UNIVERSIDAD DE LOS ANDES

Faculty of Engineering

Systems and Computer Engineering Department

ArchinotesX: An ArcOps framework

Thesis work presented by

Lorena Salamanca Rojas

Advisor

Dario Ernesto Correal Torres Ph.D.

To opt for the title of

Master in Software Engineering

November 2016

Further information

ArchinotesX full source code, documentation and deployment instructions can be found at

https://github.com/imTachu/ArchinotesX

Acknowledgements

I would like to thank Juan Sebastian Urrego whose previous work is the base for ArchinotesX. I would also like to thank Juliana Davila, whose research and support mostly in the implementation process was crucial for this investigation project. Finally and mainly, I would like to thank Dario Correal, without his guidance this project just couldn't happen, he is so immerse in the entire investigation that from now on I'll write in plural.

Table of Contents

List of Figures

List of Tables

1

Introduction

Software architecture is the structure or structures of a system which comprises software elements, externally visible properties of those elements, relationships among them and boundaries or constraints that have to be kept in mind to enable the system to fully satisfy business requirements [3]

Defining a software architecture is a fundamental activity that allows to have an initial high level technical view of the components that will be build or integrated as well as the relation among those components and their environment [4], all of this has to be conceived before any implementation or design investment. Furthermore, software architecture also keeps in mind non functional but vital aspects of a system such as: Availability, integrability, security, performance, usability, accessibility, etc [5].

Currently, there exists a wide portfolio of methodologies, philosophies, and frameworks to create and manage software and independent of their methodology

differences, they recognize in a greater or lesser value that software architecture is a fundamental stage in a system, e.g., the Waterfall model proposed by Winston W. Royce in 1970 [6] divides the life-cycle in interdependent phases i.e., sequential phases, in which the first phase is "System and software requirements", where business requirements are analyzed in order to determine key features and main goals of the system. On the other hand, Scrum currently recognizes an initial phase (Or sprint zero) where a base architecture is defined by analysing preliminarily the priority of business requirements and its main quality attributes [7] [5]. Traditional and agile methodologies propose different ways to achieve the common goal of building a functional efficient system acknowledging software architecture as a key activity.

Nevertheless, even though software architecture importance is well recognized, there is a growing gap between the state of the art and the state of the practice in software architecture [8]. The state of the art investigates and advances towards advanced architecture description and modeling languages, while in practice, software architecture has neither the power nor the tools to evolve according to the software and therefore to be useful throughout the entire software life cycle [8].

1.1 Problem statement

In a fast moving digital world that changes constantly, software systems need to be in the capacity to evolve rapidly in order to fulfill new trends, new features, etc. Unfortunately, most of the systems are conceived to satisfy just a limited set of features because when they begin are predicted to be small enough just to cope with initial requirements [8]. In most cases there is no documented software architecture at all perhaps because of the rush to produce valuable assets plus, he though that a project can survive without much upfront design [8].

Back in 1974, Dr. Meir Lehman started researching about evolutionary software systems and formulated a series of empiric laws that describe the balance between forces that support new developments and forces that slow down progress on development projects. Second law, the "Increasing Complexity" law states that

"as an E-type system evolves, its structure tends to be more and more complex unless work is done to maintain or reduce it" [9]. This law supports the idea that a system's architecture will be more complex whenever new features should be added and that it will require more effort to add a new functionality.

Since a software architecture is the asset that the clients cannot see, there is typically no cost-effective way to continuously keep the implementation in sync with the architecture artifacts. "As the software undergoes changes due to new requirements, it increasingly deviates from the intended architecture" [8]. This causes that the software can become very hard to maintain since the real state of its base is unknown. Understanding how new requirements might impact the software or if it will be incompatible with any existing component becomes very difficult. The problem is that even if the source code is written with very high quality, it does not imply a high-quality software architecture with decoupled layers, components, and rules governing how software entities are structured and how they communicate with one another.

This lack of an explicit connection between the software architecture and the source code is difficult to overcome and makes it hard for anyone to understand exactly how the software system works and where changes belong [8]. To this situation is added the fact that normally software architects have more than one functionality on their mind and just as software architecture artifacts, they are outdated in relation to the real software in production [10] [8] [11].

A similar problem has been already tackled down, if development and operations staff work uncoordinated at some point, business will pay for the increased difficulty that the communication gap between these teams has arised. DevOps creates a tight relation between development and operations teams, aligning the development of the source code and its deployment into production, (More of this concept in chapter 2), however the software architect is still out of this coordinated flow. For a non-DevOps environment is hard to keep all the artifacts aligned because during bugs, hotfixes, new requirements, there is not a clear coordination flow among any of the artifacts [10] [8] [11] [12] [8] [13].

In concrete, our problem statement is that even though DevOps creates a

constant coordinated flow between development and deployable artifacts. However, the software architecture remains outdated and therefore is not clear what the real state of a system is in a certain point of time causing a disconnection between strategy and systems.

1.2 Contribution

Complementary to the DevOps approach, we state that the software architect needs to be closer to the development and deployment processes, i.e., the software architect should be invited to that coordinated-close relation between development and operations teams. "ArcOps" is the term we coin for this approximation. As seen in Figure 1.1, ArcOps is an intersection not only between Development and Operations teams, but also with software architects.

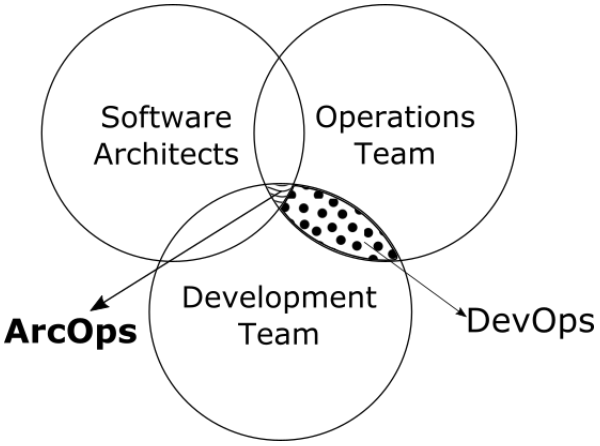

Figure 1.1 ArcOps: An extension of DevOps

ArcOps purpose is to create a continuous work-flow between architects, developers and system administrators aligning all the artifacts in a system. This will grant software architects full governance of their projects delivering new features and updates while keeping projects architcture artifacts up-to-date. DevOps gives robustness to the functionality of a project, ArcOps enhances this "robustness" by giving a fully updated structure definition of the system.

Along with the ArcOps approach comes ArchinotesX, a framework that allows the ArcOps execution: An architecture will be modeled over the functional and

the deployment viewpoints [14] defining the 'what' and 'where' of its definition. In a deployment view on ArchinotesX, when a component (a webservice) is assigned to an infrastructure node it will actually deploy the service: Webservice A will be deployed in X machine. When an update is required, the related component is updated, when this happens, architecture, code and deployment are consistent.

ArchinotesX is oriented to the explicit scenario where there are multiple data sources and the business need is to constantly create and update services or even microservices that will bring specific information from those sources. This services will be orchestrated by reacting to heterogeneous devices requests at any time. The main goal is to be able to obtain data from different sources in order to rapidly fulfill business needs: New products launching, data analytics or just anything that needs data access. Typically, a company has multiple data sources (Relational and non-relational databases, Excel files, etc) that will be accessed over and over again in order to bring relevant business information for different projects; this operation is our main focus. ArchinotesX was validated by a case study in a telecommunications company; this allowed us to identify the real impact that an ArcOps approach can have in the industry as well as its advantages and disadvantages.

1.3 General objective

To define a concept that extends the principles of DevOps towards software architects in order to create a more fluent work-flow along the entire life-cycle of a software.

1.4 Specific objectives

- To define an extensible value proposition of ArcOps taking into account the core values of DevOps.
- To design and develop a tool that will allow us to test the value proposition of ArcOps.

- To test the ArcOps concept and the tool in a real industrial environment.
- To analyze the results obtained during the validation process and deduce if there is a perception change after explaining ArcOps and using the tool.

1.5 Document structure

The remainder of this document proceeds as follows: Context is presented in Chapter 2. Chapter 3 presents the Solution Strategy followed to solve the aforementioned problem statement and the objectives of this research work that will also be named in the same chapter. Chapter 4 defines and shows the evolution across three different strategies: Traditional, DevOps and ArcOps. Chapter 5 presents the development process of ArchinotesX, here we enunciate the technologies that were used and also how the project was developed. Chapter 6 presents the validation process and the analysis of the results of ArchinotesX in a telecommunications company. Chapter 7 introduces some approximations that have been developed to solve partially the problem presented in this research. Finally, we present our conclusions as well as future work in Chapter 8.

2

Context

This chapter introduces core definitions that are used throughout the document and are included here since those are the concepts we relied upon to base our research.

2.1 Software Architecture

Software architecture refers to the basic structure of a system and although it has been widely defined, we took two of the most known definitions: Philippe Kruchten refined a previous software architecture definition stating that "Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements

into larger subsystems; and an architectural style that guides this organization" [15] This definition recognises the power of the software architecture across structure and behavior. Bass et al. states that "the software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them" [3], and this means that an architecture is an abstraction of a system that shows the relation of its elements. Yet another definition is that "Architecture is the set of design decisions that must be made early in a project" [16] which means that architecture decisions should be taken seriously since they affect every aspect of the system. There are many definitions of software architecture but what really matters is to understand that regardless of the nature of the system a poor architecture choice can be prohibitively expensive on large systems [13] [17] [18] [19].

2.2 Architecture views and viewpoints

Just as there are several software architecture definitions, there are multiple ways to adopt it however, there are two common concepts across these adoption approaches that are presented in the ISO/IEC/IEEE 42010:2011 Systems and software engineering architecture description [20], viewpoints and views.

- **Viewpoint:** Work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns.
- **View:** Work product expressing the architecture of a system from the perspective of specific system concerns

In other words, a view is a representation of a whole system from the perspective of a related set of concerns and a viewpoint defines the perspective from which a view is taken. For example Kruchten defines "4+1 View Model of Software Architecture" [15] that demarcates four main views, logical, process, physical, development and a final view that relates all the aforementioned, the

scenarios view, although it is a well known approach, ArchinotesX was developed following Rozanski and Woods proposition [14]: The one thing an architect should strongly avoid doing or even thinking is to try to answer all of an architecture questions by means of a single, heavily overloaded, all-encompassing model. More specifically, "It is not possible to capture the functional features and quality properties of a complex system in a single comprehensible model that is understandable by and of value to all stakeholders" [14]. They created a catalog of viewpoints that can be used in order to address the concerns across the architecture and can be seen in Figure 2.1:

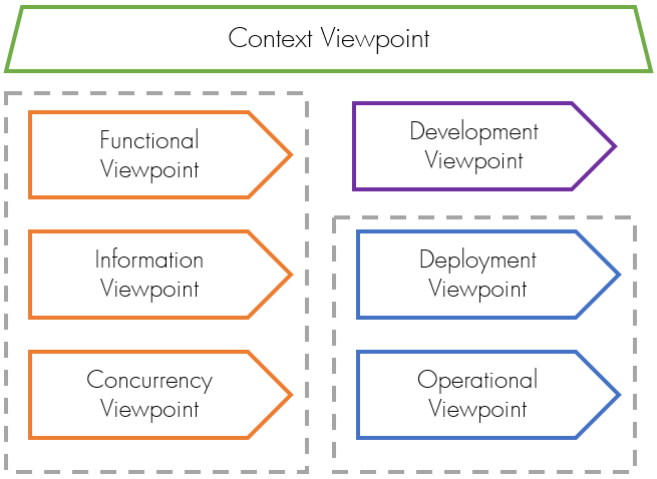

Figure 2.1 Rozanski and Woods viewpoints

- **Context:** "Describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts)."

- **Functional:** "Describes the system functional elements, their responsibilities, interfaces, and primary interactions."

- **Information:** "Describes the way that the architecture stores, manipulates, manages, and distributes information."

