T-SoftwEng Project

Berger Antoine
MSE
HES-SO Geneva, Switzerland
Email: antoine.berger@master.hes-so.ch

Burri Maxime
MSE
HES-SO Geneva, Switzerland
Email: maxime.burri@master.hes-so.ch

Cicciù Salvatore
MSE
HES-SO Geneva, Switzerland
Email: cicciu.salvatore@master.hes-so.ch

Polla Michaël
MSE
HES-SO Geneva, Switzerland
Email: michael.polla@master.hes-so.ch

António Domingos Ana Sofia
MSE
HES-SO Geneva, Switzerland
Email: anasofia.antoniodomingos@master.hes-so.ch

***Abstract*—This document presents the assignment we did for the T-SoftwEng course for the MSE Master HES-SO. The purpose of this assignment was to choose an hypothesis in the field of software analytics and to create a project using some metrics method to see if the hypothesis is correct or not.**

***Keywords*-master mse; software engineering; software analytics; github; jira; metrics; hypothesis**

I. Introduction

The goal of this project is to find an hypothesis related to computer science, more precisely about software analytics or open affect. To find an hypothesis all the people of the team had to propose a subject for each theme. Then we decided which one we wanted to develop. Once a hypothesis was chosen, the aim was to prove that our hypothesis is verified or not. To do that, we built an application with technologies that we have seen in class such as Docker. This document is our report of our work.

Hypothesis

Once all the team members proposed a subject, we decided to keep this hypothesis : ***"Will the quality of the work be influenced at the approach of a deadline ?"***. This hypothesis was proposed because all of us already faced a deadline and it was always stressful. So we were wondering if our work is of the same quality when we are stressed as when we are calm. If this hypothesis is verified, new mechanisms can be created or used to avoid that problem, thus improving the work quality in these critical periods. For example, a code reviewing could be done by one of the team before any important release or commit.

To determine if our hypothesis is correct or not we decided to work by steps :

1) Find a project on both Github and Jira
2) Extract the commits and issues from Github
3) Extract the deadline/date of the release version from Jira
4) Build a pipeline Docker for each action
5) Use the data to create graphics
6) Analyze the graphics
7) Respond to the hypothesis

II. Project

First we had to find a project on both Github and Jira, so we have looked on those sources : jira.spring, jira.jboss and issues.jenkins but finding a projet on both Github and Jira with enough data that we could use to analyze was quite difficult.

Then we found one project on both Github and Jira with enough metrics that we could use for this assignment. The project we decided to use is the project spring AMQP [1]. We choose this project because it had a lot of commits, issues and released versions that we could use to create graphics to analyze and respond to the hypothesis. When the pipeline of the assignment will work we will search for more project to be able to compare the results and assumptions we had with the first project.

With the project found we were able to start the extraction of the metrics. We started with the metrics from GitHub : the commits and the issues, then the metrics from Jira : the release version with the name and the date of the release. To do so we created one script for the extraction of the metrics from GitHub and another one for the extraction metrics from Jira. Those scripts extract the metrics into a CSV file that will be used to create graphics.

When the scripts were working, we created one Docker per script and at the same time we started using the CSV files to create graphics. First, we tried to use the software proposed by our teacher : Tableau, but we weren't able to create graphics like we wanted so we finally decided to create a python script to create them. Indeed creating this script gave the possibility to create a pipeline of Docker containers that will extract the metrics and then generate the graphics automatically.

Finally, with the pipeline working we searched other

[1]Spring project on Github : https://github.com/spring-projects/spring-amqp and Spring project on Jira : https://jira.spring.io

projects to test our pipeline. We found 2 other projects with enough metrics to use, those projects are also about spring. The projects are spring framework and spring data redis [2]. With those 2 project we were able to test our pipeline and generate another sets of graphics that we will analyze in the section IV.

The figure 1 shows the architecture of the project implemented that correspond to the description of the project below.

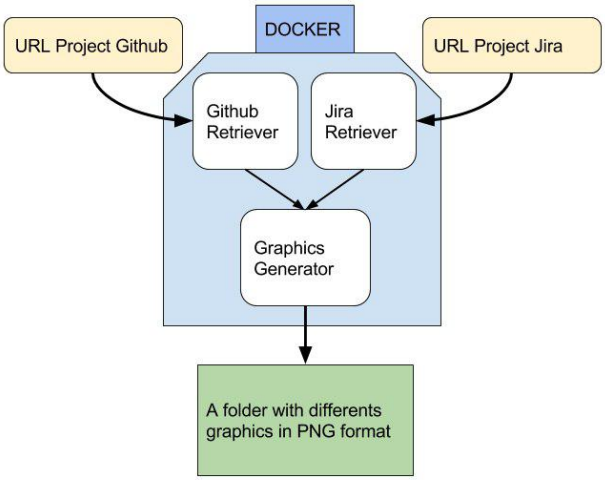


Fig. 1: Project architecture

III. Development

Tools

We chose to use Python for the scripting part of our project. Python is very useful for scripting and for making simple HTTP request. It's easy to use. We used the Python GitHub API and Python Jira API to get the metrics we want from the projects and also used the library Panda to create the CSV files containing the metrics extracted and that will be used to create the graphics. We also used Python Graph and the library Matplotlib to generate all our graphics in PNG format. All of the Python scripts were put into Docker containers that allow us to create a pipeline.

