=TRUE)

# We are interested in successful runs only
df_wf <- df_wf[(df_wf$SUCCESS==1),]

# As there are irregularities in the way some of the variables are stored, we will
    have to re-calculate values in a number of columns
df_wf$FINISHEDTIME <- gsub(",", "", df_wf$FINISHEDTIME)
df_wf$time <- as.numeric(df_wf$FINISHEDTIME) / 1000
df_wf$time <- df_wf$time + (60*60) # Adding this amount represents a shift to GMT+1
    as as.POSIXct belox converts to GMT
df_wf$endtime <- as.POSIXct(df_wf$time, origin="1970-01-01", tz="GMT")
df_wf$starttime <- as_datetime(df_wf$NT.START_TIME.)
df_wf$timediff <- df_wf$endtime - df_wf$starttime
df_wf$VMNAME <- ifelse(df_wf$<configuration item name>=="", as.character(
    df_wf$DEPLOYID), as.character(df_wf$<configuration item name>))
df_wf <- df_wf %>%
  group_by(<configuration item name>) %>%
  mutate(start_next = order_by(starttime, shift(starttime, 1)))
df_wf$stepdiff <- df_wf$start_next - df_wf$endtime
df_wf$stepdiff_minutes <- as.numeric(df_wf$stepdiff) / 60
```

```
# Next, we can read in CMDB data in order to link it with our workflows
deploys_CMDB <- read.csv(<File path>, sep = ",", header=TRUE)
df_wf_servers <- merge(x = deploys_CMDB, y = df_wf, by.x = "name", by.y = <
    configuration item name>, all.x=TRUE)

# Exlcude servers from the CMDB without workflow data
df_wf_servers <- df_wf_servers[(is.na(df_wf_servers$WORKFLOWNAME)==FALSE),]


# Here, we can build in an option to re-run this entire script for non <DEVELOPMENT
    TENANT> related servers only
# df_wf_servers <- df_wf_servers[(df_wf_servers$ref_cmdb_ci_vmware_instance.u_tenant
    != '<Development tenant>'),]


# We can start with a count of the number of unique steps
result <- df_wf_servers %>%
  group_by(u_pattern_type, WORKFLOWNAME) %>%
  summarise(n_distinct())
# View(result)

# Next we can get a list of steps per pattern type with counts
result1 <- df_wf_servers %>%
  group_by(u_pattern_type, WORKFLOWNAME) %>%
  summarise(n())
# View(result1)

# Then, we can create a log for process mining
log_wf <- df_wf_servers[,c("name", "u_pattern_type", "WORKFLOWNAME", "install_status
    ", "starttime", "u_requested_by.x")]
log_wf$id <- 1:nrow(log_wf)


# We will want a separate event log for each of the infra deliveries we are examining
    .
# The for loop below takes each of the infra deliveries, subsets the full event log
    into event logs for each delivery,
# retains a set percentage (the value after 'perc' in the function call below) of
    logs that a workflow step must occur in to be included in a plot,
# and plots a process map showing the flow of cases through the process, annotated
    with the time each step takes on average.
patterns <- unique(log_wf$u_pattern_type)

for(i in patterns){
  log_wf2 <- log_wf[(log_wf$u_pattern_type == i),]
  log_wf2 <- log_wf2 %>%
    eventlog(
      case_id = "name",
      activity_id = "WORKFLOWNAME",
      activity_instance_id = "id",
      lifecycle_id = "install_status",
      timestamp = "starttime",
      resource_id = "u_pattern_type"
    )
  fpath <- paste('C:\\Data\\', as.character(i), '.png', sep="")
  log_wf2 %>%
    filter_trace_frequency(perc = .99) %>%
    process_map(performance(mean, "mins"), render=FALSE) %>%
    export_graph(file_name = fpath, file_type = 'PNG', width = 10000, height = 2000)
}

# We can see from the plots that process flows contain workflow steps related to both
    deployment of a configuration item and decommissioning.
# With the maps of these process flows, we can identify the cutoff point (in terms of
    duration of a step) that signifies the gap between the deployment of the CI
# and the decommissioning for the configuration item. We can deduce breakpoints for
   each process map, and filter the event log for the specific delivery on these
   breakpoints to get
# a flow that represents just the deploy steps. The function below does so. We can
   then use the maps to derive the total running time of a deploy on average,
# and the components used in executing the workflow. These are included as metrics in
    the repository.


