FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Towards DevOps: Practices and Patterns from the Portuguese Startup Scene

Carlos Manuel da Costa Martins Teixeira

DISSERTATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Hugo Sereno Ferreira

Co-Supervisor: Tiago Boldt Sousa

June 27, 2016

Towards DevOps: Practices and Patterns from the Portuguese Startup Scene

Carlos Manuel da Costa Martins Teixeira

Mestrado Integrado em Engenharia Informática e Computação

June 27, 2016

Abstract

The DevOps movement aims to facilitate the work of both Developers and Operators by creating a new culture of collaboration between the two as well as introducing new practices and methodologies.
With initial discussions going back as far as 2008/2009, ideas regarding the DevOps movement and what it means are still evolving and, eight years later, are still topics of discussion even within the community. Combining that with the fact that some studies are showing a significant number of companies adopting DevOps and an equal amount looking into doing the same, it becomes worrying that DevOps literature is still largely based on personal opinions rather than scientific or academic literature.
This thesis represents a step toward a better and broader understanding, within the scientific community, of DevOps and its practices. With the objective of enabling more businesses to adopt DevOps and do so with less uncertainty we identify, in this thesis, 13 DevOps related patterns extracted from 25 Portuguese startup companies. Then, we conclude that the progress made with this thesis should be extended and we point out the direction to do so.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Abbreviations

ADT	Abstract Data Type
NIST	National Institute of Standards and Technology
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
IoT	Internet of Things
API	Application Interface
CAMS	Culture, Automation, Measurement, Sharing
CI	Continuous Integration
SCM	Source Control Management

Chapter 1

Introduction

The name DevOps comes from joining the words ’Development’ and ’Operations’. With some of the original work that triggered the rise of the DevOps movement being traced back as far as 2008 [9], we have been able to find multiple accounts and histories of personal opinions regarding the subject. This information has not, however, been the subject of scientific and academic analysis meaning that DevOps is still surrounded by uncertainty.

1.1 Context

Developing and maintaining/operating software are often seen as disjoint tasks and responsibilities. This pattern has been observed in several organizations like the ones described in *Agile infrastructure and operations: How infra-gile are you?* [9] and project management techniques like Waterfall [20]. Nevertheless, this was not always the case, and, in the *dawn of the computer age* , the same person that developed the software was also the person that operated it [17].
As a result of this separation, two separated departments often exist within organizations [11]. This two departments, Development and Operations, are usually not able to efficiently articulate which causes friction between the two [11] as well as a bottle neck for businesses [9].

1.2 Problem

The DevOps movement spans throughout a vast set of areas and tries to change both the technical and cultural aspects related with software development and with the software operations.
Looking at the current state of DevOps, we can see that interest in the movement continues to grow [25]. We can find, with a simple Google search, numerous blog posts about experiences and opinions regarding Devops and its adoption. Not only that, but there are also a growing number of *Devops ready* tools that promise to simplify and empower companies with the benefits of Devops. Contrastingly, a similar search on *Scopus* will yield close to 200 results which is a much smaller list when compared with other terms like *SCRUM* (more than 2000 results) or *Waterfall* (more than 700 results).

This lack of academic literature and study means that Devops understanding is still mostly based on opinions and personal experiences rather than scientific, peer reviewed literature. Consequently, adopting and practicing DevOps is still a surrounded with uncertainties.

1.3 Motivation

Devops represents a new way to look at the entire software pipeline. From development to delivery and maintenance, Devops represents an advantage for both teams and businesses by creating a more efficient, agile and collaborative way of working. DevOps also reduces the software time to market which provides a competitive advantage for those that are able to practice it.
Being such a strong driver of positive changes, we believed that studying Devops, its values and common practices will enable broader adoption, further strengthening the movement and benefits for both practitioners and businesses.

1.4 Goals

The main goal of this thesis is to increase the existent knowledge regarding DevOps in order to enable teams and companies that want to adopt DevOps with the required understanding of common pitfalls and solutions related with DevOps. At the same time further studies of the DevOps movement will be able to build upon this study by extending the current concepts or by having a base from which to search new ones.

1.5 Outline

This thesis documents a field study that tried to identify, near Portuguese startups, common practices and methodologies related with DevOps.
Chapter 2 introduces first the Cloud and then DevOps. This serves as, respectively, an introduction to some key concepts needed to understand this document and a base on which we based some of our study. The following sections, Patterns and a characterization of the Portuguese startup scene serve as justification for some of the choices described later in the document.
Chapter 3, describes how the study was made including the methodology used, what information was extracted, the filters applied to achieve the final sample and how the extracted data was handled and compiled.
Chapter 4 shows the results of the field study. In this chapter, thirteen patterns are described as well as the relations existing between them.
Chapter 5 shows the approach used to validate those patterns as well as the results of that validation.
Finally, chapter 6 identifies the main contributions of this thesis and suggests some future improvements.

Chapter 2

State of The Art

DevOps is usually associated with different types of technologies and practices. Effectively practicing DevOps means not only to understand the cultural aspects but also the technological aspects that enabled some of its practices.
In this chapter an introduction to cloud computing will be presented, providing the needed context for the following section where a description of the DevOps movement is presented. After this, both a small characterization of the Portuguese startup scene and of pattern languages will be presented in order to justify some of the choices presented in the next chapters.

2.1 Cloud Computing

Paraphrasing [4], consider the need for electricity. Rather than building an entire grid and a power plant, one only needs to connect to the public network. In the common scenario, charges are calculated based on the usage amount and no special knowledge of how the network setup is needed.
Clouds follow exactly the same rationale, rather than having to build/purchase the computing resources, one needs only to connect to a provider and the resources are available to be used. Charges, like in the electric grid, are calculated based on the usage and clients do not need to know how the resources are managed or setup.

2.1.1 Definition

In [18], NIST[1] defines Cloud computing as *a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

NIST also defines the following essential characteristics of cloud computing:

[1]National Institute of Standarts and Technologie

- **On-demand self-service** : A consumer can unilaterally provision computing capabilities, e.g. server time and network storage, automatically without requiring human interaction with each service provider [18].

- **Broad network access** : Users must be allowed to access resources through standard mechanism [18].

- **Resource pooling** : A multi-tenant model should be used in order to serve multiple users. Resources are allocated dynamically meaning that users do not know were, physically, the allocated resources are [12]

- **Rapid elasticity** : Resources can be elastically allocated or deallocate. This should be done automatically [18].

- **Measured service** : The usage of resources should be measured providing transparency in the provider-client relation [18].

2.1.2 Delivery methods

In regards to their accessibility, it is common to identify three main categories of clouds [29] :

- **Public Clouds** : Public clouds are a pool of resources hosted by cloud providers who rent them to the general public. This resources can be accessed over the Internet and are shared among users.

- **Private Clouds** : Private clouds are usually administered and used by the same organization. Alternatively a third party can also be hired to manage the resources. The main difference between public and private clouds is the usage of the resources. Private clouds resources are only used by one company as opposed to public clouds were resources are shared.

- **Hybrid Clouds** : Hybrid clouds combine both the private and public concepts. When using a hybrid clouds approach, infrastructure is divided by the two types of clouds meaning that some modules may be hosted in the private space and others on the public one.

- **Virtual Private Cloud** : Virtual private clouds are an alternative to private clouds. This type of cloud are essentially a public cloud that *leverages virtual private network (VPN) technology* [29] allowing users to combine characteristics of both public clouds and private clouds.

2.1.3 Service Levels

In terms of service levels cloud computing can be classified in regard to the provided abstraction. The main categories are the following [27] :

- **SaaS** - Software as a Service (SaaS) gives users access to a platform usually through a web client without the need to download or install software. The user is able to use the provided software instantly and virtually everywhere.
Applications of this model include messaging software, email services, collaborative platforms, etc.

- **PaaS** - Platform as a Service (PaaS) allows its users to quickly deploy applications with little to no configuration. In this type of platform environments are usually setup previously or configurable. PaaS users should nevertheless expect only to be able to deploy applications or software supported by the provider.

- **IaaS** - Infrastructure as a Service (IaaS) represents the lowest abstraction made available by cloud providers. In this model the user is able to configure and access a machine directly without constraints. This machine, usually a virtual server managed by the provider, can be configured and maintained by the user.
This model is used when applications are complex and therefore need complex configurations.

2.1.4 Benefits

The main advantages of cloud computing for its users can be summarized as following:

- **Monetary Efficiency** - Cloud providers allow users to keep their resources to the needed minimum. By allowing users to quickly and easily increase/decrease the allocated resources amount and billing clients only for the resources used, cloud providers are good way to save money and spend only the needed amount [12, 18].

- **Scalability** - usually through a public API of some kind most cloud providers allow for the quick increase or reduction of resources [18]. This enables businesses to quickly go from zero to millions of users with minimum overhead. Additionally, because processes related with the management and configuration of cloud servers can be automated, it is usually possible to manage large systems with small teams [17].

- **Maintainability** - Cloud Providers are responsible for the maintenance of all the hardware and infrastructure aspects. Cloud computing users therefore do not need to worry about updating the hardware or maintaining the physical infrastructure. This enables users to focus their resources in improving their product rather than improving the structure that supports it [12].

2.2 DevOps

In *Why DevOps Is Like An Onion* [7], Dave Sayers chooses to use the analogy of peeling an onion, with each layer representing a different concept, in order to describe DevOps. Further developing upon his initial idea, in this section, we will try to introduce DevOps using that same analogy.

2.2.1 The cultural layer

Looking at some of the first DevOps efforts we see that a lot of them aimed to create a better articulation between Developers and Operations [9, 2].
Being two distinct departments with different work methodologies and objectives, this meant that some cultural changes had to happen in order for the two departments to be able to create a common understanding of each other worlds [2].
The main cultural values associated with DevOps are, as a result, values that promote cooperation and communication. These are some of the ones that we identified:

- Respect [8, 2]
- Trust [13]
- Collaboration [8]
- Sharing [28]

2.2.2 The methodology layer

The DevOps movement does not define a specific methodology. Nevertheless, when we look for instance at CAMS 2.2.6 or at some of the practices like *Continuous Deployment* it becomes clear that, although a methodology is not defined, the one chose should enable an iterative and continuous improvement focused approach.

2.2.3 The practices layer

Broadly speaking, DevOps practices gravitate around the automation of processes. From the setup of environments on the developers machine, up to the deployment phase, the capacity to automate repetitive tasks is a key feature of DevOps. Common practices associated with the DevOps movement include:

- **Continuous Integration** - Regular integration of software helps discover risks associated with the integration of the software. [3]
- **Continuous Deployment** - Deploying often means that less functionality are deployed each time. This makes deployments more manageable and in turn safer [2].
- **Continuous Monitoring** - Continuously monitoring infrastructure and applications allows for the early detection of bugs [6]. It can also be a way to identify areas that can be improved [28]
- **Defining infrastructure as code** - By defining infrastructure as code it is possible to increase reliability and consistency [17].

- **Using feature Flags** - Feature flags are special flags that toggle features on and off. This can improve error handling by allowing faulty features to be turned off. Feature flags also enable more complex schemes of operation where certain functionality are launched, but are not displayed or are displayed just to certain users. Once it is observed that the new functionality works, the flag can be turned on and the new functionality is available [4].

- **Others** - As we will see in 2.2.6 there can be a great number of practices that can be considered DevOps practices.