- **Concurrency:** "Describes the concurrency structure of the system and maps functional elements to concurrency units to clearly identify the parts

of the system that can execute concurrently and how this is coordinated and controlled."

- **Development:** "Describes the architecture that supports the software development process."
- **Deployment:** "Describes the environment into which the system will be deployed, including capturing the dependencies the system has on its runtime environment."
- **Operational:** "Describes how the system will be operated, administered, and supported when it is running in its production environment."

In the development of ArchinotesX we included the Functional, Information and Deployment viewpoints.

2.3 Service Oriented Architecture

The entire base of ArchinotesX are web-services and how they are operated, we rely on the principle that web-services are totally independent from each other. Service Oriented Architecture or SOA 1.0 is "an application framework that takes everyday business applications and breaks them down into individual business functions and processes, called services. An SOA allows to build, deploy and integrate these services independent of applications and the computing platforms on which they run" [21]. But then, an SOA is also a way to design, implement, and assemble services to support or automate business functions [22]. An SOA can be seen as a hub of software connections that will itself allow the creation of new and more robust components, a centralizer and integrator of web-services [23].

We selected web-services and SOA as our core concept, the one that will be the center of the development in order to correctly validate our hypothesis because it is widely known, standardized and more important, very useful for a company because if well approached, an SOA can be the unique entry point to a company datasources, by having a single entry point IT has to worry only about

security, performance and reliability of the web-services connecting to the data of the company rather than multiple entry points that could lead to security and performance issues in the future [24].

2.4 Monoliths and Microservices

There is a concept that is newer than SOA, a concept that extends and bases in many of the SOA principles. In 2013, Martin Fowler started writing about microservices, a component-way to see web-services stating that a component is a unit of software that is independently replaceable and upgradeable. At a high level, "the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies" [25]. While a microservice architecture complements the Service Oriented Architecture development phase by reducing the need for explicit coordination of teams, its adoption introduces new challenges for the companies because if microservices are not designed carefully they will bring performance, availability or maintainability issues [1] [26].

An important characteristic of microservices is that they are organized around business capabilities, i.e., the distribution of microservices teams should be cross-functional including the full range of skills required for the development: user-experience, database, and project management. Software architecture is generally a human construction and most wise architects understand the power of Conway's Law [27] [25]. "Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure." [28] which is just what we said before: A company has different profiles of people with different skills, level of expertise, knowledge, just like that should be a microservices team.

2.5 DevOps

Since ArcOps is an extension of DevOps, it is very important that this concept is fully clarified and understood. Len Bass et al. defines DevOps as "a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality" [1]. Basically, DevOps extends the agile principles to the software delivery phase that sprang from applying newer agile [29] and Lean [30] approaches to operations work. It also focuses on the value of collaboration between development and operations staff throughout all stages of the development life-cycle when creating and operating a service [31], it shows how important operations has become for development and of course, how important development is for operations [32] [33] [34]. As seen in Figure 2.2 there is an infinite loop among development activities (blue) and operations activities (yellow).

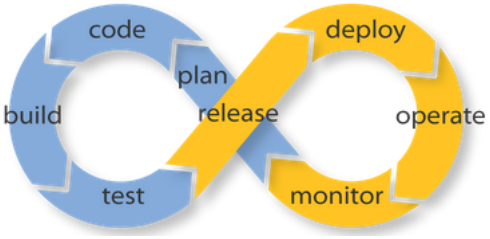

Figure 2.2 The DevOps loop [2]

SOA, DevOps and microservices are facets of the same idea: Breaking software into small, manageable pieces, developed independently and in parallel, enabling large companies to move at the speed of the small ones. Organizing the company and the development teams around microservices results in very small autonomous teams responsible for one or several microservices. These teams make their own technology and methodology choices resulting in high efficiency when it comes to improving them or fixing defects [31].

2.6 Executable Architectures

As defined earlier, a software architecture is the piece of a system on which everything relies on, its main definitions, relations, constraints, functionality etc, are structured in it. When creating a software architecture one has to think with a certain level of abstraction that enables relatively rapid analysis of alternative design concepts [35] [36]. However, current frameworks, such as the Department of Defense Architecture Framework (DoDAF) [37], prescribes only static representations of the architecture description. Furthermore, software architects neither have time to spend developing full architecture descriptions nor the expertise to convert the architecture description into an executable model -executable architecture- that can be used to evaluate those 'abstract thoughts' [38]. But the need to assess the characteristics and behavior of architecture descriptions to determine their suitability to meet user needs requires tests that go beyond those feasible with just a static representation [39] [40].

Software architecture provides an understanding of different aspects of a system. Its structure, behaviour, deployment and development environment are met as well as the context that the system will have to interact with [41]. When a software architect can execute an architecture [42]:

- The effectiveness of the planned technologies and components can be tested for the defined scope.
- Bottlenecks are early identified.
- The architecture can be stressed to see how well it addresses business requirements.
- Integration with existing components can be tested in order to avoid rebuilding.
- Architecture models will be always updated.

2.7 Docker

This and next subsection will explain some core concepts of the technical part of ArchinotesX, these are tools/technologies used to increase the performance of the development and validation exercises of ArchinotesX.

Docker is an open-source project that automates the deployment of Linux applications inside software containers. Quote of features from Docker web pages: “Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries, etc. This guarantees that it will always run the same, regardless of the environment it is running in” [43].

Docker provides an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Docker uses the resource isolation features of the Linux kernel to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.

2.8 Data Center Operating System, a.k.a., DC/OS

DC/OS is a distributed operating system based on the Apache Mesos distributed systems kernel. It enables the management of multiple machines as if they were a single computer.

An operating system abstracts resources such as CPU, RAM, and networking and provides common services to applications. DC/OS is a distributed operating system that abstracts the resources of a cluster of machines and provides common services. These common services include running processes across a number of nodes, service discovery, and package management, it automates resource management, schedules process placement, facilitates inter-process communication, and simplifies the installation and management of distributed services. It allows to deploy Docker and Mesos containers with Marathon, the production proven container orchestrator.

DC/OS is a project built out from different projects, DC/OS has several

components however, we will briefly explain its core concepts, the basic part that was used in the implementation of ArchinotesX.

2.8.1 Apache Mesos

Apache Mesos is an open-source cluster manager that provides efficient resource isolation and sharing across distributed applications, or frameworks. The software enables resource sharing in a fine-grained manner, improving cluster utilization. Apache Mesos abstracts CPU, memory, storage, and other compute resources away from machines (physical or virtual), enabling fault-tolerant and elastic distributed systems to easily be built and run effectively [44].

Mesos uses a two-level scheduling mechanism where resource offers are made to frameworks (applications that run on top of Mesos). The Mesos master node decides how many resources to offer each framework, while each framework determines the resources it accepts and what application to execute on those resources. This method of resource allocation allows near-optimal data locality when sharing a cluster of nodes amongst diverse frameworks.

2.8.2 Marathon

Marathon is a production-proven Apache Mesos framework for container orchestration. DC/OS is the easiest way to start using Marathon. Marathon provides a REST API for starting, stopping, and scaling applications. Marathon is written in Scala and can run in highly-available mode by running multiple copies. The state of running tasks gets stored in the Mesos state abstraction. It can launch anything that can be launched in a standard shell.

Marathon is a framework for Mesos that is designed to launch long-running applications, and, in Mesosphere, serves as a replacement for a traditional init system. It has many features that simplify running applications in a clustered environment, such as high-availability, node constraints, application health checks, an API for scriptability and service discovery, and an easy to use web user interface. It adds its scaling and self-healing capabilities to the Mesosphere feature set. Marathon can be used to start other Mesos frameworks, and it can also launch

any process that can be started in the regular shell -such as Docker containers-. As it is designed for long-running applications, it will ensure that applications it has launched will continue running, even if the slave node(s) they are running on fails.

2.8.3 ZooKeeper / Exhibitor

Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination. ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them ,which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

ZooKeeper allows distributed processes to coordinate with each other through a shared hierarchal namespace which is organized similarly to a standard file system. The name space consists of data registers - called znodes, in ZooKeeper parlance - and these are similar to files and directories. Unlike a typical file system, which is designed for storage, ZooKeeper data is kept in-memory, which means ZooKeeper can achieve high throughput and low latency numbers.

Exhibitor is a supervisor system for ZooKeeper. Exhibitor attempts to present a single console for an entire ZooKeeper ensemble. Configuration changes made in Exhibitor will be applied to the entire ensemble. Each Exhibitor instance monitors the ZooKeeper server running on the same server. If ZooKeeper is not running, Exhibitor will write the zoo.cfg file and start it. If ZooKeeper crashes for some reason, Exhibitor will restart it.

3

Solution Strategy

In this chapter we introduce the strategy followed to achieve the objectives that were presented in Chapter 1.

3.1 Solution process

We designed a solution process that we followed to solve the problem presented in Chapter 1. In this section we will explain the steps involved in the mentioned process.

- *ArcOps concept conception:* We started by realizing the impact and great contribution that DevOps has had in the industries in the latest years, then we started to think that since the role of the software architect is so important, they shouldn't be left out of the coordinated work-flow that DevOps has proposed for developers and operations teams. We studied

deeply the core values of DevOps and translated them into a more software architect focused environment.

- *Scope and functional requirements definition:* When we defined what we wanted to solve with the concept of ArcOps, we started thinking on what would be the most suitable and delimited scenario where we could validate if this new concept could have a real impact in the industry. We noticed that in general, around any IT team a basic concept is always known: Web services and in a more updated context, microservices. We defined that microservices were going to be the atom of the developed tool, more specifically, we identified that the most used microservices (or web-services) are those that are required to get information from a datasource and present it to other microservices or web frameworks or mobile applications. Concretely, our base unit are GET microservices. The other key point we thought is that due to the delivery time of this research work and that there is no real team involved, we needed to emulate the role of the software developer with a code generation tool. We decided that we should create a data-microservice code base that will receive only some dynamic values in order to make it independent and reliable to execute and easy to develop.

  As in any software project we had an extensive sheaf of requirements that we wanted to include but since the time was limited, we had to prioritize what we thought could bring more value to deliver to the software architects which are: (1) Administer the connections to their datasources keeping consistency that if a data-microservice is using a datasource it cannot be deleted, (2) Administer data-microservices by having an updated view of the state of every data-microservice, (3) a notification module that will alert the software architects involved in a project if something wrong happened to a data-microservice allowing them to take action very quickly.

- *Architecture design and technologies selected:* Chapter 5 describes all the technologies used in the development of ArchinotesX, the justification to

choose those technologies obeys a learning curve factor only; for the client we chose the technologies that were the most natural for us in order to reduce the time for learning new languages or frameworks to develop ArchinotesX.

- *ArchinotesX development:* ArchinotesX development process was started in June of 2016 with an average of 12 hours per week dedication, its source code is hosted in a GitHub public repository. While developing ArchinotesX we had a discussion whether if ArchinotesX was a tool or a framework, then we found that in 1999 Dirk Riehle define a software framework as "A universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate development of software applications, products and solutions." [45] With that definition we cleared out the discussion and we stated that ArchinotesX is not a tool but a framework.

- *ArcOps and ArchinotesX experimentation:* Chapter 6 describes the process followed to verify the suitability of ArcOps and ArchinotesX to the hypothesis and mentioned software architects problems. We decided to validate our proposals in a commercial environment, more specifically in a telecommunications company. The main purpose of the validation process is to realize if there is a perception change before and after using ArchinotesX.

- *Results processing and analysis:* Finally we collected the results obtained from the validation process and analyzed them. Chapter 6, also contains this analysis.

In our ArcOps proposal, we identified three main roles that will be taken into account for the entire software creation process:

- **Architecture Team**. The software architecture team capture business needs, define technical requirements and generate models and guidelines for the rest of the project.

- **Development Team**. The development team receives the project definition and creates the software that will fulfill the business needs. In this

phase bugs or defects should be discovered and fixed.

- **Operations Team**. The operations team installs, migrates, supports, and maintains the complete system.

We acknowledge the importance of QA members, testers, business stakeholders, but we delimit our interest to the three mentioned teams.

