Hello,
My name is Daniel Gomes and I'm the manager of the Arquivo.pt web archive.
I am going to share our experience on improving the robustness of our service.
I believe that our lessons learned will be useful to any person involved in Information Technology projects.

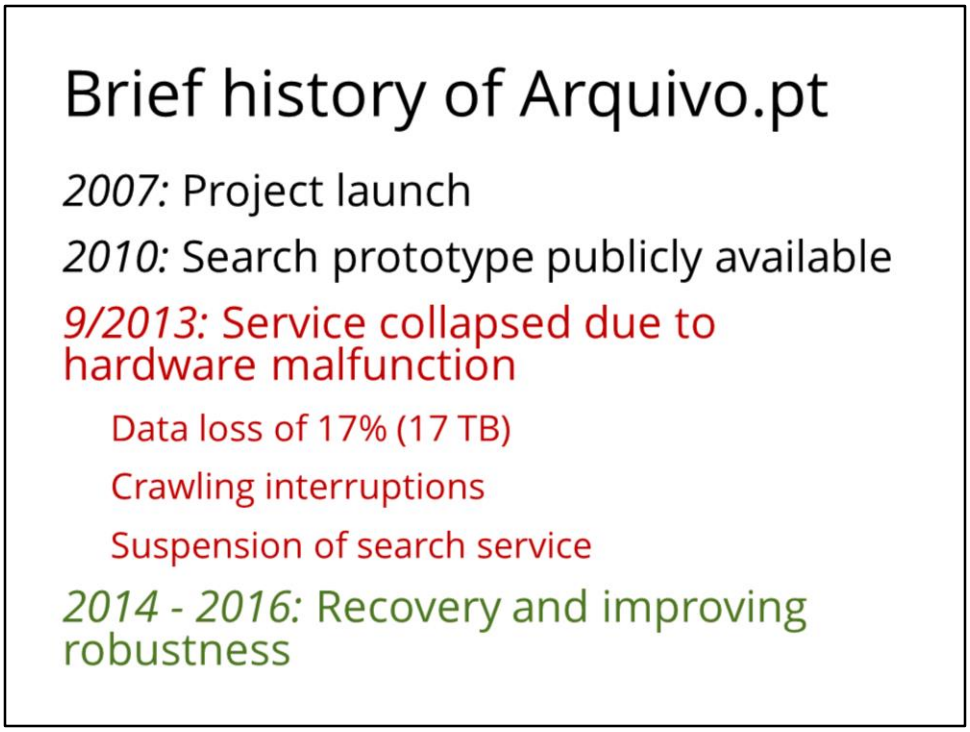
What is Arquivo.pt?

Web pages preserved since 1996

Public search service

Information in several languages

Arquivo.pt preserves millions of files archived from the web **since 1996.** It provides a **public search service** over this information. It preserves information written in several languages and it provides user interfaces in Portuguese and English.



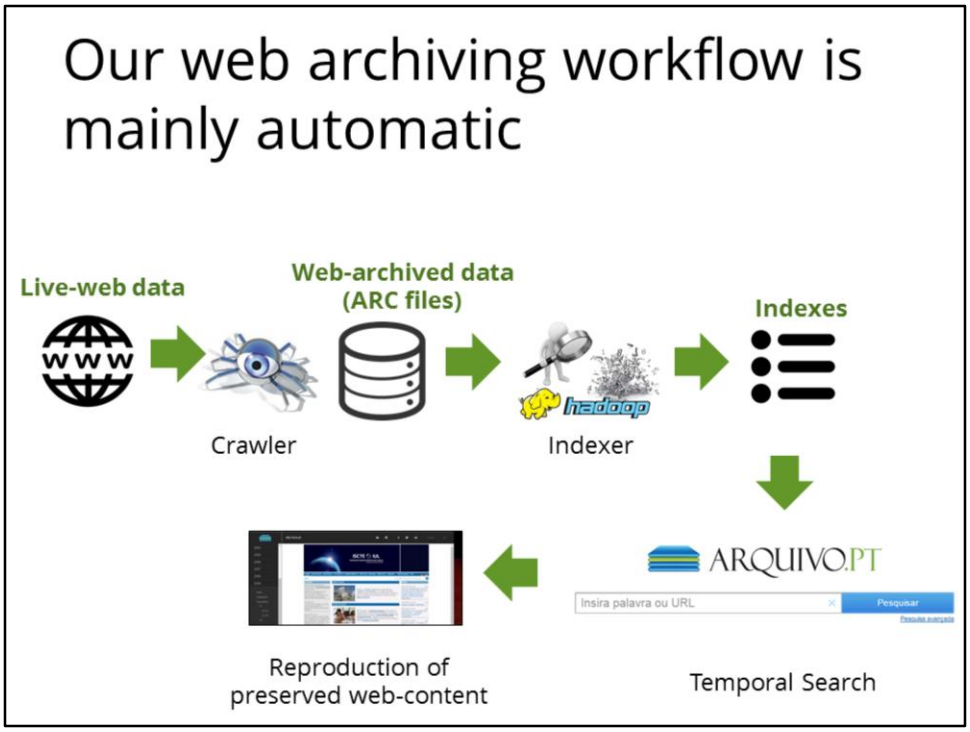
In 2007: the project was launched
In 2010: our first prototype, that enabled searching pages from the past, was made publicly available
In 9/2013: the service collapsed due to an hardware malfunction. We experienced data loss of 17% of our archived data correspondent to approximately 17 TB. There were interruptions in the acquisition of web content. And the search service was suspended
Between 2014 - 2016: We have been working on the recovery and improvement of the service, which is has been stable.