2.2.4 The tool layer

Tools allow for some practices to be more effective. In this section we will present some of the existent categories of tools. This categories are presented in [16] and are currently being maintained and increased by the DevOps community:

- Source Control Management (SCM)
- Continuous Integration (CI)
- Deployment management
- Cloud platform and infrastructure
- Monitoring
- Repository Management
- Infrastructure Provisioning
- Release management
- Logging
- Security
- Build
- Testing
- Containerization
- Collaboration
- Database Management

2.2.5 The full onion

As it was stated by Paul Hammon and John Allspawn at their *10+ Deploys Per Day: Dev and Ops Cooperation at Flickr* [2] presentation, tools and processes alone are not enough and the cultural change should be the first step to take towards DevOps.
Because of this, the outermost layer in the onion (fig. 2.1) represents the most important layer and without *peeling* it will not be possible to take advantage of the inner layers [2].

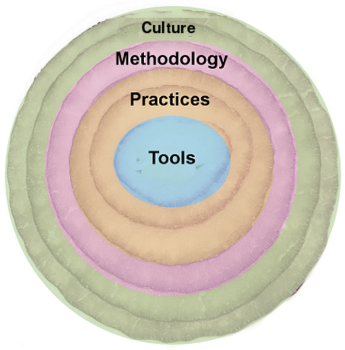


Figure 2.1: The DevOps onion [7]

2.2.6 A DevOps definition

DevOps vastness can be seen in the enormous amounts of tools categories identified in 3.1.1 related with DevOps.
As one might expect from this vastness of areas that DevOps touches, there are still difficulties to properly define DevOps. In order to reduce this lack of definition, we will use throughout this thesis a conjugation of two definitions for DevOps.
The first definition is from *DevOps: A Software Architect's Perspective* [4] and it states that:

DevOps is 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.

We chose this definition because it allows to easily classify something as being DevOps or not, i.e. one would only have to ask himself if a practice or cultural aspect will allow for the reduction of time since committing a change until that change is in production, if it does, then it is DevOps. Nevertheless, we find this definition to be a bit empty in the sense that by only reading it one would not be aware of the aspects that are associated with Devops. As a workaround for this problem we use a second definition based on the CAMS acronym.
The CAMS acronym [28] defines DevOps as being a **C**ultural movement where **A**utomation and continuous **M**easurement of processes and people are promoted and where the later can serve as an input for the **S**haring of problems and new ideas. As new ideas appear to solve the identified problems, new measurements can be made to further identify problems further improving the overall process.

2.2.7 DevOps Benefits

In a recent study [10] the following benefits were identified:

- DevOps projects are believed to accelerate in 15%-20% the ability to delivery of capabilities to the client

- Adopting DevOps allows business to practice Continuous Delivery.
- The average cost of a critical application failure per hour is $500,000 to $1 million (DevOps can help reduce application failures).
- The average cost percentage (per year) of a single application's development, testing, deployment, and operations life cycle considered wasteful and unnecessary is 25% (DevOps can help automate some repetitive tasks and by doing so reducing some of the waste associated with those tasks)

2.2.8 Patterns

Some previous progress has already been made regarding the identification of DevOps related patterns. We will summarize this progress by listing the identified patterns and briefly describing them:

- **Store Big Files in Cloud Storages**[6] - Instead of creating and managing a system to store large files, or storing them in database columns, store them in a Cloud Storage[2]
- **Queue based solution to process asynchronous jobs**[6] - When there are tasks that take a long time to complete but users still expect a quick response, create a new Job instance in a queueing service and then have a service performing those tasks. When finished, post the result of that Job somewhere acessible to the user and notify the user that the task is done.
- **Prefer PaaS over IaaS**[6] - For non technology companies, PaaS is usually preferable because it will give them lot of functionality without the need for configuration. This will allow them to simply focus on their core business.
- **Load Balancing Application Server with memcached user sessions**[6] - Use a load balancer in front of your application servers. This severs will handle sessions using memcached which means that if a application server goes down or if a new application server is needed, it will be able to use the user session.
- **Email delivery**[6] - Rather than implementing your own SMTP solution, use cloud mail delivery services which provicde REST API's to send emails.
- **Logging**[6] - Having multiple servers you need a way to consolidate your application logs. In order to do so, you should use a cloud based log service.
- **Realtime User Monitoring (RUM)**[6] - Monitor user behaviour in order to find possible bugs or errors.
- **The isolation by containerization pattern** [22] - *Use a container to package the applications and its dependencies and deploy the service within it.*

[2]A storage system provided by a cloud provider.

- **The discovery by local reverse proxy pattern** [22] - *Configure a service port for each service, which is routed locally from each server to the proper destination using a reverse proxy.*

- **The orchestration by resource offering pattern** [22] - *Orchestrate services in a cluster based on each host's resource-offering announcements.*

2.3 The Portuguese startup scene

Motivated by a recent investment in innovation and entrepreneurship, Portugal startups have been growing their position in the global startup scene [5].

A study from 2015 [23] in which the Portuguese startup scene was analyzed, revealed that there were already 40 technology scaleups [3] operating in Portugal at the time. The same study stated that this startups were able to raise a large portion of the received investment from international investors indicating, therefore, that the reach and scale of this startups was broader than just the national arena. Additionally, it is also indicated in the study that Porto and Lisbon are the main centers of innovation, encompassing 70% of the total of existing scaleups. In addition to the scaleups identified other smaller scale startups exist. Some of this startups are currently being incubated in incubators around the country like UPTEC [4] and Startup Lisboa. This incubators had, at the time of this study more than 300 companies [26, 24] under their wing.

2.4 A pattern language

Cristopher Alexander wrote *A Pattern Language*[1] in 1977 and in doing so, introduced the idea of representing knowledge on the form of a pattern language. Since then, pattern languages have been used in several areas [19] including software development where they were used to describe common architectural solutions [15] or recurring software design choices [14].
A pattern is a recurring solution to a problem that lives inside a specific context [19]. Pattern languages are *collections of patterns that are related to each other by virtue of solving the same problems or parts of a solution to a larger, partitioned problem.* [19].
There are some key characteristics that should be taken in consideration when writing patterns. In *A pattern language for pattern writing* [19], some of this characteristics are presented in a form of patterns themselves, they can be summed up as follows:

- patterns should identify a pattern **user** operating in a **context**. This pattern user should have and identified **problem** that he solves using a **solution** that takes into consideration some of the constraints defined by the context.

[3]Scaleups are companies that raised more than $1M funding (since foundation) and had at least one funding event in the last five-year period

[4]Science and Technology Park of University of Porto

- patterns should be easy to read and understand and should not force the reader to read the entire pattern multiple times in order understand it.

Because patterns represent a way to share recurring solutions to a problem [19], pattern languages help reduce re-discovery and re-invention of concepts and functionality as well as *avoiding common pitfalls that are learned from experience* [21].

Chapter 3

Towards DevOps

In this chapter we will look at the approach used in order to solve the problem described in 1.2. We will start by looking at how we defined the sample from which we extracted information and then how we handle that data in order to produce the final results.

3.1 Methodology

In 2.2 we saw that the DevOps movement emerged in the software development[1] community. Having this factor into consideration and knowing that there is still a significant lack of scientific information regarding the subject, we choose to look for a solution within the development community. We believed, before conducting this study, that not only would they be able to provide us that information, but because they were using this techniques/tools in their daily activities, it would serve also an extra layer of assurance.

3.1.1 Adjusting Granularity

The vast number of existing tool categories as well as the fact that there are a lot of possible practices that can be considered part of DevOps 2.2.6 means that fully capturing the ideals and ideas related with it would not only be a tremendously hard task but would also not fit in the length of this thesis.
Taking this into consideration and in order to avoid over specializing this thesis in some particular area, we set the desired granularity for the study by defining that we would only focus on categories of practices and if needed tools e.g. we wanted to know what steps did the Continuous Integration server run, but we were not interested in fully documenting the tool or measuring the time it took to run a full integration cycle.

[1]software development should be seen in this context as both the development, maintenance and any other tasks related with the creation of software

3.1.2 Collecting the information

Taking into consideration the vastness of topics that DevOps 2.2.6 touches, we knew that it would be hard to create a form that covered that extensively.
It was theorized, at this point, that not all startups would have similar levels of maturity(we later observed this to be true A.4). This meant that even if we could create this form, it would be too extensive for some cases and we would still have to create an all englobing form to capture all the information.
The chosen methodoly adopted was to do a more exploratory type of interview. This idea lead to the creation of a script rather than a form that would guide the interviews. This script A.3 had 5 major sections:

- **Product** - The product section would try to understand, first what did the company do and secondly if there were any kind of special requirements that would influence the choices made by the company.
- **Team Management** - Team sizes, interactions, project management techniques would be analyzed here.
- **Software delivery pipeline** - In this section, we identified if teams did Continuous Integration, how did they handled the creation of environments for each of the pipeline states and what teams did what in each state.
- **Infrastructure Management** - We tried to capture how the companies handled their infrastructer. Did they use the cloud? Which processes did they automate?
- **Monitoring & Error Handling** - With this section we aimed to understand if the companies were monitoring their infrastructure, how did they do it and, when errors were detected, how were they responding?

3.1.3 Defining a sample

As it was seen in 2.3, Portugal has a rich startup community.
Startups have strict contraints regarding the ammount of resources they have at their disposal and have, as a result, aditional incentive to automate as many tasks as they can.
Being small (in terms of staff), communication and cultural aspects are usually guided towards cooperation as this is a key factor in allowing small teams to handle large projects.
Finally, the fact that startups have as their objective to scale, further highlighs the need for automation and cooperation.
This conjugation of factors meant that the mindset of startups was aligned with the DevOps one and startups are therefore a good place to look for information that can be directly linked to DevOps.
Having more than 300 startup companies 2.3 from which to choose, we needed a way to filter out companies that may not be relevant to our study.

With that objective we first created a list of all the startups companies we knew that operated in Portugal. This list was create by looking at some of the Portuguese startup incubators and by extracting the list of companies.
From those 300 startups, we attempted to identify which ones were doing software development. To do so we looked, when available, at the company web page and tried to determine if the company product was software related or not. This approach reduced the number of companies to 155.

Then, we prioritize which companies were better or worst for our study. To do so we created a compound evaluation metric that would allow us to rank companies. We created 5 metrics to do so:

- **Cloud Usage** - We created three possible values for this metric. If a company used cloud services, we would give the company 2 points. If we were not sure if a company was using cloud services, we would give it 1 point. If we know the company was not using cloud services, we would give it 0 points. With this metric we attempted to distinguish between, for instance, companies that were developing hardware solutions from those that were developing more software oriented solutions.
- **SaaS/PaaS offering** - Having the same point attribution schema as the *Cloud Usage* metric we believed that if a company had a SaaS/PaaS product, it would need to have some sort of automation put into practice as it would need to be able to scale if there was a sudden increase in clients.
- **Company Size** - We create four possible values for each metric. Companies could have 0,1,2,3 points if they had respectively less than 5 members, between 5 and 15 memebrs, more than fifteen members or more than fifteen members and several teams.
- **Subjective Appreciation** - This metric would reflect the overall opinion of the company that we developed when searching for information for the other metrics. Some common factors that influenced this metric were for instance the fact that some companies had listed staff members working on software development task, or if the company website was down.

The resulting ranking, created by summing all the factors, is not supposed to be seen as a precise way to accurately compare companies i.e. the second company maybe as relevant as the first one, but rather as a way to prioritize them, i.e. the first company surely is more intersting to study then the last one.

In the end, we contacted a total of 60 companies from which we were able to interview 25.

3.1.4 Sample Characterization

The sample, as seen in fig. 3.1 was mostly composed of small companies with no more than 35 employees and with half of the sample size to have less than 15 members. The companies also conducted a substancial amount of their business over the internet either through a browser application or a mobile app or a combination of both.