The ArchinotesX implementation as a whole pretends to lightly implement and test the concept of executable architectures, this project will give us a more precise approximation whether if the concept of ArcOps is really useful in the industry or not.

3.2 From Waterfall to DevOps to ArcOps

As explained in Chapter 3, we acknowledge the importance of different areas and teams across the IT environment but we focus on the architecture, development and operations teams. IT staff can be organized as an independent universe in every company, different teams using different methodologies, but we will take into account three of the strategies that we studied to get to ArcOps. We also define the relation between the three teams by named arrows (Arc-Dev, Dev-Ops, Ops-Arc), showing the bonding between them.

3.2.1 Traditional strategy

The waterfall software development lifecycle describes a linear-iterable approach: It sequentially completes each phase before moving on to the next phase [46]. This classical methodology defines several phases (Requirements, Design, Implementation, Verification, Maintenance) [47], in Figure 3.1 we can see that every time a phase finishes there is a break-time, it is caused by the learning curve the different teams need to make in order to take control of the project [48]. Also, the communication only goes forward: The arrows Arc-Dev and Dev-Ops do not come back. In a refactor/update case, changes are notified to development or operations depending on the case, but rarely the artifacts of the architecture

team are kept in count. The more changes are developed and deployed, the more outdated the original definitions and designs will be.

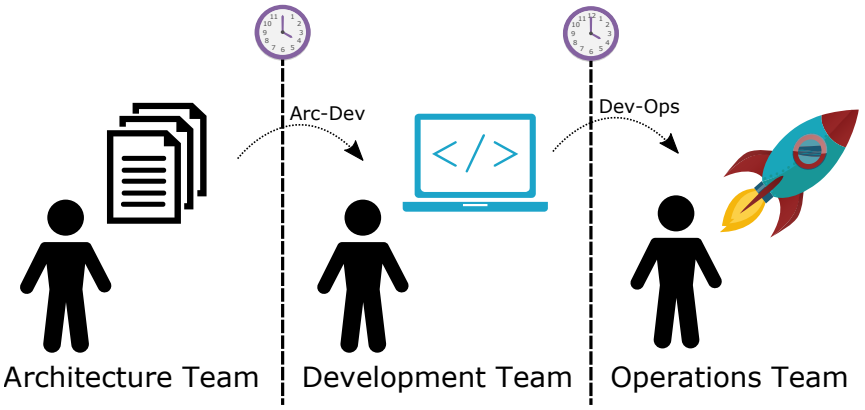

Figure 3.1 Traditional Strategy

The biggest flaw of this model is that due to its rigidity, requirements should be clearly defined up-front in order to create a healthy flow across the phases. i.e. if requirements are not well defined in the very beginning, the rest of the phases will fail to accomplish the project's mission [49]. Also, since the waterfall model defines the project release date at the start, the product is delivered only at the end of the project timeline. Studies have shown that about 60% of the initial requirements are changed during the progress of the project, using a waterfall approach makes it hard to handle this moving requirements since there is no turning back between its phases [50].

3.2.2 DevOps strategy

Agile development processes e.g. Scrum, Kanban, XP are driven by small software development iterations that will deliver a fully functional portion of the final product in order to rapidly receive feedback from the customers, this creates a continuous work flow between development and operations teams[51]. The aim of the DevOps approach is to fill the gaps between development and operation

teams since it engages the development staff in operational activities automation taking full responsibility of the system's performance. In Figure 3.2 we can see how the time intervals between development and deployment disappear. This time both the development and deployment phases are automated thanks to the progress in continuous integration and continuous delivery processes, this whole movement creates an endless loop (Dev-Ops and Ops-Dev relations) that will keep code and deployment up-to-date.

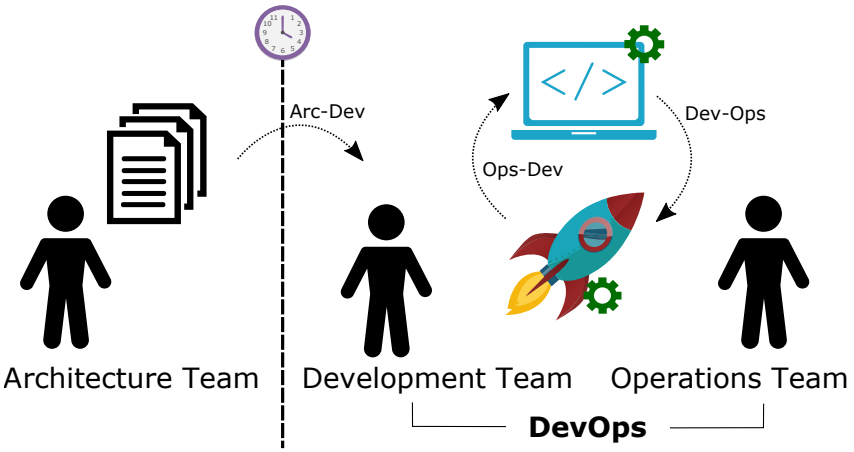

Figure 3.2 DevOps Strategy

Although DevOps clearly tackles nowadays continuous delivery tasks, there is still a gap between Architecture and DevOps since the architecture team needs to bring development and operations teams under a single governance process and there is a break-time while the DevOps team understands software architecture definitions. In the concrete case where the identified needs are different web-services accessing several data-sources, we think this time can be avoided.

3.2.3 ArcOps strategy

We add an execution strategy to software architecture in order to remove the aforementioned gaps. As seen in Figure 3.3, the software architect is now included in the relation loop joining the Development/Operations cycle of DevOps. The

initial models and definitions of the software architecture will evolve during the entire life cycle of the application, meaning that the software architecture model is not just an abstraction but is in fact, the solution.

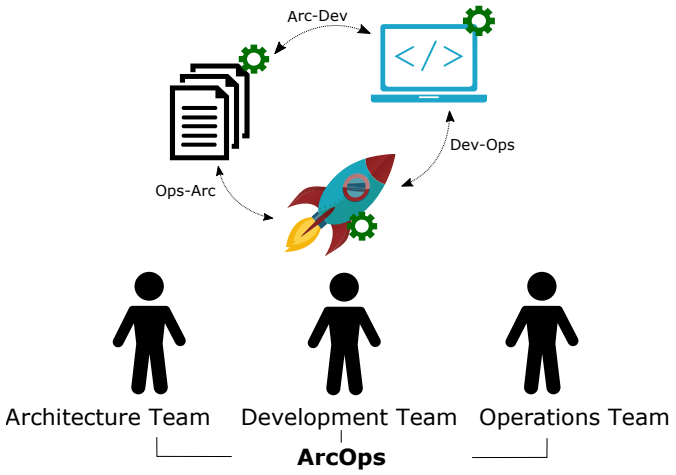

Figure 3.3 ArcOps Strategy

In this case, what the architect defines is what is executed and therefore, operated. This proposal provides a controlled environment where the architect will design suitable solutions and this solutions will have an immediate impact in the business operation allowing to experiment and stress the architecture foreseeing its real behavior.

The strategy of ArcOps aim is to extend the tightened relation between development and operations teams towards the architecture team, in order that whenever an application evolves, its architecture evolves too allowing the architect to invest more time and effort analyzing, designing, integrating and building new products and features and not worrying about the outdated architecture documents, it is a coordinated automated loop among the three teams.

As we mentioned earlier, we aim to validate the hypothesis by developing ArchinotesX, a software that contains the main principles of ArcOps. In Figure 3.4 we want to show exactly what we want to achieve by means of ArchinotesX.

In ArchinotesX we have two big components: The client and DC/OS. The client is in charge of creating, updating, deleting data-microservices and connecting them to a certain datasource. These actions, will have an immediate

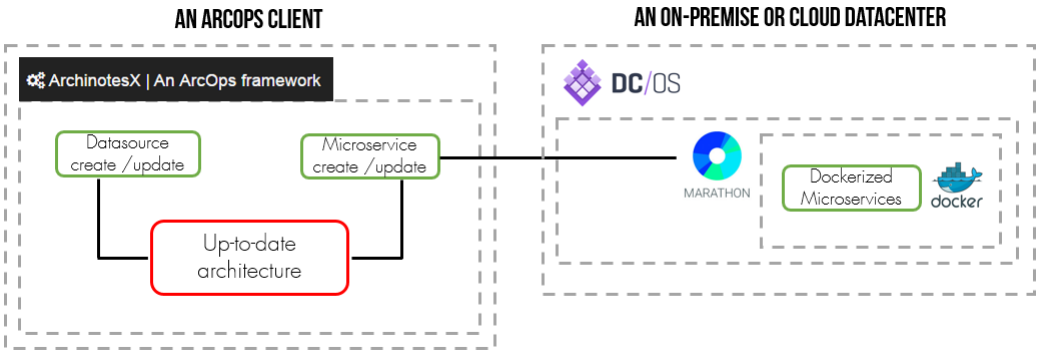

Figure 3.4 ArchinotesX Strategy

impact in the architectural views, everytime a datasource is created it will be included in different architectural views and every action around it will also be included: Let's think that we create a Datasource A and then we create a Data-Microservice AA, whenever one wants to check the architectural views, the datasource, the data-microservice and their relation is already included. In the same way when a data-microservice is created, it will deploy a fully functional dockerized-microservice deployed as a service in Marathon inside DC/OS, the ArchinotesX client will be checking every 5 seconds the state of the microservice and will reflect it visually in the client, but also, it will have an impact in the assignation and functional views as well. Finally all of this will end up generating our main goal: An up-to-date architecture.

4

ArcOps

The word "ArcOps" is a combination of the words "architecture" and "operation". This wordplay gives us a hint of the basic nature of the idea behind ArcOps. The origin of the word ArcOps is born from the identified necessity to expand the same principles that DevOps encompasses, towards software architects. Just like DevOps, ArcOps is a practice where collaboration between different disciplines of software development is encouraged.

Since ArcOps is an extension of DevOps, it has the same roots, they are branches of the agile software development principles. The Agile Manifesto was written in 2001 by seventeen independent-minded software practitioners that had the aim to improve the then current status quo of system development and find new ways of working in the software development industry. The following is an excerpt from the Agile Manifesto with the four core principles:

- Individuals and interactions over processes and tools

- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The Agile Manifesto does not denies the importance of the items on the right, but states that there is more value in the items on the left. DevOps and now ArcOps are related to the first principle, since what we aim with ArcOps is to improve the relation and coordination between now three different teams of a company: Architecture, development and operations.

4.1 What is ArcOps?

Branching out and extending the definition of DevOps [52] we can concretely define ArcOps as:

- **ArcOps definition**

  ArcOps is a development methodology with a set of practices aimed at bridging the gap between Architecture, Development and Operations, emphasizing communication and collaboration, continuous integration, quality assurance, delivery with automated deployment and an automated update of software architecture artifacts

4.2 Why ArcOps?

- *Increased effectiveness.* In a typical IT environment there is always a time-waste lapse where people are waiting for other people, other machines, new software or they are stuck solving the same problems over and over. Automated deployments and standardized production environments, key aspects of DevOps, make deployments predictable and release people from routine repetitive tasks to go do more creative things. ArcOps now realizes that one big time-waste lapse for software architects happens while trying to understand what is going on in a deployed architecture.

Let's think in this scenario: Back in 2012 there was a business requirement, in that time, the software architect created a document containing the technical specification of the business requirements, this document was handed to the software development team in order to code that business requirement, by that time also, after the development phase, the compiled or the deployable artifact was handed to the operations team that followed a "Installation manual" document that contained instructions of how to install in production environment that deployable artifact. As the time passes the business requirements evolved creating the necessity of (1) Updating or adding new features to the source code and (2) Deploying a new artifact in production that will fulfill this evolution; the software architect kept in mind all those changes and due to time-to-market and new requirements, the architecture document was never modified, this situation can happen several times and everytime it happens there is a bigger breach between the reality of what is happening in the business i.e., in production and what is written in the system architecture.