Now, we can share our experience.

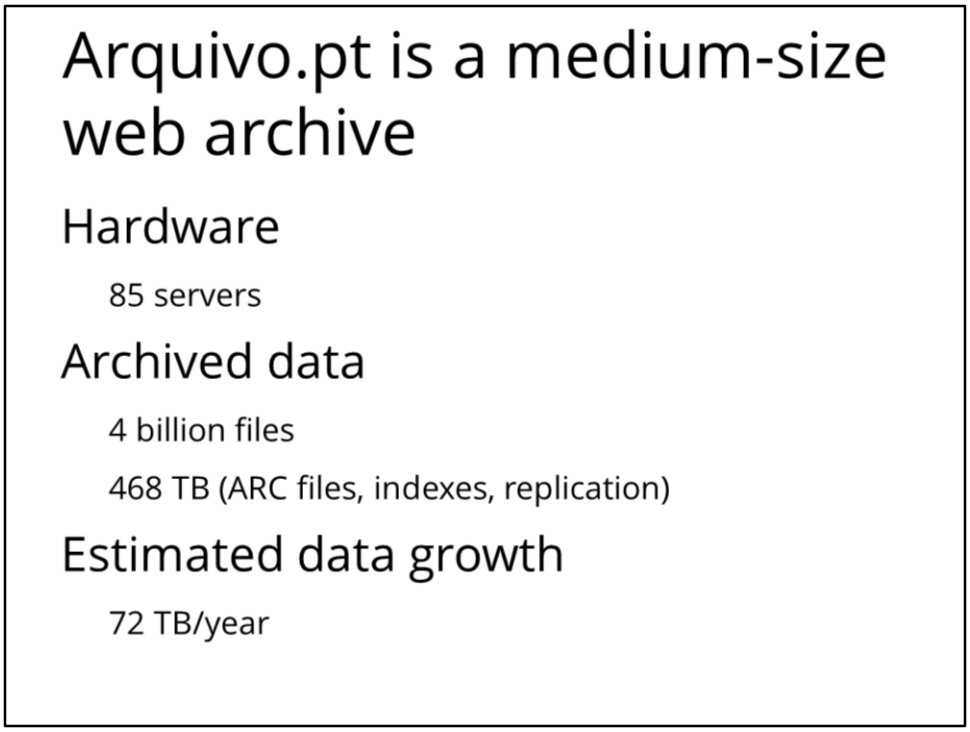
So, the objective of this presentation is to share our experience, so that other services can also learn from our mistakes, and adopted solutions.

Arquivo.pt system overview

To define the context of our work, I am going to briefly describe the system that supports the Arquivo.pt service.



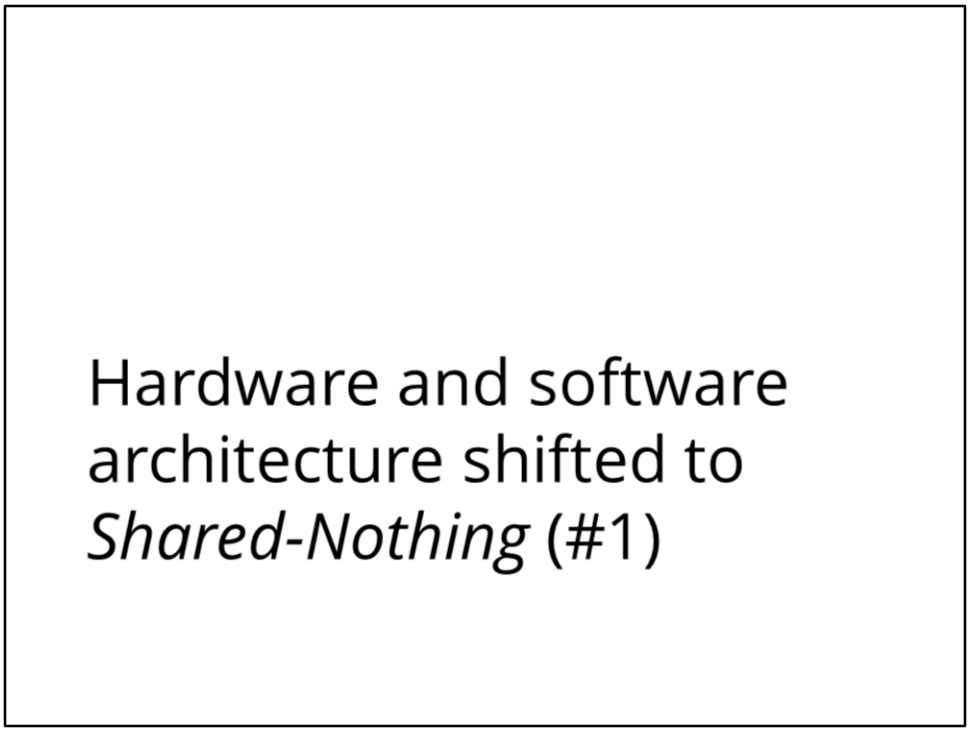
Our web archiving workflow is mainly automatic. In general, it works similarly to a live-web search engine.
Web content is crawled from the live-web and stored on our data center.
Then, it is processed using Hadoop to generate the indexes that will support the search service.
However, there are 2 main differences from a live-web search engine:
The first one, is that our search component must rank search results considering their temporal features.
The second one, is that our web archive must try to reproduce the preserved web-content as close as possible to its original format.