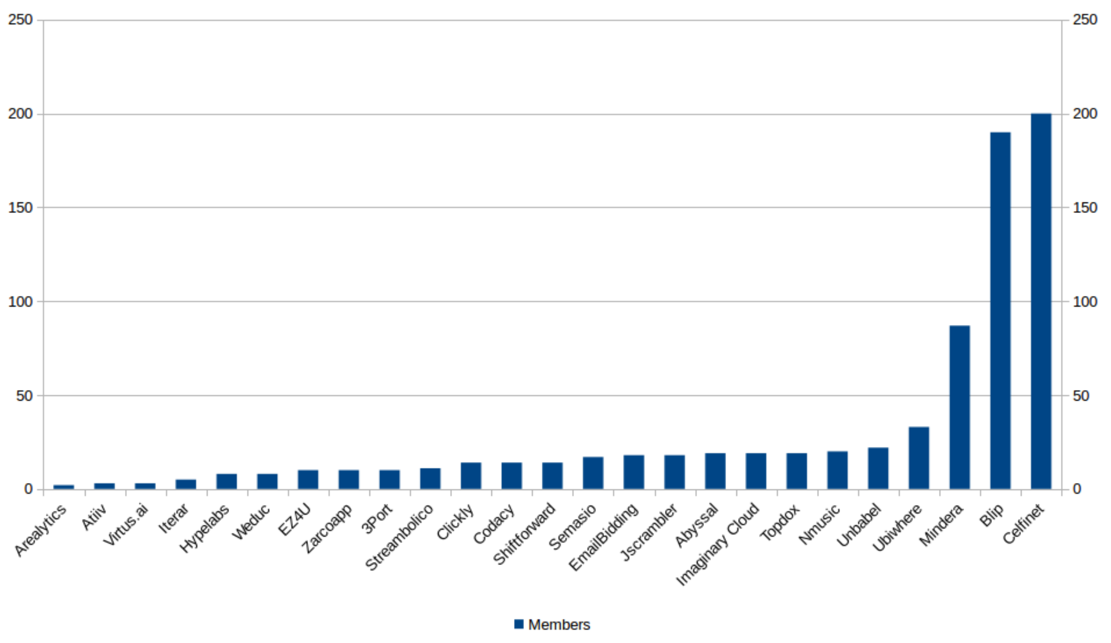


Figure 3.1: Studied companies size

3.1.5 Data Processing & Pattern Extraction

After conducting the interviews, the next task was to analyze and process the collected information.

Fig. 3.2 shows how we merged some of the identified techniques into larger groups that would represent patterns. The full grouping can be found in A.2.

As this was being done, it became clear that, because of the size of the sample (25 companies), this study would not be statistically significant. Nevetheless we found that the information we gathered had value and would be able to provide future studies with usefull pointers on how to approach the problem.

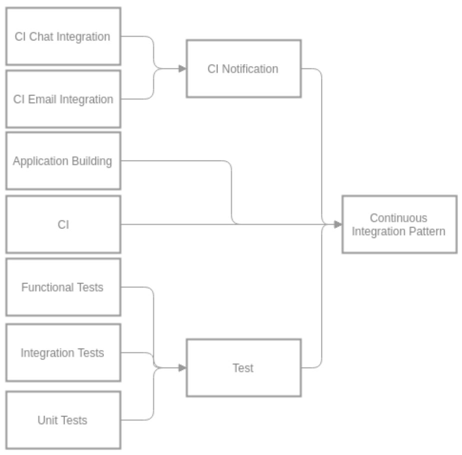


Figure 3.2: Identified concepts and subsequent refining

Due to the fact that some practices were rare, we had to group them into categories in order to have a representative value inside the sample, e.g. some companies would run integration tests on their CI tool while others would only run unitary tests; both would, nevertheless, run tests.
Using this groupings, we were able to identify a set of problems that companies were solving using different approaches. We were able, as a result, to extract in total 13 patterns each representing a problem and providing a set of solutions that took into consideration the context of the company. The final list of practices and frequencies can be found in A.1. This set of patterns can be found in chapter 4

3.1.6 Patterns

Throughout the interviews we noted several tendencies.
The first was the widespread use of Cloud computing with 76% of the interviewed companies using it. Then there was the dominance of Git as the version control system with 80% of the interviewees using it as their version control system. Chat and direct communication were also the main ways chose by companies to communicate (100%).
Being able to measure some of the key business metrics like the health of the system is also a concern for most companies with almost 70% of them having some sort of system to do so.
Having a way to replicate environments was also a major trend among companies with 60% of companies already adopting one way to do so.
The following table(tab. 3.1) enumerates the categories of problems identified with the percentage of interviewed companies that were handling that problem. These problems will be further developed in chapter 4

Categories	**% of companies that solved the problem**
Version Control	100%
Communication	100%
Cloud	84%
Auditability	68%
Continuous Integration	64%
Error Handling	60%
Reproducible Environments	60%
Scalling	48%
Code Review	44%
New Computing Instances Deployment	44%
Error Alerting	36%
Job Scheduling	12%

Table 3.1: Categories frequency

Chapter 4

Patterns from the Portuguese Startup Scene

Having discussed how patterns were identified in chapter 3 and why they are a good way to convey information in 2.4, we will present, in this chapter, the patterns identified during the interviews. A small report of each interview can be found in A.4.
To allow for better read the chapter map provided in A.1 can be used.

4.1 Team orchestration

Context

You are developing a solution or product and the requirements for that product/solution may change rapidly. This maybe due to the fact that you are experimenting new features and seeing how your users react to them or because your client needs change often.

Problem

How do you orchestrate your team(s) so that they can handle new challenges and deliver results in a sustained manner?

Forces

- *Efficiency.* Specialized teams are faster at answering specific problems related with their speciality.
- *Need for articulation.* Specialized teams will not be able to deliver a final product if it requires skills beyond their speciality.

- *Conflicting objectives.* Objectives for specialized teams may conflict with other specialized teams (e.g. the team responsible for the performance of an application may not agree with the user experience team when the later wants more content in a specific page).
- *Agility.* Multidisciplinary teams can deliver a final product if they have all specializations represented.
- *Cooperation.* To be effective, members of multidisciplinary teams must be able to work together.
- *Parallelization.* It may not be possible to have all expertise constantly working in parallel in the same team.

SOLUTION

Most companies we interviewed opted for a multidisciplinary approach to their team(s) orchestration and some, like Semasio and Emailbidding, were even changing from a specialized approach to a multidisciplinary one effectively valuing *Agility* more than *Efficiency*.
In this multidisciplinary environment teams *cooperation* and *Communication* 4.2 is the key to deliver a product that takes into account several constraints and requirements specified by each of its members specialization. By working together, this teams are also able to reduce the possible frictions due to *Conflicting objectives*.
In terms of **size**, we did not observe teams bigger than 10 elements. This was usually due to the fact that multidisciplinary teams have to be able to communicate effectively and having a lot of members would hinder that ability.
Occasionally, due to changes on requirements, some team members can be shared with other teams allowing for somze *parallelization*.

4.2 Communication

CONTEXT

Members of a multidisciplinary teams have **different knowledge** and **backgrounds**.
Professionals working in this kind of environment will sometimes disagree as they guide their work by different constraints and goals. **Facilitating** the resolution of this discussions and promoting the **sharing** of **knowledge/view** points is, therefore, key and solutions must be found to promote this.

PROBLEM

What kind of channels should you create to promote communication and facilitate it?

Forces

- *Facilitate.* You want to allow people to communicate easily.
- *Interrupted.* When concentrated on a task some people do not like to be interrupted.
- *Links and Files* Some information (e.g. links ,files and code) may not be easy to share verbally.
- *History* Sometimes you want the content of the communication to be persisted.

Solution

All companies that we interviewed had a set of communication channels that their teams could use. This way, teams could choose when to use each channel depending on their needs.

- **Direct Communication** is efective in handling day to day problems (e.g. solving doubts, giving advice, asking for help) because it *Facilitate*s communication. Having the teams physically working together is a great way of promoting this type of communication.
- **Chat tools** allow you to share *links and files* and code quickly and can keep and *history*of previous conversations. This tools can easily get cluttered with information making it hard to find old messages.
  Sometimes chat tools are also useful if you need to speak with someone and you do not want to *interrupt* him.
- **Emails** can be used for sharing information that is not urgent(e.g. scheduling a reunion for next week). Additionally, emails can be used if and *history* of your communication needs to be stored like when speaking with an outside provider or with a client.

Related Patterns

This pattern is related to both the *Continuous Integration*4.8 and the *Alerting* 4.11 in the sense that the same channels defined here can be used to send the messages those systems generate.

4.3 Version Control Organization

Context

As more people are working on the same team and contributing to the same product it becomes increasingly difficult to manage and synchronize those contributions. Tools like Git, SVN and Mercury are helpful on dealing with this kind of problem.
You have chosen to use Git either because you believed it was the best fit to your project and you have the need to know:

- What is the code in each of my environments (e.g. production, development) ?
- What was the code developed for a specific feature ?

PROBLEM

How do you setup your version control branching strategy so that you can infer valuable information about your current state and past events ?

FORCES

- *Branch Count* Having too many branches may be complicated to manage or cause confusion and having too litle may make you loose information.

SOLUTION

In the interviews we made we identified several branching techniques that were unique to each company like the one from EmailBidding. However, most of them would have similarities to some already known techninques hinting that they were adaptations of this techniques. These are the core version control oraganization patterns we have found:

- **GitFlow** specifies that at any given time two branches should be active. This branches are the **master** and **develop** ones. The code present in the master only contains shippable code. The *develop* branch contains the most recent working version of your code. This branch should not contain non working code but it may contain, for instance, features that have not been through a QA process. Adding to this two branches there are additional branches called *feature branches*. This branches represent a new feature under development and there should be one *feature branch* per feature. When a feature is implemented it should be merged into the *develop* branch. If that feature and the previous ones are considered production ready then the *develop* branch should be merged into the *master* one.

  Finally, if at some time you need to create a hotfix, you can do it by creating a new *hotfix* branch with the content of the *master* branch, applying the changes and merging it back into the *master* branch.

- **Feature Branches** can be seen as a subset of the **Gitflow** strategy. Instead of having a *develop* branch, this strategy only uses the *master* and *feature* branches The *master* branch holds tested and functioning code, the *feature* branches (one for each feature) hold the code of the correspondant feature. When a feature is ready and tested it is merged from the *feature* branch into the *master* branch.

RELATED PATTERNS

It is possible to use the *Code Review* 4.13 pattern even without using any sort of version control. Using one, will nevertheless help organize the process.
In the same way, *Error Handling* can also be improved by using the correct *Version Control Organization* as it will allow you to for instance to traceback what was the last live version or what was the code introduced in a hotfix.

4.4 Cloud

CONTEXT

Your company and/or your product needs to acquire computing resources in order to perform tasks like:

- performing **large complex operations**.
- supporting a **website** or a **web platform**.
- any other kind of **computing task**

This resources should be accessible and configurable and you believe that you do not need to physically connect to them it in order to control them.

PROBLEM

Owning computing resources is essential or at least advantageous to you or your business so the question is how do you acquire and maintain computing resources in a efficient way ?

FORCES

- *Upfront Costs.* Acquiring hardware may require significant upfront costs.
- *Maintenance.* Depending on the ammount of hardware you have to manage, a person, team or department may be needed to maintain it.
- *Customization.* Different services may provide different levels of customization/control.
- *Elasticity.* You may want to scale the amount of allocated resources to match your needs.

SOLUTION

Solutions for this problem can be seen as belonging to three categories:

- Purchasing and maintaining your own hardware allows you to have full control over your infrastructure. You can control, for instance, in which machine does a specific applications run, how that machine is configured, etc. This option represents therefore the highest level of customization *customization*.
  On the downside, this options usually means that you have to purchase hardware (*upfront costs*)and that you either acquire more resources than what you need or you risk not having enough resources to answer increasing computing needs. Additionally you will have to create and support a team or department to manage the *maintenance* of your infrastructure.