We also used the software Tableau but we didn't keep him because we prefer creating the graphics directly with Python as explained in the section II.

Implementation

For reminder, the Figure 1 shows the project architecture that will be described more specifically in this section.

Three Docker containers were created. Two are in charge of executing the Python scripts that extract the metrics desired from Github (commits) and Jira (issues and released versions). The scripts retrieve the projects URLs from environment variable created beforehand. That way, it is easier to change the values as if they were set directly in code.

The retrieved data are exported to CSV files that are shared with the others containers as shown in the Figure 2. Those files are going to be used in the third and last container. From the commits, issues and versions, it creates various graphical representations and exports them in PNG format.

It's important to know that each container as to wait for the previous one to end his execution to start his own. To work together efficiently, each container access the same shared volume, as shown in Figure 2.

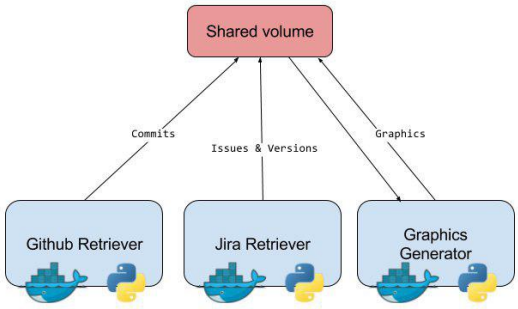


Fig. 2: Docker containers

Problems encountered

As Docker was new to us, we had a hard time figuring how to synchronize the containers, as Docker launches all of them at the same time. Apparently, Docker doesn't provide this kind of functionality ; we only found some example using the *healthcheck* mechanism, but it didn't totally suit our needs. The solution we implemented was some kind of *mutex*. Each retriever creates an empty file, that gets deleted when all the data has been collected. The graphics generator waits that no more "*mutex* file" exists before starting its work.

Another problem we faced was about reading and analyzing all the data. At first we wanted to use Tableau, a software that produces interactive data visualization. It seemed easy to use at first sight, but we never got the vizualizations we wanted, and none of us knew the software. The fact that it was a costly product was also a reason that decided us to switch to another solution. After some research, we finally chose to use matplotlib in a python script that was integrated into the pipeline.

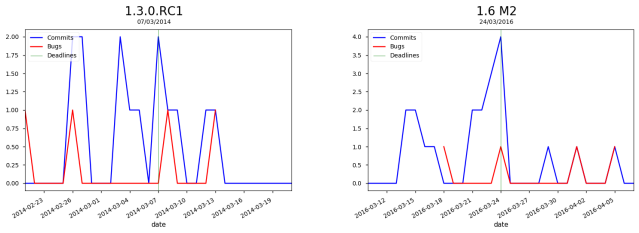
[2]Spring framework on Github : spring-projects/spring-framework, jira repo : SPR and Spring dataredis on Github : spring-projects/spring-data-redis, jira repo : DATAREDIS

IV. Observations

For each project we extracted 4 kinds of graphics : zooming 14 days before and after for each released version; all day; all week and all month for all the versions. Those graphics show the commits from GitHub, the issues and the released version with his date from Jira.
For each project, we will choose 2 graphics, from all the released versions, that are zooming 14 days before and after one version release that will be analyzed. The other graphics, all day, week and month for all the versions, won't be analyzed in this document. Indeed there are a lot of versions for each project and it is difficult to see clearly but all the graphics will be available joined with this document.

Spring project amqp

In this section we took the 2 chosen graphics in the figure 3 that show differents kind of activities and we will analyze them.



((a)) Version 1.3.0 RC1 ((b)) Version 1.6 M2

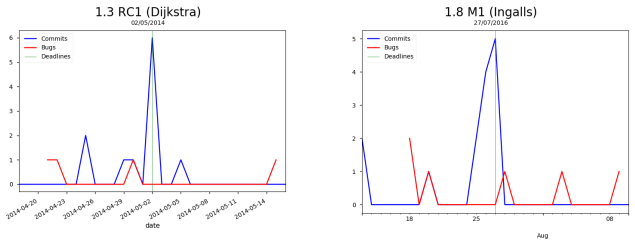
Fig. 3: Graphics of 2 released version for AMQP project

In the graphic 3(a) we can see for the issues that there is some before the release and then there is none until the release of the version. When the version is released the amount of issues climb. And for the commits we can see that during the sprint before the release they made commits regularly. There is no peak of commits just before the release, so we can assume that they weren't stressed, maybe because the work was constant. We also see that the commits and the issues are stopped at the end.