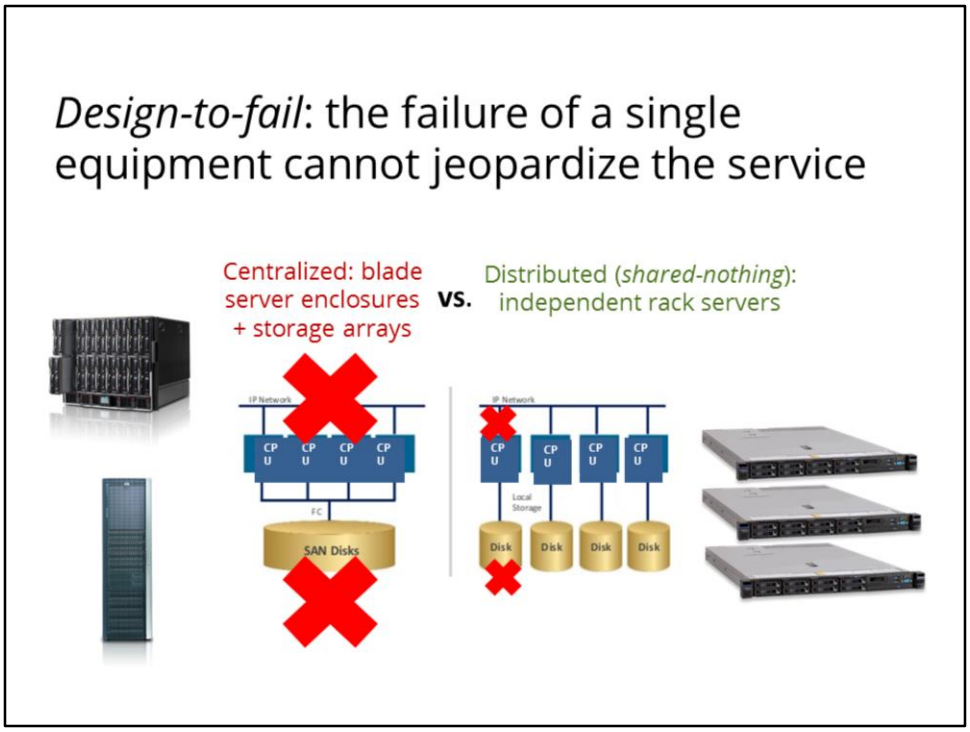
Arquivo.pt is a medium-size web archive.
Its system is hosted on 85 servers and holds 4 billion files, which require a total of 468 TB of storage space to be preserved. The estimated data growth is of 72 TB per year.

5 measures to improve the robustness of Arquivo.pt

So next, I will describe the 5 main measures we adopted to improve the robustness of our service.



The first measure was migrate the architecture of our hardware and software to a *Shared-Nothing* paradigm.



The objective was that the failure of a single equipment could never jeopardize the service availability.
The system architecture was redesigned to eliminate all single points of failure.
We abandoned the centralized hardware architecture based on blade server enclosures and storage arrays, and adopted a fully distributed architecture based on independent rack servers.

Inefficient physical space management at the data center with blade systems

Space that was never used

Space still occupied after servers disabled

Blades systems are compact.
Therefore, in theory they should lead to an efficient management of physical space at our data center.
However, in practice we witnessed the opposite.
Using blade server enclosures had been causing the waste of physical space at our data center.
The physical space occupied by the enclosures was taken, even if they did not have all its slots filled.

Then, the space occupied by the enclosures could not be released, even after some of its servers or disks were disabled.
So, physical space was occupied even when it was not being used.