- When using IaaS there is no need to purchase anything upfront. In this pay-as-you-go model you only pay for what you consume and you are able to *elastically* increase/decrease the size and/or number of resources you are using. With this model, the responsability for maintaining and setting up infrastructure is shifted to the cloud provider.
  IaaS providers usually allow you to have some degree of *customization* like choosing the operating system and resources available(CPU cores, memory, etc) but lower level configurations will not be available. As a matter of fact, most cloud providers use virtual machines to run their clients applications meaning that you will not be able to tweak network configurations or choose exactly wich machine runs what. IaaS reduces therefore the level of control in comparisson to hosting your own infrastructure.

- Paas follows the same pay-as-you-go model as as IaaS meaning that you can also increase/decrease the size and/or number of resources you use.
  PaaS represents the smallest level of customization but at the same time allows you to use already pre-configured environments in which you can run your applications.

RELATED PATTERNS

When using the *Cloud*, choosing for instance a PaaS alternative may, depending on your choice, prevent you from being able to fully reproduce the production environment. Different levels of support exist, as well, for different *Reproducible Environments* 4.5 techniques. Some providers may, for instance, create pre-programed container ready environments while others may not allow you to run your own Virtual Machine.

4.5 Reproducible Environments

CONTEXT

When you have several environments (e.g. production,staging,devevlopment) or multiples instalations/instances of your software it is desirable to be able to guarantee that all instances work the same way. With this goal in mind you have identified that the environments where your instances

run is a key factor when trying to antecipate how does the software behaves.
This consistency is important because it will allow you to have reproducibility and will give you some guarantees when you desire to increase the number of instances of your software.

PROBLEM

How do you guarantee that the environment where you setup your application is consistent across instances?

FORCES

- *Size.* Having a complete copy of your environment (OS's, libraries, etc) may create large files that may be hard to move around.
- *Parallelization.* You may want to have several running instances of different environments in the same machine.
- *External Dependencies.* Some of your dependencies may be fetched from external providers.
- *Update.* You may want to update or change the environment.
- *Infrastructure.* Depending on your choice for *Cloud* 4.4 you may have more or less access to your environment settings.

SOLUTION

- Using scripts usually means describing you environment in the form of a text file. This file is then executed/interpreted inside an environment in order to create the desired state.
  Because scripts do not contain the dependencies you need, you usually have to fetch you may need to fetch you *external dependencies* from external providers. If for instance one provider is down, your script will not be able to complete.
  In case you need to *update* your setup, depending on the change and the tool you use you may need to run the script again, run only the part you modified or reset the machine and run everything again.
  Because scripts are just text files they often represent the most efficient alternative in terms of *size*.

- Using virtual machines an environment can be created by creating an image of the operative system with all dependencies installed. This image can then be replicated across different projects. If the need to change dependencies arise a new image can be created. Because of their size, usually only one or two VM's can coexist in the same computer at the same time (*parallelization*).

- **Containers** Containers are a lightweight alternative to Virtual Machines. The setup process is pretty similar to VM's but the generated representation/image is much smaller. This decrease in *size* comes from the fact that containers share resources with the host machine and even among containers. By doing so, it is usually possible to have multiple containers running in the same host. Some cloud providers already have options were they support containers natively.

Related Patterns

Having chosen to have environment consistency across your instances means that the when creating new instances(*Deploying new instances*) and building your software (*Continuous Integration* 4.8) the same choices should be used.

4.6 Deploying new instances

Context

You have decided to increase your computing resources **horizontally** in order to increase your hability to handle a bigger load of tasks. Depending on what type of *Cloud* you choose to use new resources were allocated but you still need to have your application running on those resources.

Problem

How do you deploy your application in a reproducible and consistent way?

Forces

- Deployments must be reliable.
- Deployments should not waist time.
- Deployments should correctly articulate with your environment setup method.
- Deployments should be easy to manage.

Solution

In order have an efficient deploy both in terms of reliability and speed you should have some sort of reproducibility. Depending on your choice for setting up environments (*Reproducible Environments*) there are different ways you can manage the deploy:

- If you have chosen *Containers* you can simply pull the container from a container registry and run it in your new instance. With this approach you will have a high degree of certainty that your instance will behave as you predict. Because containers are generally lightweight you will be able to download them fairly quickly. You will, nevertheless be dependant on you container registry service.
- If you have chosen *Virtual Machine* you can create and image (AWS lets you crete a Virtual Machine using their services) and then deploy it to all your instances.
- If you have chosen *Scripts* you can simply run the script in your target machine.

RELATED PATTERNS

Depending on your choice for *Reproducible Environments* 4.5, you should use the appropriate choice to deploy your code. You should be aware of the cost of this choice because if you want to scale horizontally, your performance maybe tied to the velocity with which you can provision new environments.

4.7 Scalling

CONTEXT

Having a 1:1 ratio between your needs and your resources may be easy to achieve if your needs are fixed in time. If, however, your needs fluctuate as a result of, for instance, new users accessing your application you would want to be able to **increase** or **decrease** (in case users numbers drop) the **resource** allocated.

PROBLEM

What strategy do you choose to increase your computing power ?

FORCES

- Costs are a factor.
- You want to change the allocated resources quantity without having to stop the existing application(s).
- Your application may have need to keep state.
- You may not have an upper limit for the amount of resources you will be using.

Solution

Usually if your are using the *Cloud* you can easily allocate new machines or increase the CPU cores, RAM, Disk Space, etc of your current machine(s). This two options represent the two existing approaches to scale your computing resources. The first one (increasing the number of machines) is usually refered to as **Horizontal Scalling** and second approach (increasing the resources of each machine) is usually refered to as **Vertical Scalling**.

Horizontal Scalling usually is the **cheaper** option and allows for **no downtime** when upgrading (the existing machine can be put into production while the old one is running). This approach will also allow you to **scale** virtually **without** a **limit**. On the downside, this approach will force to have some considerations in mind concerning state keeping. If you have a need to keep sessions, for instance, and you are storing them in the machine, the new machines will not have access to that.

Vertical Scalling is usually more expensive and depending on your *Cloud* provider may have associated downtime. **Verical Scalling** also has a maximum amount of resources you can allocate to a single machine. On the upside if you scale vertically(and have only one instance) you can keep the state of your application inside your machine.

Both approaches can be combined in order to accomodate your needs.

4.8 Continuous Integration

Context

There are several people contributing code to your application.

Problem

Having several people collaborating into the same project can be challenging. If a developer, unaware that is introducing an error, pushes code it into the team repository a long time may pass before the error is detected. Once detected, the error cause must identified and, because the code that introduced the error was pushed a long time ago, it may not seem obvious where the error is.

Forces

- Running your entire test suit before pushing code may take to much time.
- It can be challenging to setup an environment simillar to the production one in your local machine.
- Your environment may need to be different from the production one(you may need some extra tools to aid you developing).

- Your environment may be subject to bias (ex: case where you may manually set an environment variable that you code uses).

Solution

Use (or develop) an automatic continuous integration(CI) system. This system should detect when code is pushed to your repository and then run the following steps:

- **Build**ingyour software consists in, depending on your choice for *Reproducible Environments*, building your environment, then fetching all required dependencies and finally compiling the code(if needed).
- **Test** your build. When your build is successfull you should run your test suite against that build in order to check if everything is running according to plan.
- **Notify** If any of the previous steps fails you should notify the developer that checked the code and any other people to whom the build integration status is relevant.

Related Patterns

The *Continuous Integration* pattern can use the *Communication* defined channels to send its messages.
The test and build steps of this pattern, shoud be done in a environment equal to the production and development one and should follow the chosen *Reproducible Environment* 4.5 strategy.

4.9 Job Scheduling

Context

Sometimes there are tasks that, due their complexity may take a long time to finish. Cases may also exist when you have tasks that you want to schedule for later(e.g. maintenance tasks may be runned at a time when your application is under).

Both this problems can be solved by scheluding jobs to be run when possible or later.

Problem

How do you setup your infrastructure to handle this cases?

Forces

- Having a fixed set of resources for dealing with scheduling tasks may not be cost effective.
- Your load may vary during the day.

Solution

You may launch new instances of your infrastructure to handle each of you tasks or batches of tasks. Each new instance receives the desired tasks and does the needed computation. When the task as been computed the new piece of infrastructure should be shutdown.
Alternatively you can have a set of daemons running alongside your applications that handle this tasks.
Tasks are generally fed through a queing system altough you can also store them in a database.

Related Patterns

When you are launching new pieces of infratructure to handle your jobs the effectiveness of that technique may be dependant on how fast you can *Deploy new instances* 4.6.

4.10 Auditability

Context

As applications grow identifying potential problems within your infrastructures will become increasingly difficult. If you have several machines and/or different possible points of failure you can not predict or assume that everything will always go without incident and you will therefore have to be prepared. Building a robust system may seem enough but is not. When problems appear (and they will appear) being able to identify them ,where and why they appear is essential for the resolution of those problems.

Problem

What metrics should you extract and what should you do with them?

Forces

- Extracting too many metrics may cluter your hability to effective analyse them.
- You may want to keep an history of how your system behaved.
- You want to have information about the current state of your service.

Solution

Monitoring your application health can be done by using your own or external tools. Some cloud providers even provide you with a health view that tells you if your machines are healthy and running. Identifying some key indicators and some metrics is also important, by defining thresholds for each metric you can setup different levels of alerts for your teams. This way you can tackle problems as soon as they happen. Additionaly some indicators can also trigger automatic responses (e.g. if a platform is taking to long to answer requests you may launch new resources to distribute the traffic).

RELATED PATTERNS

You must be monitoring some metrics in order to create alerts. This mean that the *Alerting* 4.11 pattern will only exist if there are metrics being monitored.

4.11 Alerting

CONTEXT

You have defined a set of metrics for checking the health of your application. For some of those metrics when values reach a certain level a solution can not be automated (e.g. server repetitive failures). You still would want an immediate responese to that alert in order to make sure your services will not go down or in order to put them back on.

PROBLEM

Who are you going to call?

FORCES

- People may not be available to answer alerts or may be unreacheable.
- Alerting everyone may solve your problem quickly but may not be needed.

SOLUTION

Notifying can follow three main strategy. The first one is to wake everyone up. This approach is wastefull and as you usually do not need your entire team to solve the problem. The second one is always notifying the same person. This person should preferably be someone capable of diagnose the origin of your problem and then solve it or contact someone that can. The third option is two have a system were the responsability of handling errors rotates among the team members. Alerts can be sent using email, calling people, sending an sms and/or sending a notification to the person you want to notify.

4.12 Error Handling

Context

It is important not only to be able to detect errors but also to be able to respond to them in a proportional way.
As someone involved in a team developing a product, being able to find the best way to handle a crysis may prove to be fundamental.

Problem

In case of error what should the response be?

Forces

- *Impact.* Errors have different degrees of impact.
- *Hard bugs.* The cause of an error may not be easy to find.
- Sometimes you can not use a previous backup (e.g. when you have removed a column from your database).

Solution

Handling and error is a delicate task. There are several things that need to be taken into account.

If an error as a direct and significant *impact* in your applicaion (e.g. there is an error that allows people to login without checking the users passwords) you would want to respond as quickly as possible. In this types of cases you can rollback to an older version of your software. The rolling back effectiveness is nevertheless constrained by the speed with which you can do it and by the fact that you may not be able to do it. Strategies for rolling back you application can be of two types:

- **Deploying the previous version**: You can order your system to deploy a version of your software that you know works.
- **Keeping a backup** : You can keep a backup of your application/infrastructure and if an error is detected you can switch the DNS servers to point to your old infrastructure.

When you can not roll back and/or the error you detected does not have a substancial impact, you can try to find and fix the problem. Depending on your choice for *Version Control Organization* 4.3 you can create a newer hotfix branch and work on that. In the end, when you have found and fixed the problem, merging that branch with master and deploying the version will have fixed your problem.