With an ArcOps approach, software architecture artifacts are totally coordinated with the flow between source code and deployables in production, and this coordination can cause that new members of a team can start delivering value to the company instead of spending time understanding what is real and what is not in an IT environment.

- *Shorter Development Cycle.* ArcOps promotes a culture of increased collaboration and communication between the architecture, development and operations teams. This translates into shorter time-frames to move from documented and designed architecture to engineering code into executable production code.

- *Improved Ability to Research and Innovate.* By fostering a culture of high trust between team members of the architecture, development and operations teams and by reducing the time that a new feature or bugfix will take to be analyzed, developed and deployed it is possible to open up a space to

research newer customer needs and innovate accordingly to address those needs. A metric to track here would be to see the number of innovations that have resulted, before and after, moving to DevOps.

4.3 Coordination among three teams

One goal of ArcOps is to minimize the coordination effort in order to reduce the time-to-market. Len Bass et al. [1] defines different coordination mechanisms in DevOps that are also applicable in ArcOps.

4.3.1 Forms of coordination [1]

- *Direct.* The individuals coordinating know each other (e.g., team members)
- *Indirect.* The coordination mechanism is aimed at an audience known only by its characterization (e.g., system administrators).

Coordination mechanisms can be already implemented into many of the tools used in DevOps. For example, a version control system is a form of automated coordination that keeps various developers from overwriting each other code [1]. ArcOps, since is a new concept, doesn't have a dedicated tool, however, even in traditional methodologies team members have found different ways to share uncoordinated software architecture artifacts e.g., Google Drive, Dropbox, Sharepoint, FTPs, etc. The ideal characteristics of a coordination mechanism are that it is low-cost in terms of delay, preparation required, and people time, and of high benefit in terms of visibility of the coordination to all relevant stakeholders, fast resolution of any problems, and effectiveness in communicating the desired information [1] [53] [54].

Concretely, our ArcOps proposal can be seen in Figure 4.1 we are thinking about the functional and non-functional requirements of a project, this part is not contemplated here since the business needs are outside the boundaries of software architecture, whenever those requirements are well defined, our approach is to automatically generate dockerized-microservices that will be created from

the architecture itself, since the architecture is executable it has the power to generate microservices on demand but it also has the ability to keep the main architectural views synchronized.

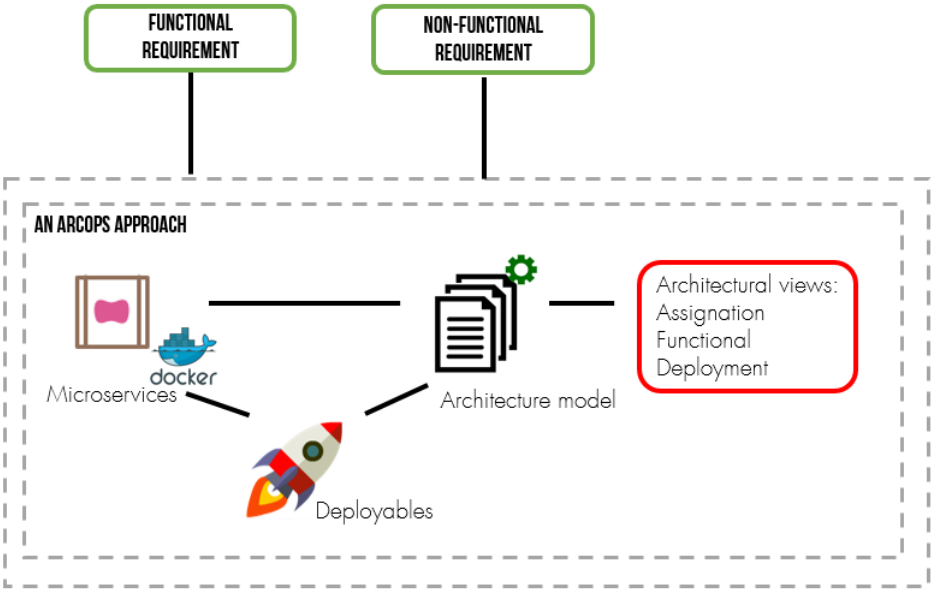

Figure 4.1 An ArcOps approach

We want to loop the relation of the source code, i.e., the dockerized-microservices, its deployment and the architecture views in order to generate a dependency among these three components.

5

ArchinotesX

ArchinotesX is an implementation of our ArcOps approach and it is also an extension of Archinotes [55], a tool that allows users to build architectures collaboratively via web, the goal is to facilitate the software architecture design process for distributed agile software development methodologies. But, even if Archinotes solves the problem of collaboratively designing architectures, there is a gap between the architecture design artifacts and the artifacts that are really deployed i.e., the software that is in production. We decided to reduce this gap by extending the tool.

5.1 Architecture

ArchinotesX uses microservices in order to make possible to design, test and deploy multiple software and hardware artifacts independently and taking into

account the single-responsibility principle [56], it also uses a distributed operating system called DCOS that will handle the resources across the infrastructure in order to have high-availability and the possibility to give or take hardware resources in run-time for the data-microservices. Our architecture can be described in the relation of three components: Datasources, the ArchinotesX Client and DCOS.

The Datasources component are the databases where the company has its data stored; this is the only component that does not belong to this project, it is understood that a company, independently of the technologies or the methodologies that implements relies on a database to store its data.

The ArchinotesX Client has two layers: The *datasources layer* performs a connection to all the datasources that want to be used in one or more data-microservices, it checks whether a datasource can be reachable as well in the client as in DCOS and performs a search in order to get its tables so it can be feasible to deploy a data-microservice connected to the selected datasource. In the *data-microservices layer*, we can create data-microservices selecting a datasource and a table from that datasource, from here the client will send a command to DCOS that will actually create and deploy a data-microservice with the selected configuration, generating an */api/<table>* like endpoint. The main purpose of data-microservices is to extract and exhibit data from different sources. The idea is that the architect can define which datasources are needed in order to generate a data-microservice that will present its data. Each data-microservice is capable to extract data from a SQL databases that is already linked to the ArchinotesX client. The data-microservices can be designed and deployed in run-time and will be atomically deployed at anytime, without incurring in development hours, through code generation.

As seen in Figure 5.1 there is a datasource connector in the ArchinotesX Client per datasource but there can exist several data-microservices per datasource. For example, let's say that there is a PostgreSQL database that has 20 tables created, in ArchinotesX should exist one connection to that datasource but there can exist up to 20 data-microservices (one per table), whenever a data-microservice is

created in the ArchinotesX Client, it will generate a data-microservice in DCOS, this is an actual testable components in it exposes an endpoint for each data-microservice.

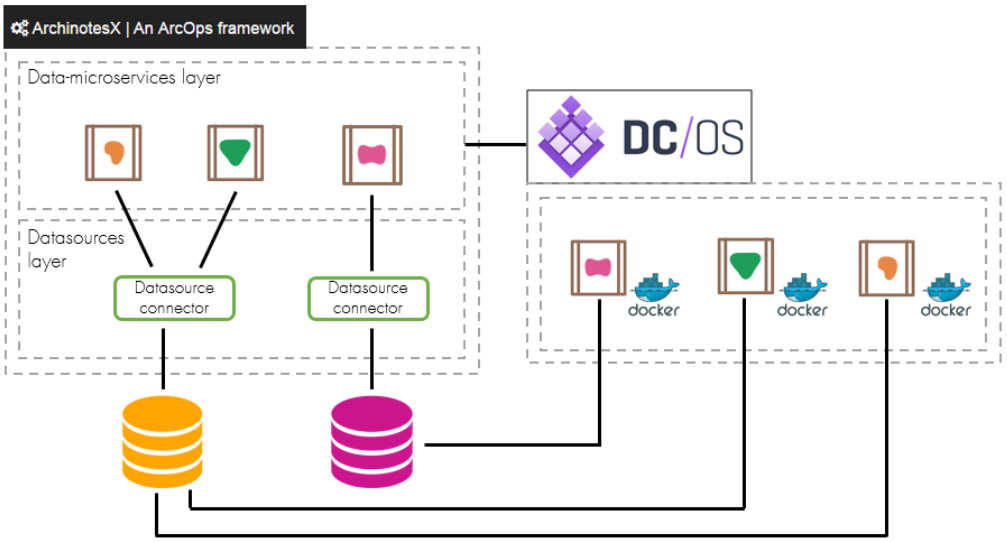

Figure 5.1 ArchinotesX architecture

5.2 Implementation

ArchinotesX was developed in a three part ecosystem: (1) A client that handles all the interaction with software architects, the client will host the links to the datasources and will receive the instruction to create the data-microservices, (2) the generated data-microservices that will be hosted separately in docker containers and (3) an Operating System for Data Centers that will be in charge of handling all the hardware in the company infrastructure and will manage the resources allocation for the data-microservices.

5.2.1 Client

The client is the only gateway that the software architects will have to interact with their architecture artifacts, here they can create links to their datasources and with that connection they can create data-microservices. Figure 5.2 shows the technologies, languages, frameworks, tools used to develop the ArchinotesX client.

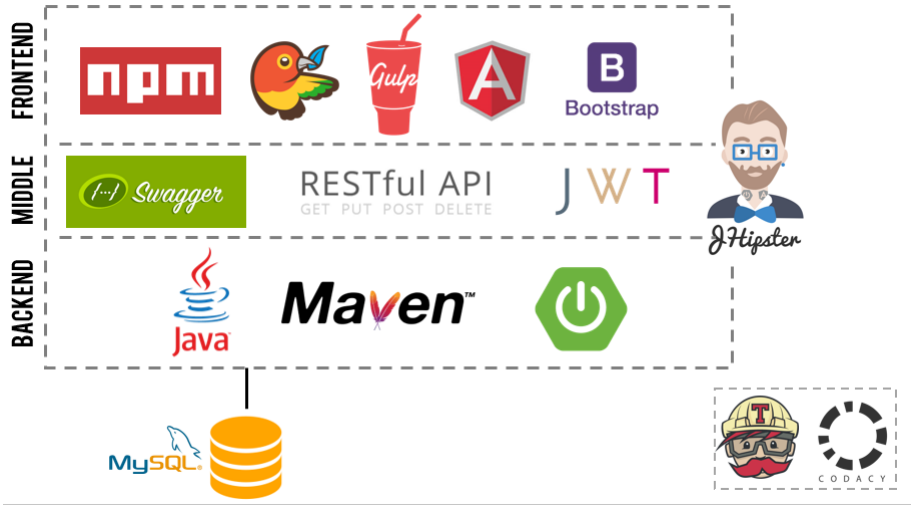

Figure 5.2 ArchinotesX: The client

The core structure of this project was generated using JHipster, an open-source application generator used to develop quickly a modern web application using AngularJS and the Spring Framework, it provides tools to generate a project with a Java 8 stack on the server side (using Spring Boot) and a responsive Web front-end on the client side (with AngularJS and Bootstrap). The client uses JWT as a security provider that defines a compact and self-contained way for securely transmitting information between parties as a JSON object, in this case it is just the RESTful API generated in the backend communicating with AngularJS requests in the frontend. The RESTful API can be easily and rapidly tested by means of Swagger, a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. The backend uses Apache Maven, a build and dependency manager for Java projects in order to have organized and centralized all the packages and dependencies that the project might need. The application is using a MySQL database hosted in Amazon Web Services and it is there where all the users, datasources and data-microservices information will be stored.

Regard the frontend, we have the most popular package-handlers such as npm, bower and Gulp.js, a javascript task runner that automates tasks such as bundling and minifying libraries and stylesheets.

In order to ensure the quality and consistency of the code developed in the client, we used Codacy, an open-source online tool that automates code reviews and monitors code quality over time assigning a grade for each commit in the repository, it is very useful to keep code duplication and several other errors at minimum. We also used Travis CI, a continuous integration service used to build and test software projects hosted at GitHub, this tool also executes on every commit performed on the GitHub repository and makes sure that nothing breaks from change to change keeping a consistent robust software.

5.2.2 Microservices

Notwithstanding the importance of software developers in any project and in order to make the project work very quickly, we jumped over the software developers labour by generating a standardized source code that will fit the requirements of any data-microservices required. Figure 5.3 shows the inside of any data-microservice that will be solicited through ArchinotesX.