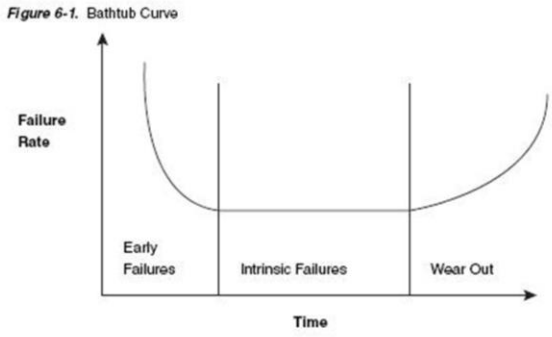
Independent rack servers

Only operational servers occupy physical space

Physical space is released as servers break

On its turn, the management of physical space using independent rack servers is simpler and more efficient because:
Only operational servers occupy physical space.
And this physical space is can be immediately released as servers break.

Perform load tests immediately after buying to induce failures



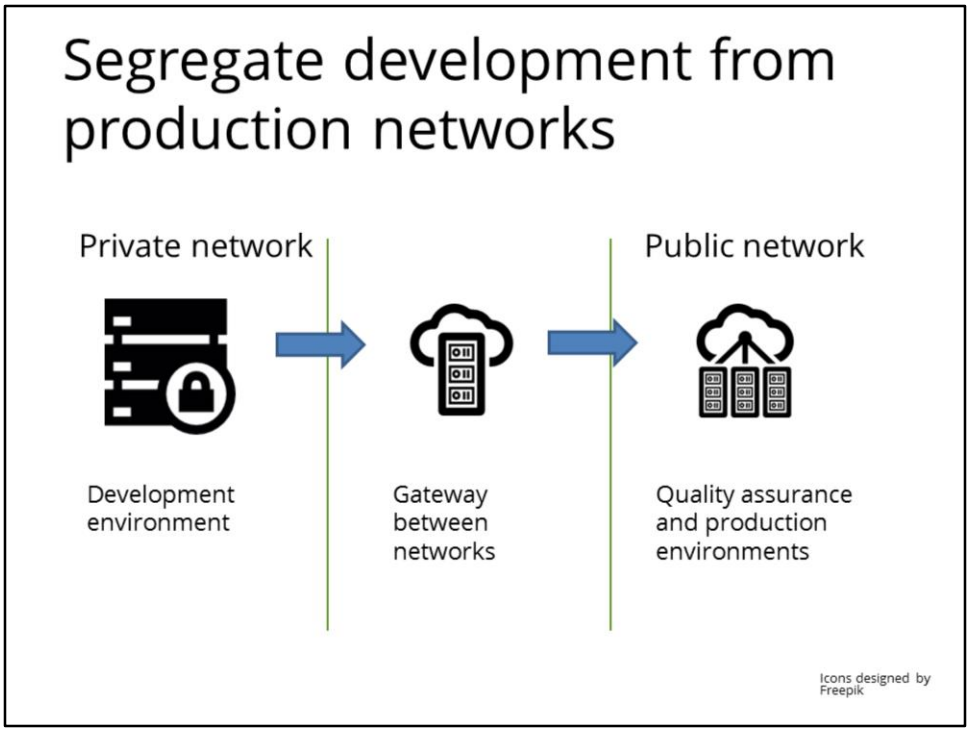
Open source tools: *bonnie (disk)*, *stress (CPU)*, *memtest (Memory)*

Bathtub curve: identify Early Failures during the warranty period

It is a well-known fact in Engineering that the failure rate along time follows a Bathtub curve.
That is, engineering components fail more often, when they are new, or old.
In Computer Engineering there is a common awareness that old components tend to fail.
However, we frequently forget that new components are also very prone to failures.

Therefore, we decided to perform load tests immediately after buying new hardware to induce failures by applying simulated workload using open source tools such as: *bonnie*, *stress* or *memtest*.

The objective is to identify faulty hardware during the warranty period and before deploying hardware into the production environment where failures could have impact on the service availability.



At the network level, we segregated development from production, so that these networks do not interfere with each other. For instance, if we cause a network overflow during a development experiment, this will not affect the normal functioning of the service in production.
All the development machines are connected in a private network.
The machines that belong to the quality assurance and production environments are accessible through the Internet.

Reinforced replication policies (#2)

The second measure, was to reinforce our replication policies .

Tape

Offline backup

Bundle backup to tape every 4 months

- ARC files, indexes

Random test recoveries from t

- Data recovery from tape is very slow

We started replicating data at several levels, using several distinct media types.

We use tapes to perform offline backups of data. We make bundle backups to tape every 4 months, including archived data and corresponding indexes.
And then later, we perform random test recoveries from tape.

Notice that this process is demanding because data recovery from tape is very slow, but it is the only way to assure that the backups are being properly performed and documented.

Hard disks

Online backups

Redundant server disks (RAID-5)

All data is replicated across 2 independent servers

- ARC files, indexes, software

Daily backup during crawl on live hard disks

- Lose at most 1 day of crawled data

We use hard disks to perform online backups of data.

The disks installed on each server are redundant (using RAID-5)

As a rule of thumb, all data must be replicated at least on 2 independent servers.

During the execution of web crawls, we perform daily backups of the crawled data on independent servers.
This way, in case a crawl server completely fails, the worse case scenario is that we lose 1 day of crawled data. This has never happened so far.