Related Patterns

When you choose to rollback of to do a hotfix, you will be limited by the time it takes you to change the running version on your machines. If this change includes changing the environment you will need to update it. In order to do so, you will probably have to use the same method you defined for *Deploying New Instances* 4.6.
Additionally, rollbacks may depend on the fact that you can detect what your last working software version was. This information can be store in your version control system if you used tags for instance.
Information about the hotfix (if you chose to do one), can also be stored in you version control system(*Version Control Organization* 4.3)

4.13 Code Review

Context

You have several members of your team with different levels of knowledge and you want them to get to the same level and learn from each other. Additionally, you also want your team to have a global perception of the project and to guarantee some code quality standarts.

Problem

What practice could you employ to create this ?

Forces

- *Assurance.* You may have a critical module that you want to make sure does not break.
- *Knowledge sharing.* You want all developers to participate.

Solution

Some of the companies that we interviewed like Shiftforward or Codacy, had implemented a code review process to handle this types of situations. Rather than merging their changes directly to the master or developing branch, developers would create a pull request that would be reviewed by a different developer.
To some companies, the *Assurance* force would be the most important and the reviewer would be a technical owner of the module or a senior developer.
Other companies would not be so concerned about the *Assurance* force and would allow any developer to review the pull request increasing the *knowledge sharing*.

RELATED PATTERNS

Pull requests can be organized by the choice for *Version Control Organization* 4.3 .

Chapter 5

Validation

In this chapter we describe validation process for the pattern language presented in the chapter 4.

5.1 Methodology

The methodology used to validate the pattern language defined in chapter 4 had three main steps. The first step was to identify a company that wanted to incorporate some of the DevOps ideals into their workflow. Then, a process of identification of possible improvements would be done and a set of metrics would be identified and measured indicating if the process was successful or not.
By analysing the existent patterns and finding the best fit to the problems, the company would then be able to guide its adoption knowing beforehand which were the consequences and impact of their choise.
Finally, the indicators would be measured again. If the indicators improved, then the pattern language was able to guide a successful adoption of DevOps practices.

5.1.1 Ventureoak

We choose to do our validation at VentureOak. Ventureoak is a startup currently operating at UPTEC, Porto that started its activity in 2014. Having more than 20 employees, Ventureoak focus is on developing software products for other companies and in offering consultancy solutions both in the product idealization and developing phases.
At Ventureoak projects have usually a short duration - 3 to 6 months - and they usually target the web market. Several projects are developed concurrently at Ventureoak by teams of 2 to 6 elements. This teams are self organizing and are usually composed of developers.
The infrastructure is managed manually by the development teams and hosted in the Cloud.

5.1.2 Objecties & Metrics

Ventureoak wanted to improve and automate its processes. Before beginning this study, Ventureoak set the following goals in order to guide the adoption of some of DevOps practices:

- Reduction of the need for developers to handle deployments for the staging environment.
- Reduction of the time it took to setup the project in the developers machines.
- Improve consistency across the developers machines in order to reduce configuration errors.
- The company wanted to start migrating projects to the newer version of PHP.

With these objectives in mind, the follwoing metrics were collected:

- Deployment strategy
- Deployment duration
- Deployment cadency
- Environment setup strategy
- Environment setup time
- Ease to make environment changes
- Build errors frequency

5.1.3 Initial State

The initial values for each metric were as follows:

Metric	**Observed Value**
Deployment strategy (staging)	Manual
Deployment duration (staging)	3 to 10 minutes
Deployment cadency (staging)	When needed
Environment setup strategy	Manual
Environment setup time	5 minutes to several hours depending on the developer level
Ease to make environment changes	Low
Build errors frequency	Low

Table 5.1: Initial metrics identified

5.1.4 Adoption Process

The adoption process was done in a period of two weeks in one of the projects being developed at Ventureoak. The project was composed of three services. Two of the services used PHP and the remaining service used Javascript. There were additionally, two types of database being used: Redis and MongoDB.

Reproducible Environments

Having defined that there was a need to reduce the time it took to configure new environments and to improve the consistency of environments, it was obvious that the *Reproducible environment* pattern should be used as it was an exact fit.
This pattern presented us with three possible solutions. Wanting to have an environment as similar as possible with the production one were each service would run separately and because we had to have different versions of PHP running in different services we choose the Containerization option.

Continuous Integration

Wanting to change the environment easily meant that there should be a way for developers to be sure that the environment was correctly specified and that when they changed something that would not arm the system. Additionally, having a need to decrease the time it took to deploy the code for the staging environment, it was determined that having a pre-built container image of the environment with the needed code was beneficial.
This lead us to apply the Continuous Integration pattern.

Deploying new instances

In order to deploy the newer version, we used the *Deploying new instances* pattern. The process was triggered by the CI tool and managed by the cloud provider that would pull the pre-built container image and run it.

5.1.5 Final State

In the end of the adoption process we measured the defined metrics again obtaining the following results:

Metric	**Initial Value**	**Final Value**
Deployment strategy (staging)	Manual	Automatic
Deployment duration (staging)	3 to 10 minutes	15 minutes
Deployment cadency (staging)	When needed	On every push
Environment set up strategy	Manual	Automatic
Environment set up time	5 minutes to several hours depending on the developer level	Dependant on the connection speed, but usually less than 5 minutes
Ease to make environment changes	Low	High
Build errors frequency	Low	Medium

Table 5.2: Results from the use of the pattern language

5.1.6 Discussion

Except for the deployment duration, we observed a significant improvement between what was the initial state and the final one. Additionally, because we automated some of the processes, the ability to determine which version was running in the staging environment could be transferred to any one which could allow Project Managers to set up the staging environment with version the they wanted.
Unfortunately, the validation process only used three patterns from the 13 available. This means that we were only able to measure the effects of 23% of our pattern language and because of that we considered this process to have been inconclusive.

Chapter 6

Conclusions

6.1 Contributions

As of this moment, DevOps is still evolving and as more people join the discussion more perspectives and experiences will become part of the movement. Regardless, we believe that the initial work made in this thesis towards a more structured way of presenting and talking about DevOps will help reduce the barriers for new adopters and help the movement grow by providing a common dialect for discussing DevOps. This work can also be seen as a starting point for further investigation.

6.2 Future Work

6.2.1 Specializing the identified patterns

While pursuing the study of DevOps we kept a high level of granularity with the objective of capturing a wider view of the movement and its practices. We believe that based on this work, each pattern can be further analyzed in a more precise way.

6.2.2 DevOps monitoring

As referred before, the DevOps movement is still evolving. We believed that, by monitoring the DevOps movement it will be possible to identify more practices being employed by companies and individuals.

6.2.3 Further validation

The initial results observed hint that DevOps and its practices can bring benefits to both companies and individuals. In this thesis we were not able to fully validate all of the identified patterns and we also did not observe long term effects of this developments in both teams and companies. Validation was also done with only one team and one company, which is insufficient to prove that those benefits can be reproduced in other teams or organizations.

6.2.4 Statistical Analysis

A statistical analysis of the practices and methodologies employed by startup companies should also be developed in order to validate if the presented patterns are indeed a common practice among them.

Bibliography

[1] C. Alexander. *A pattern language: towns, buildings, construction*. Oxford University Press, 1977.

[2] J. Allspaw and P. Hammond. 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr.

[3] S. S. And and B. Coyne. *DevOps For Dummies*, volume 53. John Wiley & Sons, Inc., 2015.

[4] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. 1 edition, 2015.

[5] A. Coleman. Portugal Discovers Its Spirit Of Entrepreneurial Adventure, 2015.

[6] D. Cukier. DevOps patterns to scale web applications using cloud services. *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity - SPLASH '13*, (Figure 2):143–152, 2013.

[7] Dave Sayers. Why DevOps Is Like An Onion, 2013.

[8] J. Davis and K. Daniels. *Effective DevOps*, volume 1. 2015.

[9] P. Debois. Agile infrastructure and operations: How infra-gile are you? *Proceedings - Agile 2008 Conference*, pages 202–207, 2008.

[10] S. Elliot. DevOps and the Cost of Downtime: Fortune 1000 Best Practice Metrics Quantified. *IDC Insight*, (December), 2015.

[11] F. Erich, C. Amrit, and M. Daneva. Report: DevOps Literature Review. (October):27, 2014.

[12] G. Garrison, S. Kim, and R. L. Wakefield. Success factors for deploying cloud computing. *Communications of the ACM*, 55(9):62, sep 2012.

[13] M. Hüttermann. *DevOps for Developers*. Apress, Berkeley, CA, 2012.

[14] R. Johnson, E. Gamma, R. Helm, and J. Vlissides. Design patterns: Elements of reusable object-oriented software. *Boston, Massachusetts: Addison-Wesley*, 1995.

[15] M. Kircher and P. Jain. *Pattern-Oriented Software Architecture, Patterns for Resource Management*, volume 3. John Wiley & Sons, 2013.

[16] X. Labs. Periodic Table Of DevOps Tools.

[17] M. Loukides. *What is DevOps?* "O'Reilly Media, Inc.", 2012.

[18] P. Mell and T. Grance. The NIST definition of cloud computing. *NIST Special Publication*, 145:7, 2011.

[19] G. Meszaros and J. Doble. A pattern language for pattern writing. *Pattern languages of program design*, pages 1–36, 1997.

[20] D. W. W. Royce. Managing the Development of large Software Systems. *Ieee Wescon*, (August):1–9, 1970.

[21] D. C. Schmidt. Using design patterns to develop reuseable object-oriented communication software. *Communications of the ACM Special Issue on Object-Oriented Experiences*, 38(10):65–74, 1995.

[22] T. B. Sousa, F. Correia, and H. S. Ferreira. DevOps Patterns for Software Orchestration on Public and Private Clouds. page 11, 2015.

[23] Startup Europe Partnership. SEP MONITOR PORTUGAL RISING: MAPPING ICT SCALEUPS. 2015.

[24] Startup Lisboa. Startups.

[25] M. H. Syed and E. B. Fernandez. Cloud ecosystems support for internet of things and devops using patterns. In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 301–304. IEEE, 2016.

[26] Uptec. Companies.

[27] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds. *ACM SIGCOMM Computer Communication Review*, 39(1):50, dec 2008.