for(i in patterns){
  log_wf2 <- log_wf[(log_wf$u_pattern_type == i),]
  log_wf2 <- log_wf2 %>%
    eventlog(
      case_id = "name",
      activity_id = "WORKFLOWNAME",
      activity_instance_id = "id",
      lifecycle_id = "install_status",
      timestamp = "starttime",
      resource_id = "u_pattern_type"
    )
  fpath <- paste('C:\\Data\\', as.character(i), '.png', sep="")
  breakpoints = ""
  <Series of If-statements specifying the names of process steps for each cloud infra
      service, at which break points occur>
  log_wf2 %>%
    filter_trim(start_activities = "Set DeployId", end_activities = breakpoints) %>%
    process_map(performance(mean, "mins"), render=FALSE) %>%
    export_graph(file_name = fpath, file_type = 'PNG', width = 10000, height = 2000)
  times <- log_wf2 %>%
    filter_trim(start_activities = "Set DeployId", end_activities =  breakpoints) %>%
    throughput_time("case", units = "mins")%>%
    summarise(mean(throughput_time), sd(throughput_time), n())
  # print(i)
  # print(times)
  }
```

G. Data processing scripts - Correlation matrix script.R

```
# Read in relevant libraries:
library("Hmisc", lib.loc=<Path to R library folder>)
library("dplyr", lib.loc=<Path to R library folder>)
library("corrplot", lib.loc=<Path to R library folder>)

# Function definitions:
# No functions to define

# Note: You can edit this script to view or print the contents of any defined
    variable by inputting a call to print() or View() at the appropriate point in the
    script.

repository <- read.csv(<File path>, sep = ";", header=TRUE)
repository_corr = repository[,2:21]
repository_sd = repository[,22:ncol(repository)]
repository_nona <- repository_corr[is.na(repository_corr[,1])==FALSE,]
repository_nona_use <- repository_nona[is.na(repository_nona[,2])==FALSE,]
repository_dm <- data.matrix(repository_nona_use)
repository_corr <- rcorr(repository_dm, type="pearson")
#write.csv(repository_corr$r, <File path>)
#write.csv(repository_corr$P, <File path>)
sds <- repository_nona_use %>%
  summarise_all(funs(sd(., na.rm=TRUE)))

# Correction for p-value fishing
pvals <- as.vector(repository_corr$P)
pvals_adjusted <- p.adjust(pvals, "BH", length(pvals))
matrix_adjusted_vals <- matrix(pvals_adjusted, nrow = nrow(repository_corr$P), ncol =
     ncol(repository_corr$P))
#write.csv(matrix_adjusted_vals, <File path>)