Figure 5.3 ArchinotesX: Microservices

We chose Javascript as the main language used in the data-microservices, more specifically we chose Node.js, a cross-platform JavaScript runtime environment for this task, we just had to add a module through npm called Express, a minimal and flexible Node.js web application framework that provides a robust set of features for web applications that allows to create a HTTP utility interface. We wrapped this code around a Dockerfile, i.e., a container-like configuration that isolates every data-microservice in order to keep everyone functionality independent of the environment. The docker configuration receives two environment

variables, (1) the datasource connection string and (2) the table name that will require a data-microservice, this is already mapped inside the standardized code and the HTTP GET endpoint that will be generated obeys the values sent over those two environment variables.

5.2.3 Data Center Operating System, a.k.a., DCOS

DC/OS is a distributed operating system based on the Apache Mesos distributed systems kernel. It enables the management of multiple machines as if they were a single computer.

DC/OS can be used through a CLI, we use this method to communicate the ArchinotesX client and DC/OS, but as shown in Figure 5.4 it also provides a very understandable GUI that will show the real state of the allocated resources of the entire datacenter.

Figure 5.4 ArchinotesX: DC/OS Dashboard

DC/OS relies on three value proposition concepts.

- **High Resource Utilization** Deciding where to run processes to best utilize cluster resources is hard. Deciding where to place long-running services which have changing resource requirements over time is even harder. In re-

ality there is no single scheduler that can efficiently and effectively place all types of tasks. There is no way for a single scheduler to be infinitely configurable, universally portable, lightning fast, and easy to use - all at the same time. DC/OS manages this problem by separating resource management from task scheduling. Mesos manages CPU, memory, disk, and GPU resources. Task placement is delegated to higher level schedulers that are more aware of their tasks specific requirements and constraints. This model, known as two-level scheduling, enables multiple workloads to be colocated efficiently.

- **Container Orchestration** Docker provides a great development experience, but trying to run Docker containers in production presents significant challenges. To overcome these challenges, DC/OS includes Marathon as a core component that is capable of orchestrating both containerized and non-containerized workloads. Marathon has the ability to reach extreme scale, scheduling tens of thousands of tasks across thousands of nodes. There is a highly configurable declarative application definitions to enforce advanced placement constraints with node, cluster, and grouping affinities.

- **Extensible Resource Isolation** The simplest isolation method is to just delegate to Docker. It is trivial to run Docker containers on DC/OS, but Docker is a bit of a blunt instrument when it comes to isolation. The Mesos containerizer is much more flexible, with multiple independently configurable isolators, and pluggable custom isolators.

5.2.4 The ecosystem

Figure 5.5 explains how the aforementioned components communicate to each other.

- *Step 1.* Performs a connection between datasources testing if they will be reachable from the data-microservices.

- *Step 2.* Sends a request to DC/OS through the CLI asking to create a data-microservice with the configured environment information.

Figure 5.5 ArchinotesX: The ecosystem

- *Step 3.* DC/OS deploys a data-microservice somewhere in the cluster and the data-microservice becomes available through an HTTP endpoint.

Although these are the main steps, ArchinotesX and DC/OS communicate for some other issues, e.g., ArchinotesX requests the state of the deployed data-microservices every 5 seconds in order to notify the architect that something wrong is going on with the services.

The main idea of this implementation is to have a real environment that will allow us to verify our ArcOps proposal and test with experienced software architects the perception of the ArcOps concept, the explanation of the validation process can be found in next chapter.

6

Validation

6.1 Methodology

In order to correctly validate the contribution of ArchinotesX in a real environment we invited three software architects from a Colombian telecommunication company that currently holds more than 100 people only in IT; more specifically we invited architects from the Middleware team, whose main objective is to create web services that will be consumed by different areas of the company, satisfying repetitive needs in different software products, e.g., retrieve a client billing information that can be useful in an Android app as well as in a Web Portal, always through web services.

In Figure 6.1 we can observe more concise what the Middleware team does. Their web services are attached to an Enterprise Service Bus and they represent a single entry point from the business areas to the company datasources.

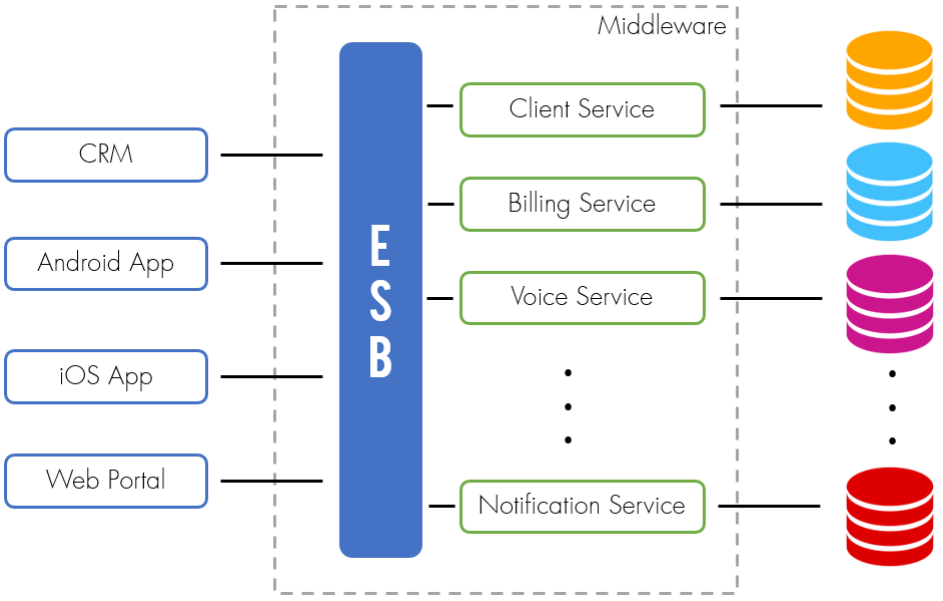

Figure 6.1 Middleware main functions

Regarding the architects, each one of them has more than five years of experience in both software development and software architecture, as well as a deep understanding of the process that a web service goes through, since both of them have worked in the whole lifecycle of a web service, from its conception to its launching.

6.2 Experimentation

The main goal of the validation process is to measure the potential of an ArcOps approach in a real environment. For this reason, we performed a Pre-ArchinotesX survey and a Post-ArchinotesX survey in order to know the perception of the architects towards ArchinotesX and corroborate if it presented a positive change in the architects perception towards the mentioned topics. For both surveys we used a typical five-level Likert scale [57] being 1 a Strong disagree option up to a 5, which is a Strongly agree option. The experiment was executed individually in order to have impartial opinions from the architects. In average the experiment lasted for sixty minutes.

6.2.1 Personal and Project introduction

(5 minute time-box) First, a brief professional introduction, then some gratitude words to the architect for making some time to participate in the experiment and finally an explanation of the purpose and steps of the activity as well as a brief explanation of our key concepts: DevOps, ArcOps, Executable architectures, Microservices.

6.2.2 Pre-ArchinotesX survey

(10 minute time-box) Based on your experience can you tell us if you agree or disagree with the next statements:

- **Q1.** The principles of ArcOps might be useful in my daily work and I think they can be implemented

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

- **Q2.** The use of executable architectures can help to keep architecture, coding, and deployment coordinated

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

- **Q3.** Microservices and SOA maintain the coherence between application architecture, application code and application deployment in run-time

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

- **Q4.** A SOA platform/framework will allow me to have an overview of the 'production environment' state e.g., resources utilization, deployment view, assignation view, etc.

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

- **Q5.** As an architect I think it is important to have a mechanism to really test the architecture I propose i.e., By means of an executable web-service I want to rapidly test if what I propose adjusts to the business requirements.

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

- **Q6.** As an architect, I think it is important to have a tool that will notify me any event that can happen to any of my services in production.

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

6.2.3 ArchinotesX introduction

(5 minute time-box) After the first survey was concluded, we introduced the ArcOps concept and if necessary, the DevOps concept in order to clarify what ArchinotesX has a background and therefore its main features; we already had a username/password for each architect so we started with a live example right away.

6.2.4 ArchinotesX experimentation

(30 minute time-box) We start by giving a quick tour inside the app and to explain at a high level what is the value proposition of DCOS, why we use it and how it is useful in this use case and if the architect asked for it, a technical tour of how ArchinotesX was build. After that the architect starts the process by creating a datasource and then using it to create a data microservice, we showed how that data microservice was deployed right away and ready to use. Then we simulate when a web service fails and show how it notifies the architect of the failure. Thence we show the different architectural viewpoints conceived in ArchinotesX.

6.2.5 Post-ArchinotesX survey

(10 minute time-box) Based on the features that ArchinotesX proposes:

- **Q1.** The principles of ArcOps might be useful in my daily work and I think they can be implemented

a) Strongly disagree

b) Disagree

c) Neither agree nor disagree

d) Agree

e) Strongly agree

- **Q2.** The use of executable architectures can help to keep architecture, coding, and deployment coordinated

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

- **Q3.** Microservices and SOA maintain the coherence between application architecture, application code and application deployment in run-time

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

- **Q4.** A platform/framework will allow me to have an overview of the 'production environment' state e.g., resources utilization, deployment view, assignation view, etc.

  a) Strongly disagree

  b) Disagree

c) Neither agree nor disagree

d) Agree

e) Strongly agree

- **Q5.** As an architect I think it is important to have a mechanism to really test the architecture I propose i.e., By means of an executable web-service I want to rapidly test if what I propose adjusts to the business requirements.

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

- **Q6.** As an architect, I think it is important to have a tool that will notify me any event that can happen to any of my services in production.

  a) Strongly disagree

  b) Disagree

  c) Neither agree nor disagree

  d) Agree

  e) Strongly agree

6.3 Results and analysis

Regarding the validation activity as a whole, the architects seemed very pleased of being invited to the experimentation, although they have been working under a defined methodology and a defined set of technologies, they were very open to the concepts that we explained, and in general, the experimentation took a little

longer since they wanted to understand more of the development of this project. Something curious is that none of them had heard of DevOps nor Microservices before and therefore, the answers of the pre and post surveys showed that the implementation of ArchinotesX was able to encompass the main principles of ArcOps / DevOps that we wanted to include.

The first question of the surveys was created in order to know the perception of the architects towards the concept of ArcOps itself, how they felt about the explanation of DevOps and ArcOps and the value proposition that ArcOps could bring to a team. As we can see in Figure 6.2 the average score increased by 2.7 points which indicates an increased understanding of the concept after experimenting with ArchinotesX.

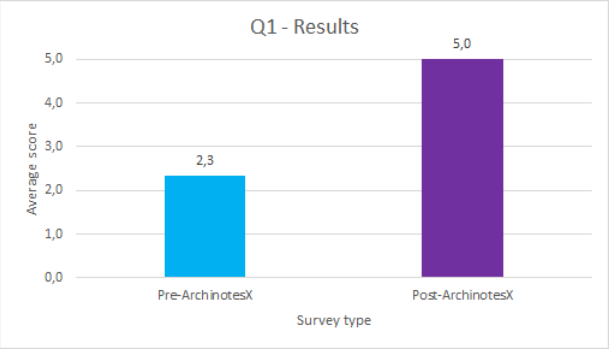

Figure 6.2 Perception Results of Question 1

The second question aims to evaluate if the architects think that their assets (architecture, development, deployment) can be fully coordinated by using executable architectures, in Figure 6.3 we can see that the average score increased by 2.3 points, this result, just as the result of the first question represents a huge difference of what the architects imagined when the concepts were explained and the value they perceived when they experimented how to implement the concepts.

In the third question the goal was to determine whether or not the architects think that by using Microservices and SOA, their architecture can be fully updated, in Figure 6.4 we can see that the average score increased by 0.3 points which indicates that it is almost indifferent to use their current platforms or ArchinotesX, they perceive that SOA and Microservices by themselves can't provide the value of having an application assets coordinated.