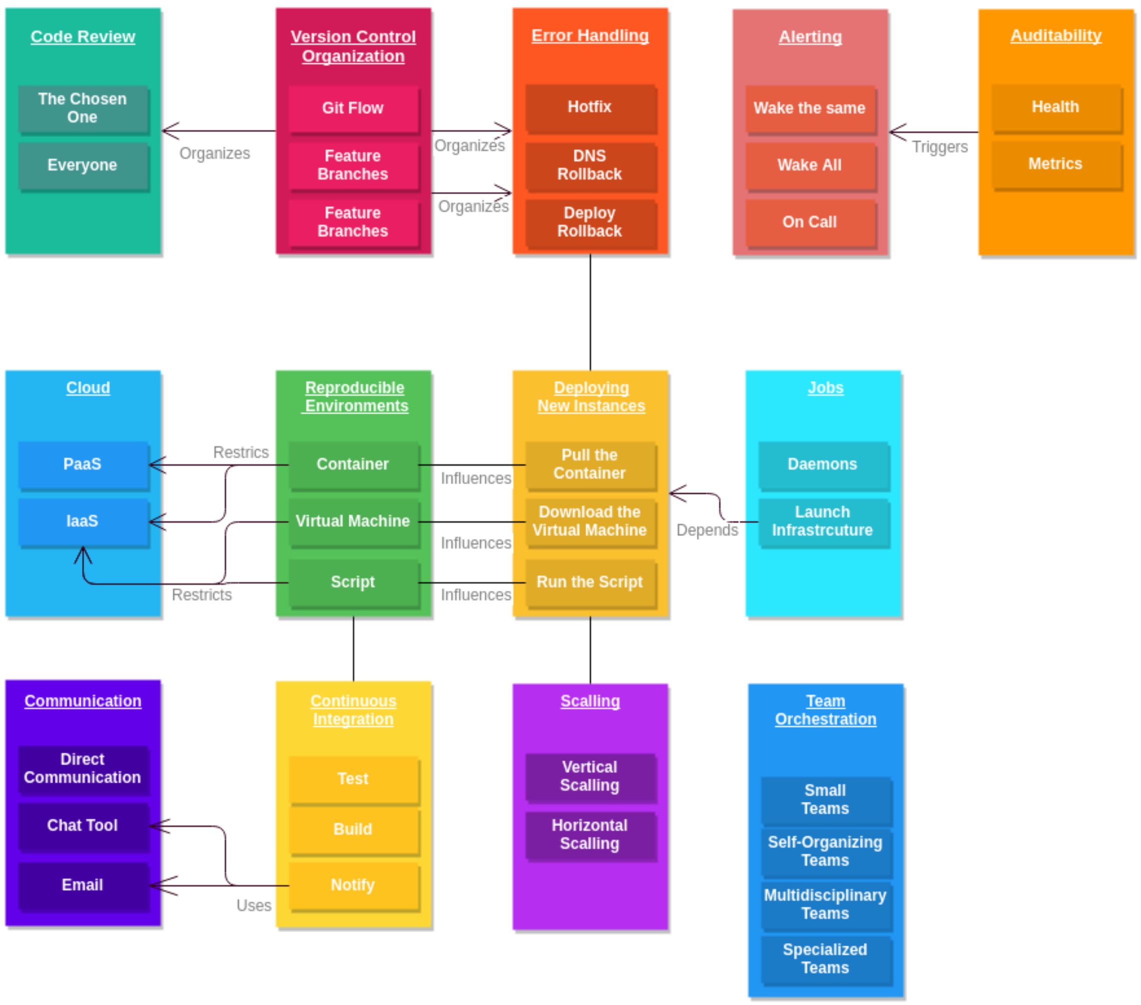
[28] J. Willis. What Devops Means to Me, 2010.

[29] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.

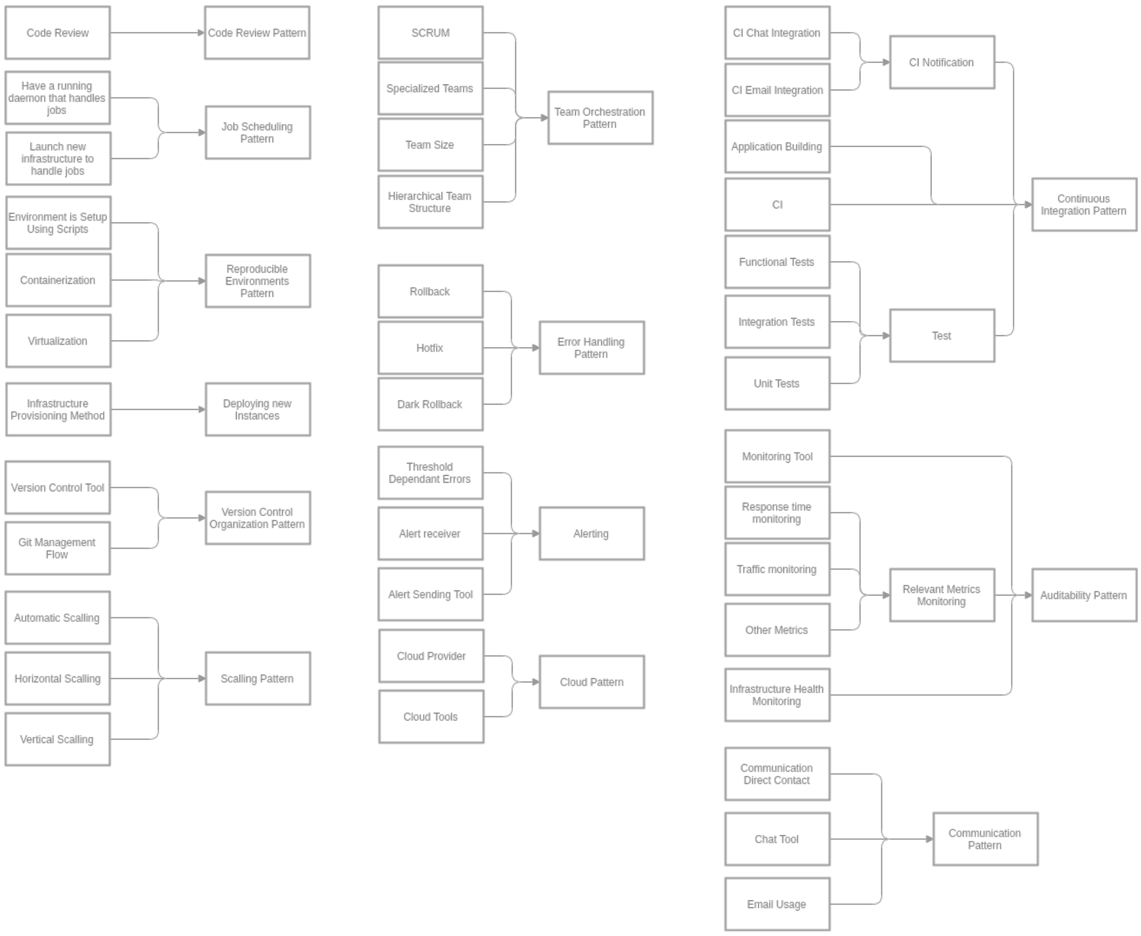
Appendix A

Annexes

A.1 Pattern Map



A.2 Concepts Grouping



	Abyssal	Arealytics	Atiiv	Nmusic	Zarco	Blip	Celfinet	clickly	Codacy	Emailbidding	EZ4U	Hypelabs	Imaginary Clou	Iterar	Jscrambler	Mindera	Semasio	Shiftforward	Streambolico	TOPDOX	Ubiwhere	Unbabel	Virtus.ai	3Port	Weduc		Practice Frequency	Pattern Related Practices Frequency	Pattern
Code Review	✓								✓	✓			✓			✓	✓	✓		✓	✓	✓			✓		11	11	Code Review
Jobs(Daemons)											✓																1		
Jobs(infrastructure)									✓					✓													2	3	Jobs
Environment Setup (scripts)			✓	✓		✓	✓						✓											✓	✓		7		
Environment Setup (containers)									✓	✓			✓		✓			✓				✓	✓				7		
Environment Setup (VM)			✓			✓							✓			✓	✓					✓					6	15	Reproducible Environments
Infrastructure Provisioning (pull container)									✓	✓			✓					✓					✓				5		
Infrastructure Provisioning (download VM)						✓							✓			✓						✓					4		
Infrastructure Provisioning (run script)			✓	✓		✓	✓						✓														5	11	Deploying New Instances
Version Control (Git)	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓		✓		20		
Version Control (Mercurial)							✓																				1		
Version Control (SVN)																								✓			1		
Version Control (Feature Branches)	✓	✓	✓		✓			✓					✓		✓	✓		✓									9		
Version Control (Github flow)										✓							✓		✓	✓							4		
Version Control (Master Only)				✓		✓																					2	25	Version Control Organization
Automatic Scalling								✓	✓									✓									3		
Vertical Scalling	✓		✓	✓	✓			✓	✓	✓	✓		✓			✓		✓									11		
Horizontal Scalling						✓		✓	✓	✓			✓			✓		✓									7	12	Scalling
SCRUM	✓			✓		✓			✓	✓						✓					✓	✓					8		
Teams (Specialized)	✓					✓	✓																				3		
Teams(Size)	4-6	2	3	1-4	8	7-9	5-8	6	1-6	2-6	2	7	2-3	5	4	5-6		8	9	1-3	1-5		4	1-3					
Hierarchical Teams Structure																											0	9	Team Orchestration
Error Handling (Rollback)				✓		✓	✓	✓	✓				✓			✓						✓					8		
Error Handling (Rollback Dark)															✓	✓											2		
Error Handling (Hotfix)	✓		✓	✓		✓		✓	✓	✓	✓		✓			✓				✓	✓						12	15	Error Handling
Errors (thresholds)				✓					✓	✓			✓			✓		✓		✓	✓						8		
Errors (alert team)				✓						✓						✓		✓		✓	✓	✓					7		
Erros (alert defined group)									✓	✓												✓					3		
Alert channels				Email,SMS					PagerDuty		SMS,Email					Chat, Email											4	9	Alerting
Cloud Usage	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	✓	✓		✓		21		
Clout Tools	AWS	Dig. Ocean, AWS		AWS, PT Cloud		AWS, own	Azure	AWS	AWS	AWS	AWS		AWS,Linode,..	Heroku	AWS	AWS		AWS		Azure	Google Cloud	AWS, New Relic	Azure		AWS		19	21	Cloud
CI chat Integration									✓						✓					✓	✓	✓					5		
CI email integration													✓			✓						✓					3		
CI	✓			✓		✓			✓	✓			✓		✓	✓		✓		✓	✓	✓					12		
CI Build Step	✓			✓		✓			✓	✓			✓		✓			✓		✓	✓	✓					11		
Functional Tests														✓		✓											2		
Unit Tests	✓			✓		✓	✓		✓	✓	✓		✓			✓					✓	✓	✓				12		
Integration Tests				✓						✓																	2	16	Continuous Integration
Monitoring tool	✓				✓				✓	✓	✓		✓	✓	✓	✓		✓			✓	✓			✓		13		
Monitoring(Response Time)	✓							✓	✓	✓	✓		✓		✓	✓		✓		✓	✓	✓			✓		13		
Monitoring(Traffic)	✓		✓					✓	✓	✓	✓		✓		✓	✓		✓		✓	✓	✓			✓		14		
Monitoring(Health)	✓		✓					✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	✓			✓		15	17	Auditability
Communication(direct)	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		24		
Communication(chat)	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		23		
Communication(email)	✓					✓	✓	✓	✓	✓	✓	✓	✓		✓	✓									✓		12	25	Communication

A.3 Interview Guide

Product

- What type of product do you develop ?
- What is the scale of that product ? Number of countries, number of users ?
- Do you have some kind of SLA or some requirements that impacts your work ?

Teams

- How many teams do you have ?
- What is the size of each team ?
- Are teams specicalized or do they have multiple specializations working together ?
- Do teams interact with each other ?
- How are teams seen from an external perspective ? Are they autonomous?
- How do you manage your workload ? Do you use SCRUM,Kanban or other ?
- How do team members communicate among themselves.

Pipeline

- How long does your code take to go from idea to production ?
- What are the states that you code goes through before reaching a production environment
- What triggers the transition between states ?
- What kind of tests do you develop? Which teams are involved in that process ? When do they run?
- What happens in each of the pipeline states ?
- In each state, which teams intervene and what do they do?
- What processes did you automate ? Did you choose to not automate some state ? If so, why?
- How do you handle your deployment process ? Which tools do you use ? Do you use containers or VMs ?

Infrastructure Management

- How do you scale ? Horizontally or vertically?
- Does scalling happen automtatically ?
- What can make infrastructure scale up/down ?
- How is infrastructure increased ?
- How do you lift new instances of your infrastructure? Is it automatic?

Monitoring & Error Handling

- What metrics do you collect from running servers ?
- What do you see as errors ?
- What process do you follow to solve errors after they are detected ?
- When errors are detected, who is notified ? How is the notification sent ?
- If errors are detected before the software reaches production, what do you do ?