Distant location backups

Tapes moved to distant geographical location

Lisbon to Porto: 275 KM

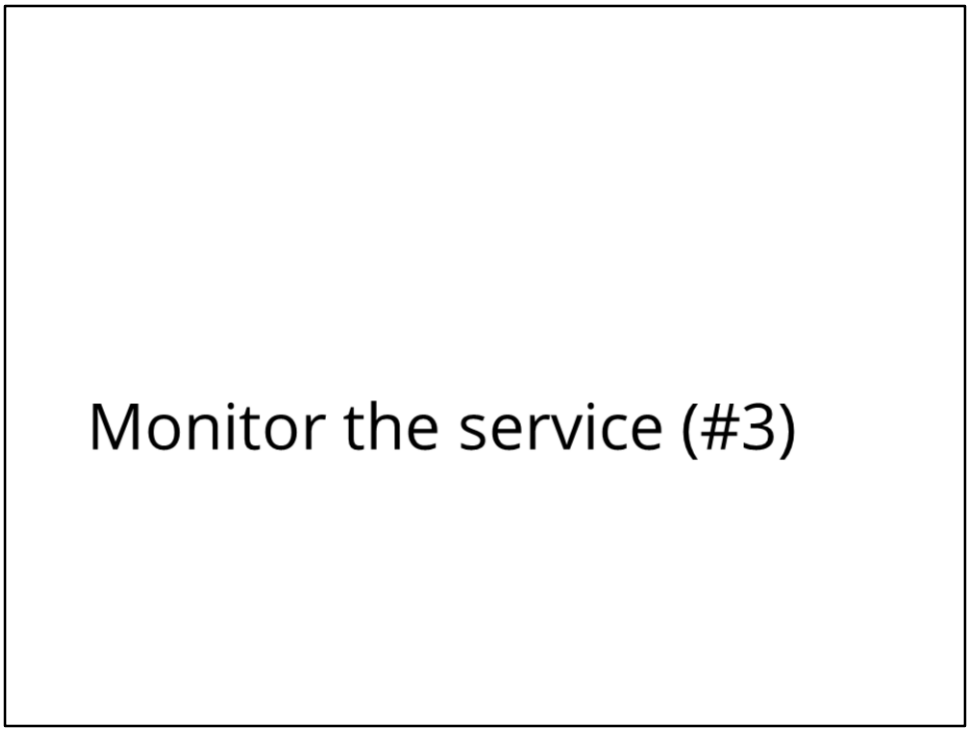
ARC files copied to the Internet Archive through the Internet

Lisbon to California: 9 000 KM

We perform backups to geographically distant locations by:

Moving the tapes with bundles of our data from Lisbon to Porto, which is 275 KM away from the origin data center.

And copying archived data to the Internet Archive through the Internet, which is 9 000 KM away from our location.



The third measure, was to improve service monitoring.

Monitoring tools fail

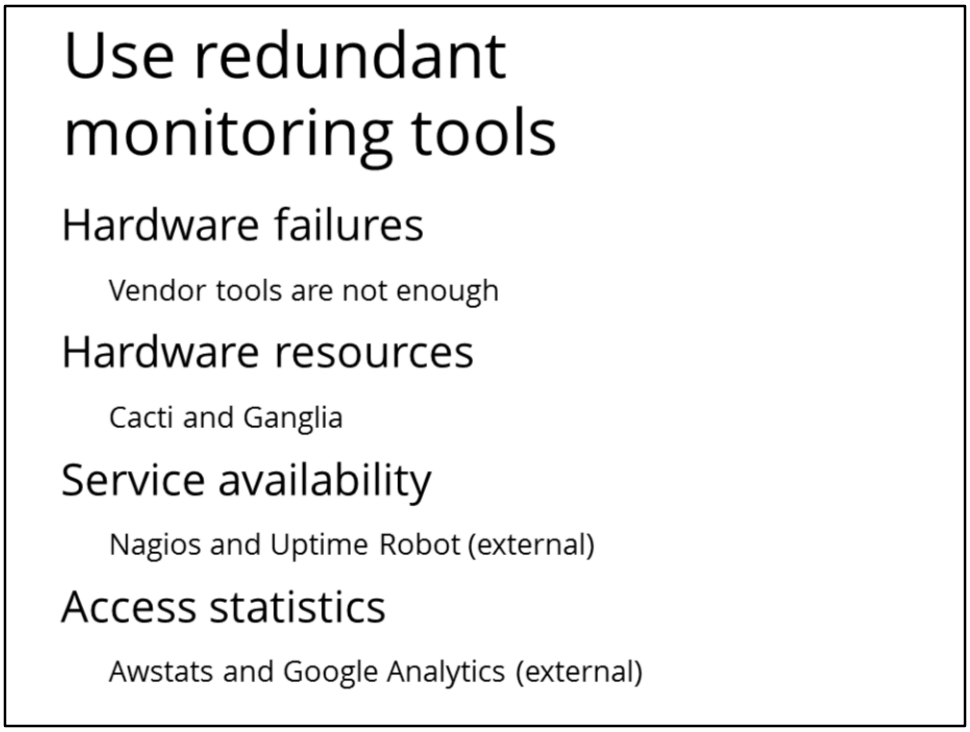
The service is broke
but we didn't know

So we did not fix it

Who monitors the monitoring tools?