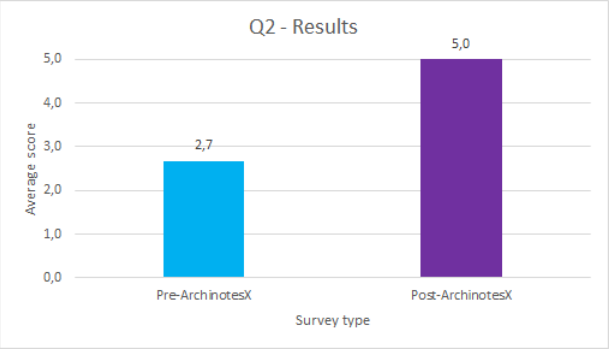

Figure 6.3 Perception Results of Question 2

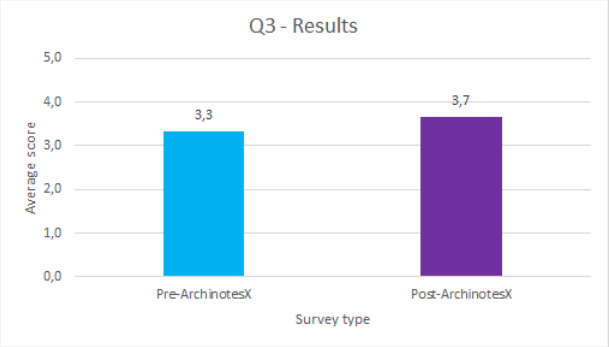

Figure 6.4 Perception Results of Question 3

In the fourth question in the Pre-ArchinotesX survey we wanted to know if the architects had an overview of the production environment as well as if they were capable of getting architectural views provided by their current SOA platform and in the Post-ArchinotesX survey we wanted to corroborate if they felt that ArchinotesX provided this functionality correctly. Figure 6.5 shows an increment of 1.7 points and the average score in the Post-ArchinotesX survey was 5, which leads us to an achieved goal of this requirement.

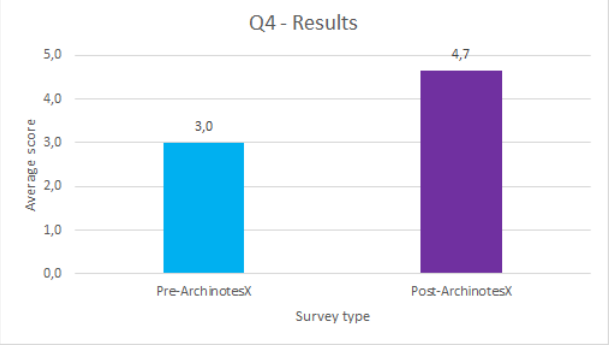

Figure 6.5 Perception Results of Question 4

Figure 6.6 shows us a variation of only 0.3 points in the fifth question which

talked about the importance of having a mechanism that would allow the architect to test the architecture to see if it is suitable in front of the business requirements, the variation is between 4.7 and 5 points which means that regardless of our proposition this is a key feature for the architects.

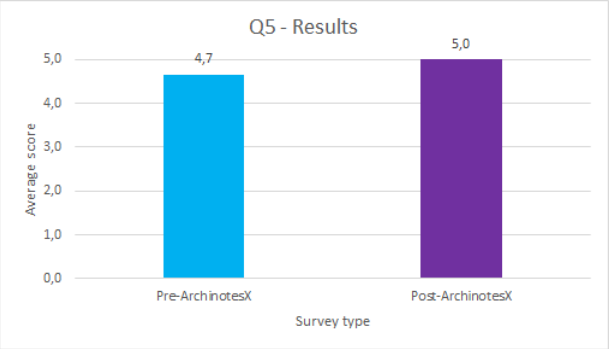

Figure 6.6 Perception Results of Question 5

Last question talks about a feature that we thought was going to be of great excitement to the architects, turns out, it didn't. We wanted to know how important it is for an architect to be the first to know when a service fails, before and after the experimentation this feature didn't seemed to be very important. Figure 6.6 shows a variation of 0.7 points, from 3.3 to 4.0. In general, they explained that they would prefer that notification to the operations staff since it is their assignment to solve those issues.

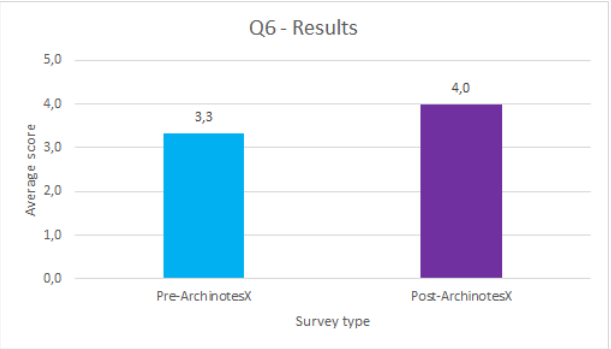

Figure 6.7 Perception Results of Question 6

6.3.1 Key findings

- The concept of ArcOps was very well received but it was truly understood just after the architects experimented ArchinotesX.

- The use of executable architectures represents a mechanism of forcing the operation to really be aware of the value of the architecture and in that way to keep it updated.

- The architects really loved the idea of having -through code generation- a mechanism to test their proposals or even to be able to respond to 'urgent' business requirements.

- In general, the architects think that implementing some of the cultural principles of ArcOps and executable architectures could bring real value to the company since the documentation of the architecture of the web-services has been updated more than 5 times and this has represented an investment of people and money, however, they also think that it is really hard that something like ArcOps or ArchinotesX could be implemented in the company due to bureaucracy, budgeting, etc.

- Some of the architects asked about the possibility of using ArchinotesX on side projects since they recognized the agility and the value that it could bring to an operation.

- DCOS really captured the attention of the architects. It was one of the key points we had to show off after the exercise was done.

- Finally, we realized that one of our premises was well established: The most repetitive requirement that the Middleware team receives is the task of going to a datasource, gather some data and present it.

7

Related Work

In order to validate the hypothesis described in previous chapters, we proposed a method -ArcOps- and a framework -ArchinotesX-. As a part of this investigation we looked up for existing solutions and we realized that none of them fully satisfies the criteria that we had in count when developing this project:

- **Architecture viewpoints.** We think it is important to have different general viewpoints of the designed architecture, viewpoints as deployment, functional and information provide a high-level picture of the components of an architecture.

- **Executable architecture.** It is a common situation to have a desynchronization between architecture artifacts and the real software artifacts deployed in production. The most straightforward way we found to mitigate this fact is that architecture artifacts and production artifacts are just

the same, that is why we add it to the evaluated criteria.

- **Failure notification.** We think that sometimes software architects do not have enough time to react to some circumstances, such as a fallen service. If an architect is notified in real time (RT) or near real time (NRT) of a web-service pitfall the impact on the business operation of the error can be reduced.

- **Collaborative architecture.** Usually, there are multiple architects and stakeholders interacting with an architecture, we think it is important that architects can work in a collaborative way since all of them have the same goal which is to build a reliable architecture for their company.

We found both widely known enterprise solutions like Enterprise Architect but also open-source projects like SaltStack, we selected three stable tools that are detailed next:

- **Enterprise Architect.** Enterprise Architect (EA) is a very complete tool with high end capabilities that help to design complex architectures. It is very robust and supports modeling business processes and modeling industry based domains. Although it is collaborative, the architecture is not executable and therefore the architects can't have a real picture of the state of the architecture's components.

- **SaltStack.** SaltStack develops systems management software used by large businesses and web-scale infrastructures for data center automation, hybrid cloud building and orchestration, server provisioning and application configuration management. Ultimately, SaltStack is used to manage all the data center things including any cloud, infrastructure, virtualization, application stack, software or code, however, it lacks on showing a big picture -or the architectural viewpoints- and also doesn't has the notifications module.

- **Distelli.** Distelli is a software workflow automation platform, it integrates the continuous integration and continuous delivery concepts and it is col-

laborative, however it lacks on showing a big picture -or the architectural viewpoints- since it is very 'ops'-oriented.

To summarize, table 7.1 shows a relation between criteria and the tools that we investigated.

Table 7.1 Related work comparison

	Other solutions			
Criteria	EA	SaltStack	Distelli	ArchinotesX
Architecture viewpoints	X			X
Executable architecture		X	X	X
Failure notification				X
Collaborative architecture	X	X	X	X

Conclusively, as our research dictates the problem stated in this thesis can't be solved with just one tool, a company can use an endless combination of them, however, these tools have no relation to each other so an extra 'layer' of administration of all of them has to be added to the whole process. ArchinotesX in its conception provides a collaborative framework in which architects can see the real conceptual and operational state of their components while also allowing them to have their traditional viewpoints.

8

Conclusions and Future Work

8.1 Conclusions

This thesis has defined a process called ArcOps that enables a coordinated relation among three core teams of any IT area: Architecture, development and operations. To achieve this previous goal, we have extended the DevOps concept enabling us to reuse the best of a research and a development that has been going on for years, also we built a fully functional framework that made possible the validation of the potential value that this process can bring to an organization.

In order to solve the goals described in Chapter 1, we proposed to include the software architects in the coordinated flow that already exists in DevOps between development and operations teams in order to keep every artifact aligned with each other -Architecture documents, source code, deployables-.

As of the validation of this investigation, we conclude that although ArchinotesX

is a beta-project, it was able to catch the attention of the architects and was a key factor in the understanding of the ArcOps concept. The architects seemed very interested in learning the concepts related to this research mostly because they expressed that the factors that ArcOps proposes and the features that ArchinotesX showed are actual problems for them right now and that the company has performed several re-documenting processes now.

As a summary, some punctual conclusions:

- We were able to extend the concept of DevOps towards the software architects reusing principles and core concepts.
- We designed and developed a framework capable of keeping coordinated software in production and software architecture artifacts by means of executable architectures.
- We were capable of collecting and analyzing the results of the perception surveys performed in an industrial environment, more specifically in a telecommunication company, helping us achieve our specific objectives.
- ArcOps is a concept that catches the attention of architects, the principles and the value proposition of ArcOps is recognized as disruptive in our case study.
- ArchinotesX as a validation project was able to express the principles of ArcOps that we aimed for, allowing us to realize that it could deliver value to IT teams.

8.2 Future Work

At the moment of the closure of this document, ArchinotesX supports the features necessary to validate key points in our proposition, however there is plenty of room for improvement and new features inclusion in ArchinotesX. Whilst the architects that helped us in the experimentation process were very open and pleased with the exercise on ArchinotesX we received some suggestions to keep in mind as future work:

- ArchinotesX should have into consideration the inclusion of securing the data- since this is one of the most important quality attributes while developing web-services.
- ArchinotesX could receive some datasources other than SQL sources, something like NoSQL databases.
- ArchinotesX could become a stable platform that would allow architects to rapidly test an architecture before diving into technical requirements, development investment, etc.

Appendix A: ArchinotesX Screenshots

We want to shot some screenshots taken while experimenting with ArchinotesX with one of the architects.

- Figure 8.1 shows the ArchinotesX landing page and login request.

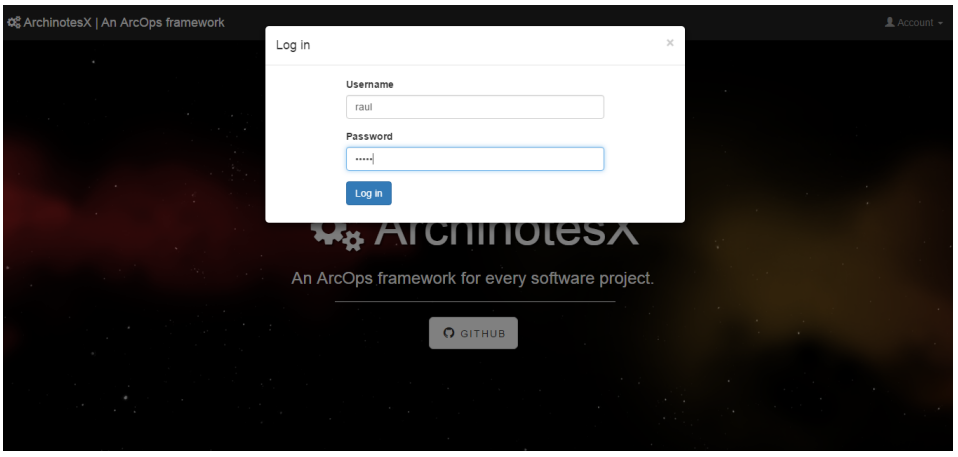

Figure 8.1 An ArchinotesX landing and login

- Figure 8.2 shows the main view of an architect when logged in ArchinotesX. The menu bar shows the Datasources, Viewpoints and Account configuration options. This view will contain data-microservices when existing.