A.4 Companie Interviews Report

Abyssal

PRODUCT

Abyssal is a startup company focused on developing Subsea Navigation Systems for Remotely Operated Vehicles (ROVs) operating since 1 de Fevereiro 2012.
Their goal is to facilitate the access to these locations by developing intuitive and precise software solutions to the major problems affecting ROV piloting and supervision: poor navigation, reduced visibility and lack of spatial
awareness.
The current solutions has two major modules. One that renders the underwater obstacles in the live video feed from the ROVs and a web one that handles the transmission of data from the rig/ship to shore.

TEAMS

Teams at abyssal are mainly divided by area of specialization and have between 6-4 people. Nevertheless, roles inside each team sometimes change(see below). Teams communicate both directly and through Slack. Confluence is used as as a collaboration platform.
In order to manage the project, Abyssal uses SCRUM.

DEVELOPMENT

The development process at abyssal focus on three languages : C++. C# and Python. In order to manage their code, Abyssal uses Git with feature branches.
Unitary tests are developed by the development team, manual tests are performed by one member of the development team (the role is changed every sprint). This unitary tests are then used in the Continuous Integration phase.
A code review process was also implemented. Every feature is analysed by one member of the development team. This member must not be the same one that did the feature.
Continuous Integration is managed by Team City. Team City is responsible for creating a build and running the existent tests.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Abyssal uses AWS services in order to manage their infrastructure. Due to well defined upper and lower bounds for users numbers and load scaling is done only manually at Abyssal.
Deployments and upgrades are mainly managed manually except for the Virtual Reality where a tool assists the the deployment process.
Logs are stored in the running instances and retrieved manually when needed.

Monitoring relies on AWS services and the main metrics monitored are server load and and disk space.
When errors are detected, Abyssal usually solves them by issuing a hotfix.

arealytic

PRODUCT

arealytic is a young company (5 month old in the time this interview was conducted) that wants to transform IP and other types of cameras into analytics tools. In order to do this, arealytic developed a web platform that analyzes camera footage and tracks user movements throughout a store as well as interactions with displayed products.

TEAMS

The arealytic team had at the time two members. Both worked in all aspects of the product and communicated either directly or by using Slack.

DEVELOPMENT

The development process at abyssal focus on three languages : PHP, Javascript and Python. In order to manage their code, arealytic uses Git with feature branches.

Atiiv

PRODUCT

Atiiv is a company that develops a web platform for Personal Trainers to monitor, register and prescribe training plans to it's clients.
At the time of the interview the company was roughly 1.5 years old and had 4 collaborators.

TEAMS

Teams at Atiiv have 3 people and communicate using several a chat tool.

DEVELOPMENT

The development process at Atiiv focus on two languages: PHP and Javascritp . In order to manage their code, Atiiv uses Git with feature branches.
Unitary tests are developed by the development team, manual tests are also performed by the development team.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Atiiv uses Digital Ocean and AWS services in order to manage their infrastructure.
Deployments and upgrades are managed by Laravel Forge (a PaaS service).
Monitoring relies on New Relic services and some of the metrics monitored are the server state and load.
When errors are detected, Atiiv usually solves them by doing a hotfix.

NMusic

PRODUCT

NMusic is a portuguese startup that develops a platform that allows other businesses to stream and synchronize music.

TEAMS

Teams at NMusic are mainly divided by speciality and have between 1-4 people. Teams communicate both directly for doubts and discussions and using Slack to share files and links.

DEVELOPMENT

In order to manage their code, NMusic uses Git with only a master branch.
Unit tests are developed by the Development team and manual tests are performed by a QA team.
Continuous Integration is managed by Jenkins and its main objective is to create a nightly build .
This tool is responsible for creating a build and running the existing tests.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

NMusic uses AWS and PT Cloud services in order to manage their infrastructure.
Deployments and upgrades managed by Capistrano .
Monitoring relies on Zabbix. If errors are detected depending on their severity different approaches can be adopted. If an error is not critical, an internal interface will be updated, if an error is bad but still not critical, an email is sent. In a case where an error is considered critical there are two developers that are notified by SMS. Some common solutions at NMusic can be issuing a hotfix or deploying the previous version.

Zarco

PRODUCT

Zarco is a mobile app that aims to link travelers with traveling guides.

TEAMS

The Zarco team has 8 members. The team communicates directlyfor doubts and discussions or by using Slack(to share files and links).

DEVELOPMENT

The development process at Zarco focus on Java, Javascript Swift and PHP. In order to manage their code, Zarco uses Git with Feature Branches.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Zarco uses AWS services in order to manage their infrastructure, deployments and monitoring.

Blip

Product

BLIP is a portuguese company that develops web applications and solutions in the betting exchange market.

Teams

Teams at BLIP are mainly divided by specialization and have between 7 and 9 people. Teams communicate directly or using Slack.
Projects are managed using SCRUM.

Development

The development process at BLIP focus on 2 languages: Java and Javascript . In order to manage their code, BLIP uses Git wit only a master branch.
Unitary and functional tests are developed by the development teams.
A code review process was also implemented in cases where a feature is seen as critical. When this happens the code is review by two developers, preferably from a different team. Continuous Integration is managed by Jenkins. This tool is responsible for creating a build, running the existent tests and running JSLint. Then, a deployment is made to an internal environment where functional tests are run and where exploratory tests can be made.
Environments are setup using Chef and Ansible.

Operations & Infrastructure Management

BLIP uses its own and AWS services in order to manage their infrastructure.
Deployments and upgrades managed by the Jenkins server. Whenever an upgrade is made, the new version is deployed in an inaccessible infrastructure from the outside world. When the upgrade has been done the DNS servers stop pointing at the old infrastructure and start pointing at the new upgraded one. This process is repeated for each deployment.
Monitoring relies on some internal services and the main metrics monitored are server.
When errors are detected, BLIP usually solves them by issuing a hotfix of if needed a rollback.

Celfinet

PRODUCT

Celfinet is consultancy and software development company. The company started 2012 and has currently around 300 employees.
The product analysed in this interview consists in a monitoring and auditing solution for mobile networks.

TEAMS

Teams at Celfinet are grouped into departments and are mainly divided by speciality. Teams have between 5 and 8 people. Teams communicate using (from less to more formal) TeamFoundation, Skype and Email.

DEVELOPMENT

The development process at Celfinet on the C# language. In order to manage their code, Celfinet uses mostly Git.
Unitary tests are developed by the development team, manual tests are performed by the QA team.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Celfinet uses Azure services in order to manage their infrastructure. Scaling is done manually by lifting a new instance in the Azure platform and then configuring it.
Deployments and upgrades managed by Octupus.
Logs are manually extracted and used for BI purposes.
Monitoring is handled by New Relic and Nagios services and the main metrics monitored are (server load, server state, latency, response time).
When errors are detected they can be solved by rolling back (if possible). If the error is not urgent then it can be solved in the next release.

clickly

PRODUCT

clickly is a web startup that curates content from all over the web, and matches it with interactive ads from brands.
Its technology monetizes online content by programmatically matching it to relevant advertisers through an interactive native ad unit.

TEAMS

The clickly team is not divided by speciality and they have between 6 developers.
Teams communicate directly and by using Slack.

DEVELOPMENT

In order to manage their code, clickly uses Git with feature branches.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

clickly uses AWS services in order to manage their infrastructure. Scalling is managed by the AWS Beanstalk service and occurs in case traffic increases.
Deployments and upgrades are also managed by AWS Elastic Beanstalk.
Logs are stored in the running applications and retrieved manually if needed.
Monitoring relies on the AWS services as well and the some of the monitored metrics are the network traffic.
When errors are detected, clickly usually solves them by issuing a hotfix. In cases where a hotfix is not possible the previous version is deployed.

Codacy

PRODUCT

Codacy is an automated code review tool that helps developers to save time in code reviews and to tackle technical debt. It centralises customisable code patterns and enforces them within engineering teams.
Codacy tracks new issues by severity level for every commit and pull request. It provides advanced code metrics on the health of a project and on the performance of teams.

TEAMS

Teams at Codacy are mainly divided by specialization and have between 1 and 6 people. Teams communicate directly or using Slack. SCRUM is used to manage the project.

DEVELOPMENT

The development process at Codacy is done in Scala . In order to manage their code, Codacy uses Git.
Unitary tests are developed by the developers.
A code review process was also implemented. Every feature is analysed by a module technical owner(each module/service has someone responsible for guaranteeing the quality of that module).
Continuous Integration is managed by Bamboo . This tool is responsible for creating a build and running the existent tests.
Environments are defined using Docker.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Codacy uses AWS ElasticBeanstalk services in order to manage their infrastructure.
Deployments and upgrades are managed by AWS and the rolling update strategy is used.
Jobs are managed by launching a new instance/container to perform the job.
Logs are stored and processed using the ELK stack.
Monitoring relies on Ruxit.
When an error is detected the IT member is notified.

EmailBidding

PRODUCT

Emailbidding is an Email Marketing Advertising Platform. A self-service, web-based platform for advertisers and agencies that allows them to segment and bid for the audience in opt-in publisher's networks.

TEAMS

Teams at Emailbidding are mainly divided by speciality and have between 2 and 6 people. Teams communicate using Skype (to share files and links) or directly (for doubts and discussions). Teams at Emailbidding are divided in two groups: Developers and IT Operations. Nevertheless, there is a competence center that organizes activities that aim to provide the operations groups with some of the developers points of views and vice-versa.

DEVELOPMENT

The development process at Emailbidding focus on several languages: Javascript, PHP, Java, ... In order to manage their code, Emailbidding uses Git with a branching system where several environments are mapped. First there is one branch for each issue. On each commit, codesniffer is run and if it passes, a pull request is created for the master branch. A CI tool(Circle CI) runs unit and integration tests on every pull request and a code review process is also implemented. If everything passes, the code is then released to a stagin environment where a QA process is done. The code then goes to production if everything goes as planned.
There are also additional branches for hotfixes. Environments are setup using Docker.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Deployments are managed using Fabric that manages asset building, clusters, CDN asset publication. Emailbidding uses a combination of AWS services with non cloud services.
Scalling is managed both manually(for expected load increases) and automatic in case the load increases without warning.
Logs are stored using Logentries, Stackdriver and Librato.
When errors are detected an email is sent to the developers and a SMS is sent to the VP of Engineering and the CTO. They are usually fixed by issuing a hotfix.

EZ4U

Product

EZ4U's platform allows sending SMS with global coverage. The platform allows:

- Recurrent Contacts and / or massive marketing campaigns
- Sending through the web or automated mechanisms
- White Label solution for agencies and resellers
- Sending to both national and international receivers
- Inbox with automatic message processing
- Integration with external systems: RESTful API - JSON/XML

Teams

The EZ4U development team has currently 2 developers.

Development

The development process at EZ4U focus on PHP. In order to manage their code, EZ4U uses Git.
Unitary tests are developed by the development team.

Operations & Infrastructure Management

EZ4U uses AWS beanstalk services in order to manage their infrastructure. Scaling is possible altough it is not automated. Deployments and upgrades are managed by AWS Beanstalk. Logs are stored in the running instances and retrieved manually if needed. Monitoring relies on the AWS services and the main metrics monitored are the sms sending speed variance, server load, network traffic, etc. When an error is detected the entire development team is informed either by sms or email depending on the severity of the error.

Hypelabs

PRODUCT

HypeLabs develops a cross-platform communications SDK that uses multiple transport technologies, such as Wi-Fi or Bluetooth, to create local mobile ad hoc networks, making devices communication-capable even if there's no Internet access.

TEAMS

The Hypelabs team is multidisciplinary and has 7 members. Teams communicate directly and using slack of facebook.

DEVELOPMENT

The development process at Hypelabs focus on three languages: C#,Java and Objective C . In order to manage their code, Hypelabs uses Git.

Imaginary Cloud

PRODUCT

Imaginary Cloud builds web and mobile applications.