Since the beginning of our project that we used monitoring tools to verify service availability.

However, if a monitoring tools fails, we would not be able to identify a service failure. So, the question that arose was: who is monitoring the monitoring tools?



We decided to apply redundancy on the monitoring tools, to detect failures or malfunctions, even when the monitoring tools fail.

The vendor tools are not enough to detect hardware failures or resource exhaustion. So, we adopted free open source platforms, such as Cacti and Ganglia, to monitor hardware resources.

The service availability is monitored using an internal Nagios platform but also using a free cloud service named Uptime Robot.

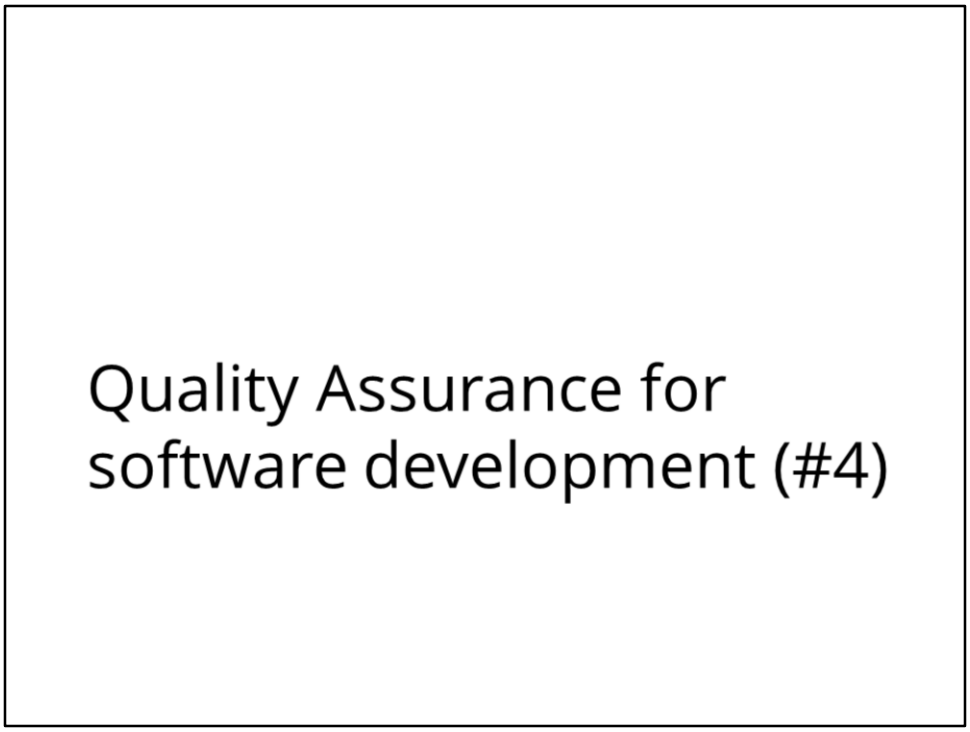
The access statistics are monitored using an internal awstats instance, but also Google Analytics.

Besides, different monitoring tools provide complementary perspectives that enable a more comprehensive analysis of the performance of our service.

Induce faults to test monitoring!

It's better to identify problems when you are ready for them

Moreover, we periodically induce faults on system components to test monitoring, and fail-over mechanisms.
It is always better to identify problems when you are ready for them.



The fourth measure, was to reactivate quality assurance for software development.

Regression:
"when you fix one bug, you introduce several newer bugs."

People get tired from doing repeatedly the same (testing).
Computers don't.

When we fix a software problem, it is common to introduce a new one. Therefore, something that was previously working fine, stops working. These events are so common in software development, that they have a proper name: regressions.

The main reason why regressions occur, is because software developers focus their attention on the solution of the new problem. And not on the validation of the solutions that they applied, to solve previous problems.

Basically, people get tired from doing repeatedly the same. That is, testing.
The good news is that **Computers don't.**

Code testing: automatize

Compilation: the code is well written!

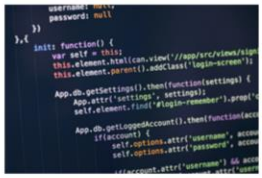
Unit: does what it supposes to do!

Functional: makes the service work

Simulate user workflows (e.g. search for an archived page)

Many free and powerful tools to automatize testing

SeleniumHQ, SauceLabs, Jenkins, SonarCube



Thus, we automatized code testing at several levels.

The first level of testing, is to periodically automatically compile all code to detect integration problems.

The unit tests, verify that the software components comply with their specifications.