- Figure 8.3 shows the datasources view, it is a common view that presents the basic information of the different linked datasources.

- Figure 8.4 shows the creation of a data-microservice, it is a simple view that requires an existing datasource and table selection as well as a non-repeated data-microservice name. The tags are also mandatory and they will be used in the architectural viewpoints.

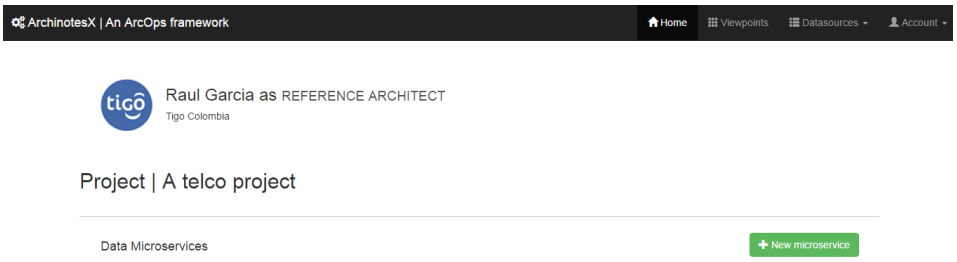

Figure 8.2 ArchinotesX main

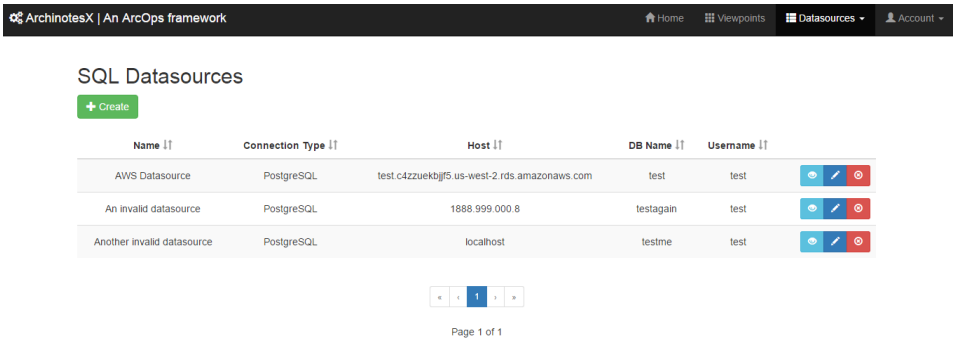

Figure 8.3 ArchinotesX datasources main view

- Figure 8.5 shows the action taken in DC/OS when a data-microservice is created, it automatically starts to deploy the microservice.

- Figure 8.6 was just about 10 seconds away from the last step, it is the time that takes DC/OS to have our data-microservice up and running.

- Figure 8.7 shows a petition to the URL that the data-microservice we just created is pointing to, it shows the values stored in the database.

- Figure 8.8 shows the main view of an architect when logged in ArchinotesX but this time the deployed data-microservice appears. It appears with a green color since the data-microservice is available.

- Figure 8.9 shows the functional viewpoint, we added some filters to the viewpoints since we wanted to avoid a terrible diagram if several microservices or datasources were created.

- Figure 8.10 shows the assignment view, we can filter here by datasource or

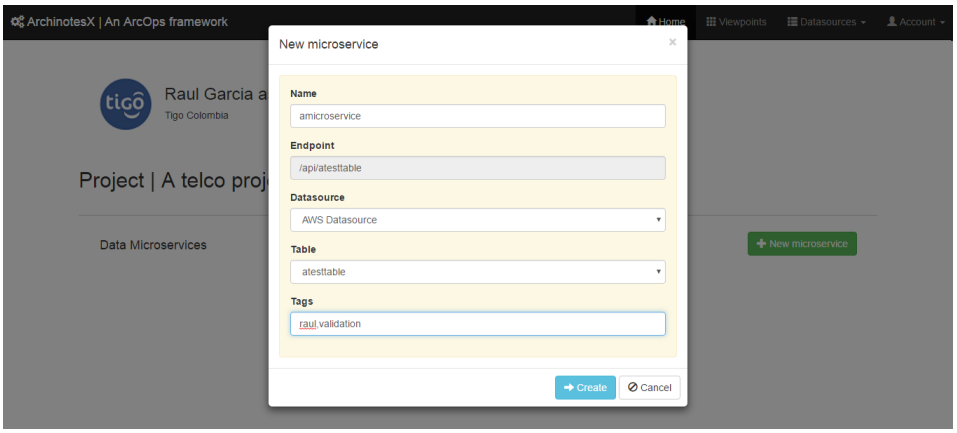

Figure 8.4 ArchinotesX: Create a data-microservice

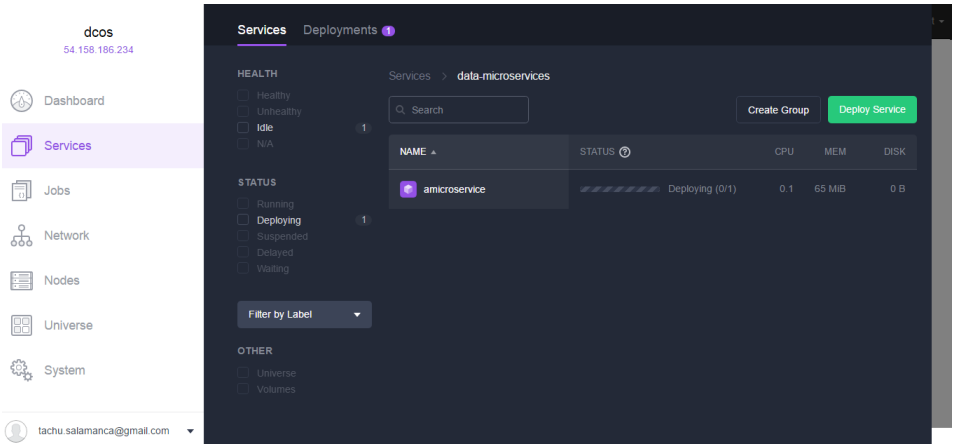

Figure 8.5 DC/OS service deployment

by tag and we will see the data-microservices that are tied to a datasource.

- Figure 8.11 shows the deployment viewpoint, we can see here were the data-microservices are deployed and what is the resource allocation of the datasource.

- Finally, to get Figure 8.12 we had to delete the deployed microservice in DC/OS. Whenever the ArchinotesX client realizes that the service is down it will change its color as a sign of alert.