# Select variables to retain for report
r_values_report <- repository_corr$r
p_values_report <- matrix_adjusted_vals
dimnames(r_values_report) <- list(<Concatenated names of columns to plot in
    correlation matrix>, <Concatenated names of columns to plot in correlation matrix
    >)
dimnames(p_values_report) <- list(<Concatenated names of columns to plot in
    correlation matrix>, <Concatenated names of columns to plot in correlation matrix
    >)
corrplot(r_values_report, p.mat=p_values_report, type = 'lower', method = 'circle',
    sig.level = .05, insig='label_sig', pch=c('*', '**', '***'), tl.col="black", tl.
    srt=45, tl.cex=.5, pch.cex=1)
pairs(repository_nona_use)
```

H. Descriptive statistics for ING Private Cloud-level metrics

TABLE VI
USAGE METRICS FOR INFRA DELIVERIES

	Deployed servers IPC Overall	Servers deployed past year	Servers active past year
Count	28	28	28
Median	2267	15851612	
Min	11	5	1
Max	22782	10534	8019
Mean	3493.82	2522.50	2376.64
Standard deviation	4285.01	2387.96	2160.21
Standard error of measurement	809.79	398.04	451.28

TABLE VII
COMPLEXITY METRICS FOR INFRA DELIVERIES

	Servers deployed	Avg. deploy duration in mins.	Avg. steps in delivery	Avg. components in workflow
Count	26	26	26	26
Median	1790	35.5	17	9
Min	25	13	12	6
Max	13707	122	28	15
Mean	2487.46	47.23	18.12	8.92
Standard deviation	2592.43	24.32	3.25	1.83
Standard error of measurement	2595.43	24.32	3.25	1.83

TABLE VIII
RELIABILITY METRICS FOR INFRA DELIVERIES

	Count	Median	Min	Max	Mean	St. Dev.	SE
Avg servers with events	26	127	0	1587	274.88	390.92	390.92
Avg events per server	26	12.66	0	195.93	33.2	46.16	8.17
Avg acknowledged events per server	26	1	0	193.66	13.56	40.39	4.44
Avg incidents per server	26	0	0	29.26	4.02	8.18	3.66
Avg events per server severity 0	26	6.26	0	193.89	20.92	44.65	8.01
Avg events per server severity 2	26	1.16	0	88	4.62	17.05	5.05
Avg events per server severity 3	26	0	0	502	87.24	164.86	2.86
Avg events per server severity 4	26	1.84	0	26.87	3.38	5.37	6.33
Avg events per server severity 5	26	4.82	0	79.04	24.01	30.94	6.32
Avg events per production server severity 5	26	0	0	39.11	3.49	8.91	2.85

IX. Survey

A. Survey Set-up

Text Introduction Email

Subject: Survey "How long do we take to develop an Infra service?"

Dear colleague,

Imagine developing an Infra service with the speed of an external cloud provider! Our business partners (devops teams) expect the same experience. They compare us with an external Cloud provider. We all have a gut feeling of how we can make this happen yet we want to support this with numbers using data analysis.

With this survey we want to measure the steps in the developments process that affects the delivery time. A clear picture of common factors helps you to remove obstacles for your future development work.

This initiative is started by the ad hoc Squad I3, which is a collaboration between ING Infra, Core Bank University and TU Delft. For questions, please contact <credentials contact person>.

Click here the following link to open your survey.

<link>

This link will be valid until <expiration date>.

Regards, <name tribe lead>

For more information about our study, see:

<I3 squad repository link>

Demographics

1) Please, select the infra-delivery you have been mostly involved in during the last years. In case you worked on more than one delivery, please select the one you spent most time on. (See list of infra-deliveries in Table **??**).
2) Which of the following best describes your role in <infra-delivery choice> infra-delivery?
   Engineer - Product Owner - Chapter Lead - Architect - Manager (e.g. Area Lead, Trive Lead) - Agile Coach - Other)
   a) Follow-up question: In case you selected Other, please specify your role below:?

Aspects of Infra-Deliveries

3) To what extent do you agree with the following statement?
   "**Consumer ordering** interface related aspects (e.g. setting up ING Private Cloud portal to consume new service) hindered the delivery of <infra-delivery choice>."
   strongly disagree - disagree - neutral - agree - strongly agree - don't know
   a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?
4) To what extent do you agree with the following statement?
   "**Orchestration Workflows** related aspects (e.g. Workflows/Automation for Virtual Machine, Operating System, Network, System Accounts, Storage, Configuration Registration) hindered the delivery of <infra-delivery choice>."
   strongly disagree - disagree - neutral - agree - strongly agree - don't know
   a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?
5) To what extent do you agree with the following statement?
   "**Stack Definition** related aspects (e.g. Capabilities for Backup, APIs, Agents) hindered the delivery of <infra-delivery choice>."
   strongly disagree - disagree - neutral - agree - strongly agree - don't know
   a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?
6) To what extent do you agree with the following statement?
   "**Second Day Operations** related aspects (e.g. Install optional software, SelfService capabilities) hindered the delivery of <infra-delivery choice>."
   strongly disagree - disagree - neutral - agree - strongly agree - don't know