The functional tests simulate end-user workflows, for example searching for an archived page.
There are many free and powerful tools to automatize testing such as SeleniumHQ, SauceLabs, Jenkins or SonarCube.

Workload capacity testing: automatize

Establish minimum thresholds for new service release

- Jmeter
- Workload average: 3 responses/second
- Speed average: 5 seconds per response

A service may stop working due to excessive workload.
The workload capacity must be periodically and systematically measured.

We simulate and measure workload capacity using Jmeter.
A new release is deployed to production only if it meets the minimum quality thresholds of 3 responses per second with response speed of 5 seconds.

Having these thresholds in mind, we can proactively respond to abnormal service workloads resultant from, dissemination activities or Denial of Service attacks.

Security testing: automatize

It's not "**if** we get attacked",
it's "**when** we get attacked"

OWASP Zed Attack Proxy (ZAP)

Expert reviews

Security concerns are a "must" for any online service.
Any computer starts suffering attacks quickly after being connected to the Internet.
It is not a matter of IF our service could suffer an attack but WHEN our service will suffer an attack.

We use the ZAP tool, from the Open Web Application Security Project to perform automatic security testing.

We are lucky to have security experts on our organization, that also help identifying potential security issues. However, the ZAP tool has been enough to identify the most critical security vulnerabilities.

Usability testing: conducted by skilled professionals

What is the **use** of a service that **users** cannot **use**?

Identify the problems that **really affect** the service

Most technical problems are reflected on usability obstacles

Help from Human Computer Interaction group from University of Lisbon and UX training

Even we provide a fully functional service, this does not guarantee that the users can use it effectively and efficiently. Communication or functional issues may arise that inhibit users to take advantage of the provided functionalities.

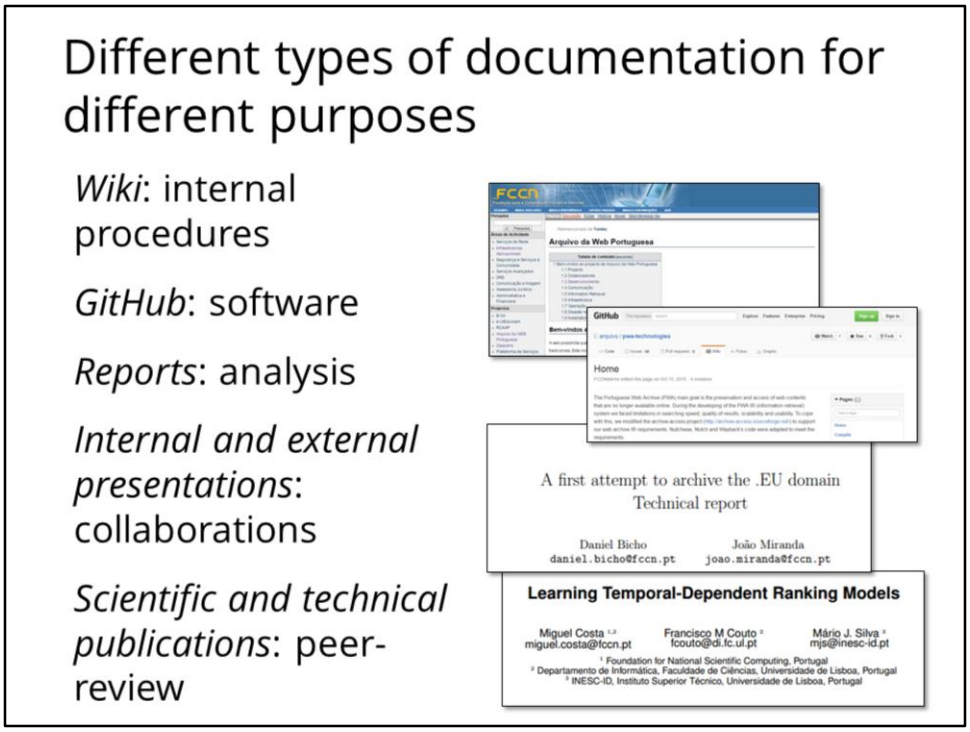
Systematic usability testing conducted by skilled User Experience professionals, enables the identification of the problems that **really affect** service quality. Notice, that most technical problems are reflected on usability obstacles. For instance, if hardware resources are scarce, this will be reflected on slowness perceived by the user.

It is really important that usability testing and results analysis is conducted by skilled User

Experience professionals. Otherwise, we may bias testing or misinterpret the obtained results.
We got help Human Computer Interaction group from the University of Lisbon and also invested on User Experience training to our team.

Document and test procedures (#5)

The fifth and last measure, that I would like to share is to document procedures but also to systematically test the generated documentation.



Generating documentation is obviously important to manage a service. But there are different types of documentation for different purposes.

We use a restricted access Wiki to document internal procedures.

An open-access GitHub repository to document software.

We write and publish technical reports about system analysis to get feedback.

We frequently make internal and external presentations to seed collaborations. Some of these presentations are recorded on video and published online.