In the graphic 3(b) we can see for the issues that there is none until the commits are made. We clearly see that there is one little commit at the beginning of the sprint and one big the last days before the release. At the same time the issues are identified and then there are commits that we supposed correct the issues and so on. This could mean that when a new functionality is pushed someone is in charge to test the functionality and report any bug, this is an interesting way to work because the project is always tested and can be quickly corrected.

Spring project data redis

In this section we took the 2 chosen graphics in the figure 4 that show differents activities and we will analyze them.



((a)) Version 1.3 RC1 (Dijkstra) ((b)) Version 1.8 M1 (Ingalls)

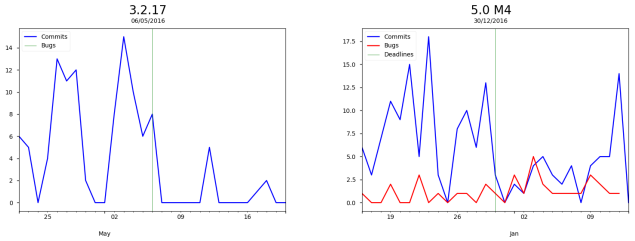
Fig. 4: Graphics of 2 released version for data redis project

In the graphic 4(a) we can see for the issues that there is almost any issue for this version but we can see that the commits occure almost all on release of the version. We can suppose that all the work was done at the end of the version and that the version wasn't tested after the release or that there were no issues to be reported.

In the graphic 4(b) we can see for the issues that there are issues reported during all the time-line but they are not a lot, just a few. And we clearly see that there is only one commit done before the release and that the issues are reported just after. With this informations, we can suppose that the developers worked a lot just before the release and that mistakes were maybe done.

Spring project framework

In this section we took the 2 chosen graphics in the figure 5 that show differents activities and we will analyze them.



((a)) Version 3.2.17 ((b)) Version 5.0 M4

Fig. 5: Graphics of 2 released version for framework project

In the graphic 5(a) we can see that there is no issues, that means that there weren't any test during the development, that isn't a good way to work. We also see that there are a lot of commits during the sprint. That means that the developers were doing the tasks and then commit them without testing report. We also see that the amount of commits drop just after the release : maybe it's because the next sprint is preparing, so the developpers are not working on new functionalities.

In the graphic 5(b) we can see that issues are reported during all the time-line. This could mean that the previous is tested and maybe that the new version is also tested

when there is updates. In this time-line there is also a lot of commits during the development but we can see that the amount increases at the end. We suppose that the increase is due to the correction of the issues. We also noticed that the amount of commits drop just before the release, this could mean that the developpers weren't stress and were careful when they commit.

V. Conclusion

After the observation of the graphics for each project we can consider that our hypothesis is verified. In fact, for almost all the graphics we saw that the amount of commits intent to climb just before the release of the version and when the tests were made the amount of issues also climb after the release of the version.
This hypothesis can change in function of the project and the team but in those projects we can say that the **approch of the release of a version can impact the quality of the commit** because of the commit done without testing before.

For the future we could apply this on one of our project and so we could also have the impression of the team directly after the releases. In fact those informations would be a bonus for the analyzes of the metrics of the project and this hypothesis.

This project was interesting, indeed we were able to see that we are not the only one that are stressed when a deadline is approach and that our work is sometimes affected.

Acknowledgment

The authors would like to thank their professor M.Liechti for supervising the project, and for all the advices he gave.

References

[1] **MAMAStress Github Project** : https://github.com/MichaelPolla/mamastress

[2] **Docker Documentation Tutorial** : https://docs.docker.com/get-started/

[3] **Jira Python API** : https://jira.readthedocs.io/en/master/

[4] **Graphical Python API matplotlib** : https://matplotlib.org/

[5] **AMQP - GitHub** : https://github.com/spring-projects/spring-amqp

[6] **AMQP - Jira** : https://jira.spring.io/browse/AMQP/

[7] **DataRedis - GitHub** : https://github.com/spring-projects/spring-data-redis

[8] **DataRedis - Jira** : https://jira.spring.io/browse/DATAREDIS/

[9] **Spring framework - GitHub** : https://github.com/spring-projects/spring-framework

[10] **Spring framework - Jira** : https://jira.spring.io/browse/SPR/