TEAMS

Teams at Imaginary Cloud are mainly divided by project and have between 2-3 people. Teams communicate both directly and by using Slack.
Projects are managed using SCRUM.

DEVELOPMENT

The development process at Imaginary Cloud focus on several languages including: Ruby, Java, Objective-C,... In order to manage their code, Imaginary Cloud uses Git with Feature Branches.
Unitary tests, functional and acceptance tests are developed by the development team.
A code review process was also implemented. Every feature is analysed by a senior developer before being accepted.
Continuous Integration is managed by SemaphoreCI.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Imaginary Cloud uses (AWS|Digital Ocean|Linode..) services in order to manage their infrastructure. Scaling can be both vertical or horizontal.
Monitoring relies on the some external tenchnologies like new relic and Mixpanel services. Some of the metrics monitored are the server load, server state, latency, response time and exceptions issued and the ratio between logged users and users that are not logged.
When errors are detected the Imaginary Cloud usually solves them by issuing a hotfix or in case of a more severe case a roll back.

Iterar

PRODUCT

Iterar is a Porto based startup focused on web and mobile software development.

TEAMS

The Iterar team has 5 people. Everyone collaborates with each other and teams communicate directly or via Slack.
Project tasks and progress are managed by Trello and the team is managed in a semi agile style.

DEVELOPMENT

The development process at Iterar focus on Ruby, Java, Objective-C and Javascript. In order to manage their code, Iterar uses Git.
Some functional tests are developed by the development team that also performs some manual tests.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Monitoring is handling by New Relic(health and load) and Rollbar(exceptions).
Iterar uses Heroku services in order to manage their infrastructure.
Deployments, upgrades, workers launching are also managed by Heroku.
Application logs and monitoring metrics values are stored in Rollbar and New Relic.
When errors are detected the development team is notified.

Jscrambler

PRODUCT

Jscrambler is a Web startup that works on security products to protect Web and Mobile Applications. On of its products is a RASP solution to make apps self-defensive and resilient to tampering and reverse-engineering.

TEAMS

Teams at Jscrambler are mainly multidisciplinary and have 4 people. Teams communicate both directly (for doubts and discussions) and by Slack (to share files and links).

DEVELOPMENT

The development process at Jscrambler focus on Javascript. In order to manage their code, Jscrambler uses Git with feature branches. Unitary tests and functional tests are developed by the development team. Continuous Integration is managed by Jenkins . This tool is responsible for creating a build, running the existent tests and triggering the deployment process.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Jscrambler uses AWS services in order to manage, monitor and upgrade their infrastructure. The company also uses some proprietary monitors with the aim to enforce load and health monitoring.
When errors are detected the the entire team is notified. Depending on the error severity emails or sms are sent to the devs. Errors are usually solved by changing the DNS to a previous version of the service.

Semasio

PRODUCT

Semasio develops a Web Platform that works analyzes internet users and their habits in order show them better and more customized ads.

TEAMS

Teams at Semasio are mainly divided by speciality and have between 2-5 members. Teams communicate directly (for doubts and discussions) or by using Slack (to share files and links).

DEVELOPMENT

The development process at Semasio focus on C# . In order to manage their code, Semasio uses Git with github flow.
Unitary tests are developed by the development team. Manual and exploratory tests are performed by the QA team.
Teams use a Code review process to both ensure code quality and also to increase knowledge sharing. Every commit is verified on pull request and by an additional developer and a QA.
Continuous Integration is managed by Visual Studio Team Studio . This tool is responsible for creating a build and running the existent tests. Semasio uses CI in order to offload their developers with the responsibility of running the tests locally and also to detect errors early.
Environments are pre-setup in a external server and developers develop on that servers.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Semasio does not directly manage its infrastructure. When errors are detected the depending on the severity an email or a call can be made to a on call engineer.

Shiftforward

PRODUCT

Shiftforward develops two web platforms that allow companies to better analyze and predict traffic to their sites.

TEAMS

The Shiftforward team has no division and has 8 members. Teams communicate both directly and using Slack.

DEVELOPMENT

The development process at Shiftforward focus on Scala. In order to manage their code, Shiftforward uses Git with feature branches.
Unitary and functional tests are developed by the development team.
A code review process was also implemented in order to improve knowledge sharing and maintain quality. Every feature is analysed by preferably a developer that fully understands the module in question.
Continuous Integration is managed by Gitlab CI and is done to run all tests in a clean environment and mark pull requests with the build status and allow reviewers to have a way to know if the code is working without having to manually test it. As tests were beginning to take a while to run, the tool also allows developers to not have to run the tests locally. The CI tool is responsible for creating a build and running the existent tests.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

In order to manage its infrastructure, Shiftforward uses Marathon on top of Apache Mesos hosted in AWS.
Logs are collected using Logstash and are store in ElasticSearch.
Monitoring relies on the Marathon and Pingdom services and the main metrics monitored are server response time and server health.
When errors are detected the team is notified.

Streambolico

PRODUCT

Streambolico is currently developing a solution that allows venus to live transmit live media to mobile devices in a efficient way.

TEAMS

The Streambolico team is multidisciplinary and has 9 members. Teams communicate directly or using Bitrix24.
All of the team members are developers but one member has the responsibility of testing the software.

DEVELOPMENT

The development process at Streambolico focus on 3 languages: C, Objective-C and Java. In order to manage their code, Streambolico uses Git with Git Flow.
Tests are mostly done manually and some linters are also used.
Every week a new build is created manually.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Deployments and upgrades managed manually.
Logs are stored in the application servers and retrieved manually when needed.

TOPDOX

PRODUCT

TOPDOX is a plug and play platform for companies to connect on premise file servers and cloud storages. Providing the best mobile experience to their workers. Currently with more than 250K users worldwide and more than 200M files indexed by our platform.

TEAMS

Teams are organized according to the application they are developing for and rotation between areas is encouraged. Workgroup sizes range from 1 to 3 people.
Teams communicate both directly(for doubts and discussions) of by using Hipchat (to share files and links).

DEVELOPMENT

The development process at TOPDOX focus on several languages: Objective-C, Java, C#, Javascript,.... In order to manage their code, TOPDOX uses Git with git flow.
A code review process was also implemented(to assure quality, promote knowledge sharing and create some redundancy of knowledge). Every feature is analysed by a different developer before being accepted.
Continuous Integration is managed by Jenkins in order to have builds ready whenever needed. This tool is responsible for building the application.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

TOPDOX uses Azure services in order to manage their infrastructure.
Production error logs are extracted using rollbar.
Monitoring relies on the New Relic and the main metrics monitored are server health and response time.
When errors are detected the entire group is notified and TOPDOX usually solves them by issuing a hotfix.

Ubiwhere

PRODUCT

Ubiwhere develops IoT solutions for todays cities.

TEAMS

Teams at Ubiwhere are mainly divided by specialization and have between (1 and 5 members).
Teams communicate directly or using Slack.
The project is managed using SCRUM.

DEVELOPMENT

The development process at Ubiwhere focus on Python and Java . In order to manage their code, Ubiwhere uses Git.
Unitary and functional tests are developed by the development and manual tests are performed by the QA team (members of the QA team share several projects).
Code reviews are done in order to increase code quality, increase knowledge sharing and avoid that only one person knows each module.
Continuous Integration is managed by Jenkins and is done to detect errors early, avoid regression and integration errors, and for the entire team to know what tests failed . This tool is responsible for creating a build and running the existent tests.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Ubiwhere uses Google Cloud Engine services in order to manage their infrastructure. Deployments and upgrades managed Google cloud engine and Jenkins. Logs are stored in (do you retrieve and store logs) and retrieved manually when needed. Monitoring relies on the Sensu+ services and the main metrics monitored are server health. When errors are detected the everyone is notified and Ubiwhere usually solves them by issuing an hotifx.

Unbabel

PRODUCT

The Unbabel platform combines machine translation with a community of bilinguals and freelance translators which results in human quality translations in a on demand pay as you go translation service.

TEAMS

Teams at Unbabel are multidisciplinary and are mainly divided by the module in which they are working. Teams communicate directly or by using Slack.
An adaptation of SCRUM is used in order to manage the project.

DEVELOPMENT

The development process at Unbabel focus on Python, Java and Javascript . In order to manage their code, Unbabel uses Git.
Unitary tests are developed by the development team.
A code review process was also implemented.
Continuous Integration is managed by Jenkins and Circle CI . This tools are responsible for creating a build and running the existent tests.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Unbabel uses AWS services in order to manage their infrastructure, deployments and scaling.
Monitoring relies on the New Relic, AWS Cloud Watch and Bugsnag. Some of the metrics monitored are the server health, runtime exceptions and response time .
When application errors are detected, the entire development team is notified. When infrastructure errors are detected the same person is notified. Notifications are sent by email.
Errors are usually solved by deploying a previous working version.

Virtus.ai

PRODUCT

Virtus.ai is a software development company currently juggling multiple projetcts wile developing their core product Netpuno. Netptuno is a cloud platform that allows retailers to search for products in a natural way.

TEAMS

The Virtus.ai team has 4 members. The team communicates using Slack or directly.

DEVELOPMENT

In order to manage their code, Virtus.ai uses Git.
Unitary tests are developed by the team.
Environments are setup using Docker.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Virtus.ai uses Azure services in order to manage their infrastructure and deployments

3Port

PRODUCT

Project 3PORT aims at designing a Web-based Geographical Information System to register, control and manage environmental operations, processes and requirements, associated with any waterway port or seaport. Using geographic and georeferenced information, the solution allows any port authority to easily and completely visualise, treat and process all port authority related information in real time and virtually at any location.

TEAMS

Teams communicate using directly or by using hangouts and skype.
Teams have between 1 and 3 members.

DEVELOPMENT

The development process focus on C# and Javascript. In order to manage their code, SVN is used.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Deployments are managed using Visual Studio or by creating a script that runs in the client machine.

Weduc

Product

Weduc is a tool that allows schools to share informations and content with parents related with their child evoltuion.

Teams

Teams at Weduc communicates using email or skype.

Development

The development process at Weduc focus on PHP and Javascript. In order to manage their code, Weduc uses Bitbucket.
Unitary and acceptance tests are developed by the development team.
A code review process was also implemented. Every feature is analysed by a senior developer before being accepted.

Operations & Infrastructure Management

Weduc uses AWS services in order to manage their production infrastructure.
Deployments and upgrades managed through scripts.
Monitoring relies on Munin, Zabix and Pingdom.
When errors Weduc usually solves them by issuing a hotfix.

Mindera

PRODUCT

Mindera is a software development company.

TEAMS

Teams at Mindera are multidisciplinary and have between 5 and 6 elements.
Teams communicate directly (for doubts and discussions) or using Slack,Skype,gotomeeting and Zoom (to share files and links, to communicate with external clients and to talk with other team members). Projects are managed using SCRUM or Kanban.

DEVELOPMENT

The development process at Mindera focus on several languages which include: Java, javascript, Objective-C and C#. In order to manage their code, most projects at Mindera uses Git with feature branches. Unitary tests are developed by the development team.

Continuous Integration is managed by Jenkins and in some projects by Go (https://www.go.cd/). This tools are responsible for creating a build and running the existent tests as well as checking the code coverage. Both this tools are also responsible for starting the deployment process.

OPERATIONS & INFRASTRUCTURE MANAGEMENT

Mindera uses AWS services in order to manage their infrastructure, scaling and monitoring. Deployments are made by lifting a copy of the current infrastructure and then switching the DNS server entries. Monitoring and auditing (log keeping and extraction) relies also in the AWS services and the main metrics monitored are the server health and response time. Some additional tools are also used in this context, including pingdom and CloudWatch. When errors are detected the the problem is usually redirected to the company through the client and Mindera usually solves them by issuing a hotfix or a rollback