   a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?
7) To what extent do you agree with the following statement?
   "**Operations** related aspects (e.g. Monitoring, Configuration Scanning, configuration management database Model) hindered the delivery of <infra-delivery choice>."
   strongly disagree - disagree - neutral - agree - strongly agree - don't know
   a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?
8) To what extent do you agree with the following statement?
   "**Security, Risk & Compliance** related aspects (e.g. OSG, BIA, Risk Assessment, SEM-I, TSCM-I, Vulnerability

Scanning, Penetration Testing, Certificate Management) hindered the delivery of <infra-delivery choice>."
strongly disagree - disagree - neutral - agree - strongly agree - don't know

a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?

9) To what extent do you agree with the following statement?
"**Service Delivery** related aspects (e.g. Documentation, Service Component Description, Service Description, Service Specification, Training + Instruction Movies) hindered the delivery of <infra-delivery choice>."
strongly disagree - disagree - neutral - agree - strongly agree - don't know

    a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?

10) To what extent do you agree with the following statement?
"**Financial** related aspects (e.g. Procurement, License Metering, Pricing & Charging) hindered the delivery of <infra-delivery choice>."
strongly disagree - disagree - neutral - agree - strongly agree - don't know

    a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?

11) To what extent do you agree with the following statement?
"**Team dynamics** related aspects (e.g. dependencies on other teams, cultural differences, many team changes, age of teams, difference in expertise) hindered the delivery of <infra-delivery choice>."
strongly disagree - disagree - neutral - agree - strongly agree - don't know

    a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?

12) To what extent do you agree with the following statement?
"**Service Verification and Testing** related aspects (e.g. Optimization, Bug Fixing, Test Resources, Test Automation) hindered the delivery of <infra-delivery choice>."
strongly disagree - disagree - neutral - agree - strongly agree - don't know

    a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?

13) To what extent do you agree with the following statement?
"**Governance** related aspects (e.g. Decision-making, Rules & Regulations) hindered the delivery of <infra-delivery choice>."
strongly disagree - disagree - neutral - agree - strongly agree - don't know

    a) Follow-up question: Could you please explain the reason(s) behind your answer choice in the previous question?

14) In your opinion, are there other infra-delivery hindering factors that were **NOT** covered in this survey?
Yes - No

    a) Follow-up question: Please, list any other infra-delivery hindering factors that were not covered in this survey according to your opinion.

B. Aggregated Survey Results

TABLE IX
AGGREGATED SURVEY RESULTS

	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13
Count	27	25	22	24	22	23	23	18	27	24	18
Mean	3.52	3.60	3.00	3.13	3.09	3.13	2.39	3.06	3.67	3.29	3.11
Median	4.00	4.00	3.00	3.00	3.00	3.00	2.00	3.00	4.00	3.50	3.00
Standard Deviation	0.96	1.13	0.90	1.09	1.08	1.12	0.82	1.22	1.15	1.17	0.99
Percent Agree	67%	68%	32%	42%	45%	39%	13%	39%	63%	50%	33%
Top-2-Box	67%	68%	32%	42%	45%	39%	13%	39%	63%	50%	33%
Top-Box	7%	20%	5%	13%	9%	13%	0%	17%	30%	17%	11%
Net Top Box	4%	16%	5%	13%	9%	9%	-9%	11%	30%	13%	11%
Net Top-2-Box	48%	44%	-5%	0%	0%	4%	-52%	-6%	37%	17%	0%
Coefficient of Variation (CV)	27%	31%	30%	35%	35%	36%	34%	40%	31%	36%	32%

TABLE X
TOP TAGS PER SURVEY QUESTION

	Tag 1(#)	Tag 2(#)	Tag 3(#)	Tag 4(#)	Tag 5(#)
Q3	Setting up ING Private Cloud portal to consume the new service (9)	Dependencies on other teams (5)	vRA/vRO issues (4)	-	-
Q4	vRA/vRO issues (5)	Dependencies on other teams (3)	Complexity of the infra delivery (3)	Lack of expertise (2)	Network (1)
Q5	Capabilities of backup (2)	Missalignments in the requirements (2)	Complexity of the infra delivery (2)	Agents (1)	-
Q6	Second day operations are time consuming (5)	Complexity of the infra delivery (4)	Dependencies on other teams (1)	vRA/vRO issues (1)	
Q7	Not unified configuration management database Models and other issues with them (8)	-	-	-	-
Q8	The process of security is too complex, with many documents to fill, get approval etc (6)	OCD (eg. OSG, SEM-I) (3)	Complexity of infra delivery (1)	-	-
Q9	Documentation issues (3)	Complexity of infra delivery (1)	-	-	-
Q10	Pricing/Charging/License (6)	Complexity of infra delivery (1)	-	-	-
Q11	Dependencies on other teams (12)	Lack of expertise (2)	Complexity of infra delivery (1)	-	-
Q12	Bug fixing (5)	Test Resources (3)	Testing is time consuming (2)	Test automation (1)	Lack of expertise (1)
Q13	Decision making (4)	Dependencies on other teams (1)	-	-	-
Q14	Pricing/Charging/Licence (1)	Change of external procedures (1)	Network related issues (1)	Interaction between countries (1)	Onboarding in Ansible tower (1)

TUD-SERG-2018-003
ISSN 1872-5392