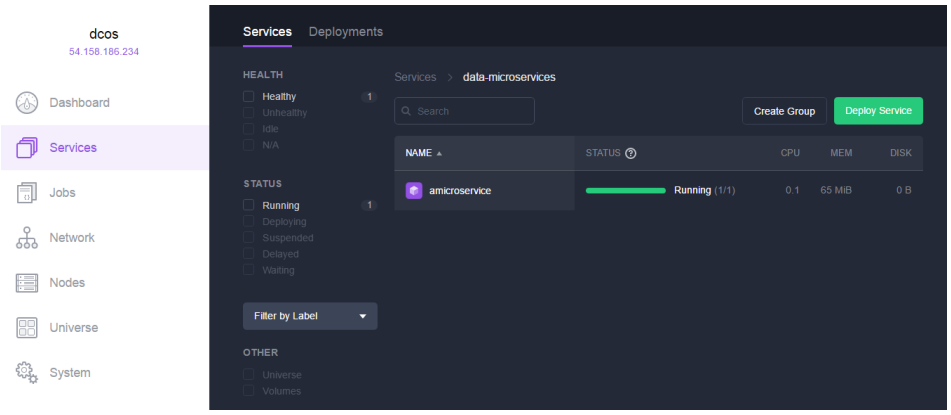

Figure 8.6 DC/OS service running

Figure 8.7 Data-microservice test

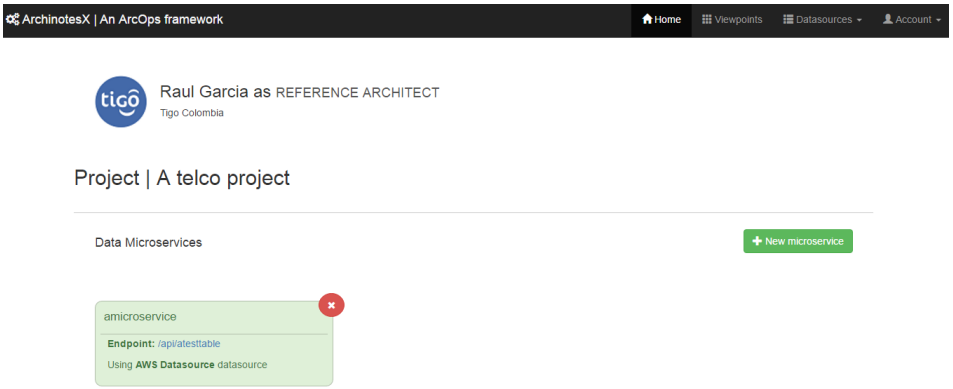

Figure 8.8 ArchinotesX, an available data-microservice

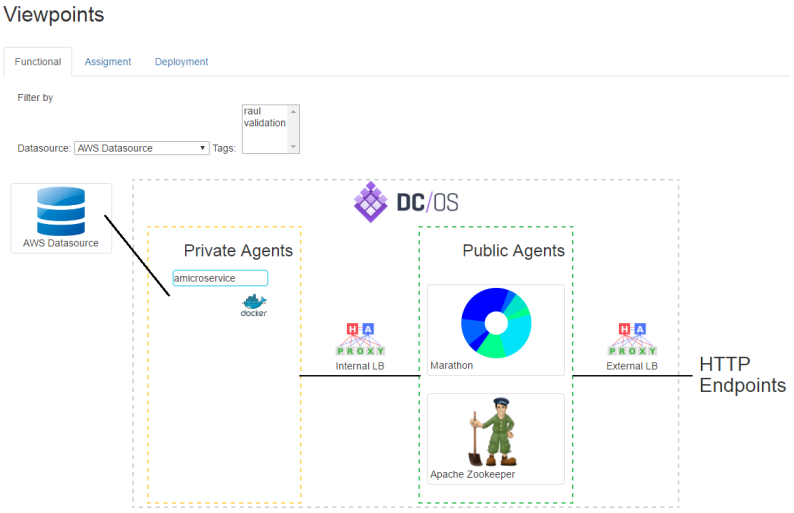

Figure 8.9 ArchinotesX, the functional viewpoint

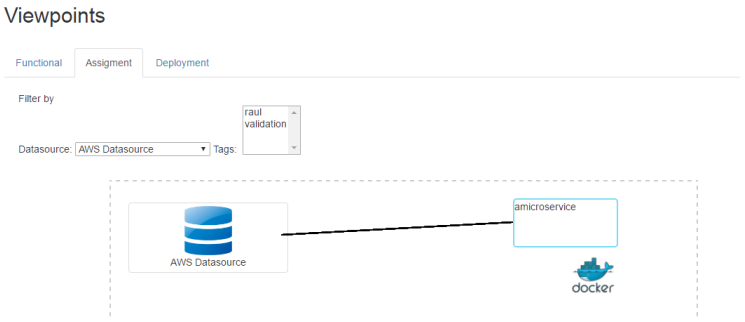

Figure 8.10 ArchinotesX, the assignment view

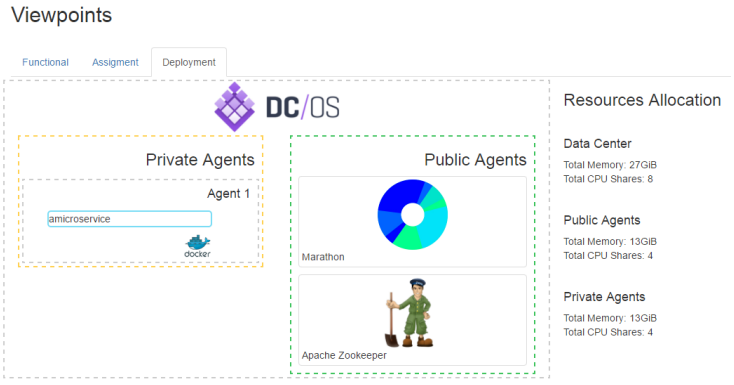

Figure 8.11 ArchinotesX, the assignment view

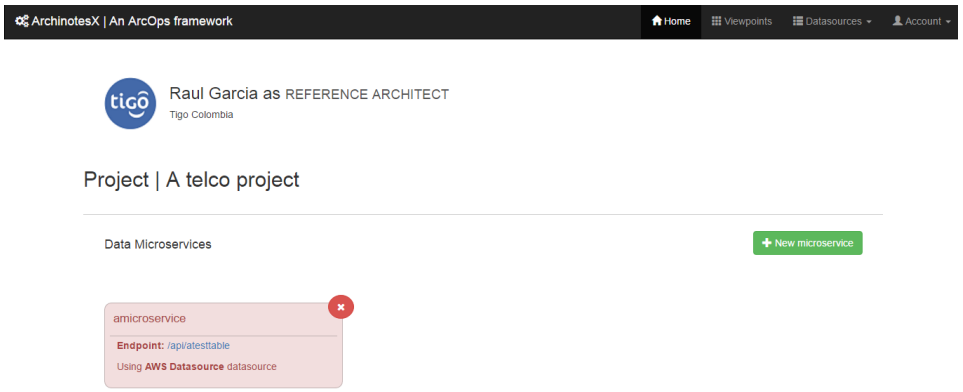

Figure 8.12 ArchinotesX, a failing microservice

Bibliography

[1] L. Bass, I. M. Weber, and L. Zhu, *DevOps: a software architect's perspective*. The SEI series in software engineering, New York: Addison-Wesley Professional, 2015.

[2] CollabNet, "The collabnet devops framework." `http://www.collabnet.cn/products/teamforge/deploy`. Online, Accessed: 2016-10-01.

[3] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. SEI series in software engineering, Boston: Addison-Wesley, 2nd ed ed., 2003.

[4] R. Murch, *The Software Development Lifecycle - A Complete Guide:*. Richard Murch, 2008.

[5] R. Kazman and L. Bass, *Toward Deriving Software Architecture from Quality Attributes*. ESC TR: Engineering and Services Center, Software Engineering Inst., Carnegie Mellon Univ., 1994.

[6] W. W. Royce, "Managing the development of large software systems: concepts and techniques," *Proc. IEEE WESTCON, Los Angeles*, pp. 1–9, August 1970. Reprinted in *Proceedings* of the Ninth International Conference on Software Engineering, March 1987, pp. 328–338.

[7] J. V. Sutherland, *Scrum: the art of doing twice the work in half the time*. New York: Crown Business, first edition ed., 2014. OCLC: ocn865157964.

[8] M. Lindvall and D. Muthig, "Bridging the Software Architecture Gap," *Computer*, vol. 41, pp. 98–101, June 2008.

[9] M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[10] C. L√≥pez, V. Codocedo, H. Astudillo, and L. M. Cysneiros, "Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach," *Science of Computer Programming*, vol. 77, pp. 66–80, Jan. 2012.

[11] X. Li and L. Huang, "An Approach to Reliable Software Architectures Evolution," pp. 305–312, IEEE, July 2013.

[12] M. Michlmayr, B. Fitzgerald, and K.-J. Stol, "Why and How Should Open Source Projects Adopt Time-Based Releases?," *IEEE Software*, vol. 32, pp. 55–63, Mar. 2015.

[13] G. Fairbanks, *Just enough software architecture: a risk-driven approach*. Boulder, Colo: Marshall & Brainerd, 2010.

[14] N. Rozanski and E. Woods, *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Upper Saddle River, NJ: Addison-Wesley, 2nd ed ed., 2012.

[15] P. Kruchten, "The 4+1 view of architecture," in *The 4+1 View Model of Software Architecture*, pp. 45–50, IEEE Software, 1995.

[16] M. Fowler, "Design - Who needs an architect?," *IEEE Software*, vol. 20, pp. 11–13, Sept. 2003.

[17] M. Fowler, "Is design dead?." `http://martinfowler.com/articles/designDead.html`. Online, Accessed: 2016-03-07.

[18] R. Martin, "The scatology of agile architecture." `http://blog.objectmentor.com/articles/2009/04/25/the-scatology-of-agile-architecture`. Online, Accessed: 2016-03-07.

[19] G. Miller, "Second generation agile software development." http://blogs.msdn.com/randymiller/archive/2006/03/23/559229.aspx. Online, Accessed: 2016-03-07.

[20] International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical and Electronics Engineers, and IEEE-SA Standards Board, *Systems and software engineering: architecture description = Ingeni√©rie des syst√®mes et des logiciels : description de l'architecture.* Geneva; New York: ISO : IEC ; Institute of Electrical and Electronics Engineers, 2011. OCLC: 782965386.

[21] L. O'Brien, L. Bass, and P. F. Merson, "Quality attributes and service-oriented architectures." http://repository.cmu.edu/cgi/viewcontent.cgi?article=1440&context=sei. Online, Accessed: 2016-03-07.

[22] D. K. Barry and D. Dick, *Web services, service-oriented architectures, and cloud computing: the savvy manager's guide.* The Savvy manager's guides, San Francisco, Calif. : Oxford: Morgan Kaufmann ; Elsevier Science [distributor], second edition ed., 2013. OCLC: ocn815363503.

[23] J. McGovern, *Enterprise service oriented architectures: concepts, challenges, recommendations.* Dordrecht, the Netherlands: Springer, 2005. OCLC: 262691631.

[24] E. Bertino, ed., *Security for Web services and service-oriented architectures.* Heidelberg [Germany] ; New York: Springer, 2010. OCLC: ocn268931371.

[25] J. Lewis and M. Fowler, "Microservices." http://martinfowler.com/articles/microservices.html, 2014. Online, Accessed: 2016-10-01.

[26] J. Thones, "Microservices," *Software, IEEE*, vol. 32, pp. 116–116, Jan 2015.

[27] J. Lewis and M. Fowler, "Organized around business capabilities." http://martinfowler.com/articles/microservices.html#OrganizedAroundBusinessCapabilities, 2014. Online, Accessed: 2016-10-10.

[28] M. E. Conway, "How do committees invent?," *Journal of Systems and Software*, Jan. 1968.

[29] M. Erder and P. Pureur, *Continuous Architecture Sustainable Architecture in an Agile and Cloud-Centric World*. MA,USA: Elsevier, 2016.

[30] C. Larman and B. Vodde, *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Upper Saddle River, NJ: Addison-Wesley, 2009. OCLC: ocn233547812.

[31] A. √Åvram, "The benefits of microservices." `http://www.infoq.com/news/2015/03/benefits-microservices`. Online, Accessed: 2016-03-13.

[32] F. Nemeth, R. Steinert, P. Kreuger, and P. Skoldstrom, "Roles of devops tools in an automated, dynamic service creation architecture," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 1153–1154, 2015.

[33] TechTarget, "How microservices bring agility to soa." `http://searchcloudapplications.techtarget.com/feature/How-microservices-bring-agility-to-SOA`, 2015.

[34] M. Virmani, "Understanding DevOps & bridging the gap from continuous integration to continuous delivery," in *Understanding DevOps & bridging the gap from continuous integration to continuous delivery*, pp. 78–82, IEEE, May 2015.

[35] J. Mooney and H. Sarjoughian, "A framework for executable uml models," in *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, (San Diego, CA, USA), pp. 160:1–160:8, Society for Computer Simulation International, 2009.

[36] T. Dobrzanski and L. Kuźniarz, "An approach to refactoring of executable UML models," in *Proceedings of the 2006 ACM symposium on Applied computing - SAC '06*, (Dijon, France), pp. 1273–1279, ACM, 2006.

[37] M. Amissah and H. A. H. Handley, "A process for DoDAF based systems architecting," in *A process for DoDAF based systems architecting*, pp. 1–7, IEEE, Apr. 2016.

[38] L. W. Wagenhals, S. W. Liles, and A. H. Levis, "Toward executable architectures to support evaluation," in *Toward executable architectures to support evaluation*, pp. 502–511, IEEE, 2009.

[39] D. J. Delgado, R. Torres-S√°ez, and R. Llamosa-Villalba, "Develop an Executable Architecture for a System of Systems: A Teaching Management Model," *Procedia Computer Science*, vol. 36, pp. 80–86, 2014.

[40] Ni Feng, Wang Ming-Zhe, Yang Cui-Rong, and Tao Zhi-Gang, "Executable architecture modeling and validation," in *Executable architecture modeling and validation*, pp. 10–14, IEEE, Feb. 2010.

[41] P. Helle and P. Levier, "From Integrated Architecture to Integrated Executable Architecture," in *From Integrated Architecture to Integrated Executable Architecture*, pp. 148–153, IEEE, 2010.

[42] T. Bagnall, "Executable architectures ,Äì do we have the time?." `http://download-na.telelogic.com/download/userpresentation/uk2007/Executable_Architectures_do_we_have_the_time_TBagnall_EADS.pdf`. Online, Accessed: 2016-03-13.

[43] D. Inc., "What is docker?." `https://www.docker.com/what-docker`. Online, Accessed: 2016-10-07.

[44] A. S. Foundation, "Apache mesos." `http://mesos.apache.org/`. Online, Accessed: 2016-10-07.

[45] D. Riehle, *Framework Design: A Role Modeling Approach*. 1999. Ph.D. Thesis, No. 13509, ETH Zurich.

[46] A. Aitken and V. Ilango, "A Comparative Analysis of Traditional Software Engineering and Agile Software Development," pp. 4751–4760, IEEE, Jan. 2013.

[47] C. Larman and V. Basili, "Iterative and incremental developments. a brief history," *Computer*, vol. 36, pp. 47–56, June 2003.

[48] M. Genuchten, *Towards a Software Factory*. Dordrecht: Springer Netherlands : Imprint : Springer, 1992.

[49] D. Leffingwell, *Scaling software agility: best practices for large enterprises*. The Agile software development series, Upper Saddle River, NJ: Addison-Wesley, 2007.

[50] I. S. Institute, "What makes waterfall software development model fail in many ways?." `http://www.scrum-institute.org/What_Makes_Waterfall_Fail_in_Many_Ways.php`. Online, Accessed: 2016-03-24.

[51] C. Riley, "Waterfall to agile to devops: The state of stagnant evolution." `http://devops.com/2014/11/11/waterfall-agile-devops-state-stagnant-evolution`. Online, Accessed: 2016-03-25.

[52] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What is DevOps?: A Systematic Mapping Study on Definitions and Practices," pp. 1–11, ACM Press, 2016.

[53] M. Keith and A. S. University, *Towards the Coordination of Service-oriented Software Development*. Arizona State University, 2009.

[54] X. Chen and T. U. of Texas at San Antonio. Information Systems & Technology Management, *Knowledge Coordination in Open Source Software Project Teams: A Transactive Memory System Perspective*. University of Texas at San Antonio, 2009.

[55] J. Urrego and D. Correal, "Archinotes: A tool for assisting software architecture courses," in *Software Engineering Education and Training (CSEE T), 2013 IEEE 26th Conference on*, pp. 80–88, 2013.

[56] S. Newman, *Building microservices*. 2015. OCLC: 902724653.

[57] R. Likert, “A technique for the measurement of attitudes,” 1932.

Index