We invest a significant amount of effort on publishing our scientific and technical achievements to get peer-review, that will contribute to define our strategies for the future.

Test the documentation

Installations of software components from scratch

Procedures executed by colleagues based on existing documentation without help

Generating documentation is important.
But who tests the generated documentation? Confusing or misleading documentation is of little-use.
So, we established a simple procedure to test documentation and assess its quality.

First, we establish an initiation ritual, in which a new member must make a full installation of the system from scratch, based on the existing documentation, with minimum help from colleagues.
During this process the newcomer must update the documentation every time an obstacle is detected.
On everyday work, every time a new important documentation is generated by a team member, the test procedure is performed by different colleague.

Open source
everything we do

github.com/arquivo

Increases responsibility

Increases software quality

Our software repository is hosted on Github.

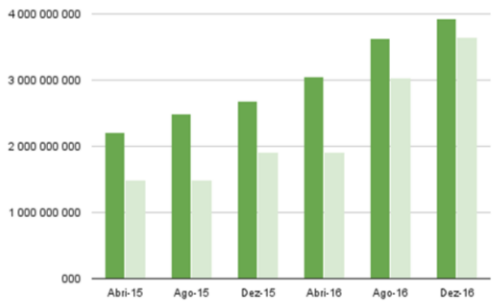
So, our software and documentation is born open-source.

This exposure increases the sense of responsibility on the developers and therefore, increases software quality.

Results

Now, I will present some results related to our service, that are indicative of the effectiveness of the applied measures.

Crawling and indexing are stable



The crawling and indexing of new content from the web has been stable for the past 2 years.

The temporal search service was available 100% of the time during 2016.

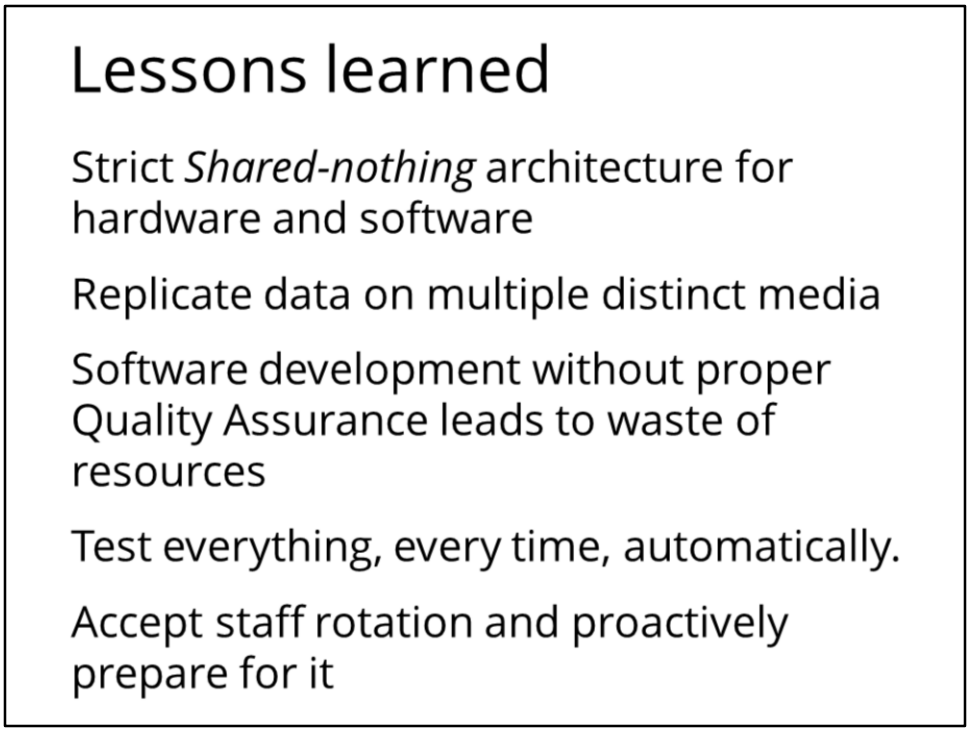
Recovering our users

4 090 users per month (average)

Gaining new users

- 90% are new users

We are recovering our users, and gaining news ones. Google Analytics registered an average of 4 090 users per month, from which 90% are new.



To summarize the main lessons learned were:

Follow strict *Shared-nothing* architectures for hardware and software design.

Replicate data on multiple distinct and independent media support.

Backup to tapes is still useful but recovery is very slow and prone to errors. Replication also on live hard-disks is a “must”.

Test everything, every time, including the testing tools and documentation. Whenever possible, automatize the testing procedures.

Despite the enthusiasm of developing new services, always keep in mind that Software development without proper Quality Assurance, leads to waste of resources, or even project failure.

Accept staff rotation and proactively prepare for it.

I hope that these lessons learned can be useful to you. I believe that they are applicable to any web-based information system.

Please feel free to email me. Any comments or suggestions are welcome.
Thank you for your attention and, I would like to invite you now, to search pages from the past using Arquivo.pt.
Cheers.