Visual analysis of interactions between developers

Favre Lucas, Pasquier Gerome, Pfeiffer Ludovic and van Dooren Nicolas

***Abstract*— This paper is made from a school project realized in the Software Engineering class at the HES-SO Master.**

I. INTRODUCTION

For many years, Git and GitHub have been really powerful tools for software programming and developments. Thanks to GitHub, anyone can access thousands of open-source projects with a lot of informations about them. These informations can be coding methods, like how do you use some methods or commits, why a modification was needed... The information which interest us is the commits, precisely who did them and on what files. With this information, we can implement a way to know whom talked to whom. This leads to know every interactions between developers in a project.

II. INTERACTIONS

A. Definition

Interactions between developers can be difficult to evaluate. It is interesting to know on which parts people cooperate in a project. If a file has been modified by many people in short time, a deduction can be made and you can be sure that these people shared informations. Which means that an interaction between them happened. You can deduce that because developers don't want to reverse engineer every piece of code when they want to work on a specific file.

B. Utility

Knowing the interactions between developers can be useful in a lot of ways.

- Organization of the office, knowing which persons interact the most.
- For big enterprises, you can know which offices work on what easily.
- If two people work on the same thing together, you want a good relationship between so they will work better. If a conflict happens between them, really bad consequences can happen to the project.
- You can identify the central person, the one who work on a lot a files. Everyone can depend on that person.
- The importance of a developer, if someone is too important for the project, if this person leaves, your project will be in a really bad spot.
- You can see connections between different services (design, developers, testers, ...), if these people use the phone or so to communicate, maybe you should put them on the same floor or so.
- You can see if an agile method like scrum is respected. If so, the code should be shared between the developers.
- After the end of a project, you can see what went wrong. You can see if some people wasted time to know what the code do instead of just asking (You have to ask them if they talked of not).

There probably is a lot more interesting utilities to think about, but those are the main ones we found interesting.

C. Why the interactions

We wanted to know the interactions between developers during a project because we didn't really know how it is done in a professional level. We are students so when we work on a project together, it really is not the same as a developer for a society. We wanted to kwow what will happen for us when our scholarship is done. This is the main reason.

Another reason is that, it is interesting to compare how it changes between companies or independents project. For companies, you can see a good number of people working on the project, somewhat equally. But for independents project, there is one guy whom work on the entirety of the project with some people trying to help here and there.

III. IMPLEMENTATION

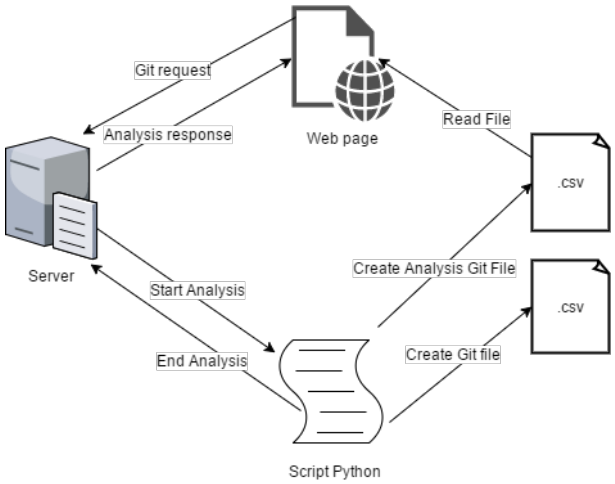
A. Technology choices

We made the choice to do this project essentially with the GitHub API. We chose GitHub because it is the biggest project database existing. Thanks to this, we got access to thousands of projects of any size. We made the assumption that if someone made a modification of a file, then another person made a modification in a small time lapse, an interaction has been made between them. To do that, we just checked the commits.

All of the backend commits retrieval and analysis has been done using Python. Python is a powerful scripting language and it is known by all 4 of us.

Some of us had already some nodejs with d3 experience so we made the choice to use it.

B. Architecture



For this project, we chose to have an architecture in four separate parts. On a hardware level, we just have a server and a client. The server is the one doing all the work. When a client connect to the server with his browser, he puts the last part of a Git repository url (e.g `https://github.com/Hoopaa/accord_project` - ¿ Hoopa/accord_project) and tells the interval he is interested in (in days, 1 to 10), it is the equivalent of the "Git request" arrow. When the server has these informations, it starts the analysis with a Python script. The script do some calls to the GitHub API to get all the informations about the repository, mainly the commits and the files. The script make 2 csv files, one containing all the commits for the project, and the other containing every relation between the commits. The second one uses the first one to make the relation. Those relations are mainly used to make the graphic that will be displayed on the client screen.

C. Chord diagram

The visualization choice for this type of work is really important. We spent some good time to search for a graphic that will be the best fit for our data and the result. For example, a histogram was not conclusive, we could not really see the interactions between developers, only the number of interactions they had.

The Chord Diagram is perfect for our work because it allows us to see the relations between every entity quite easily. All the diagram needs is an interaction matrix. Then, the possibility to put colors by sectors (a sector is a developer for us) is the best that we could hope. It allows to really show the "central" developers of a project.

Thanks to this solution, the user of our project can really easily make a conclusion about the developers that the project is really dependent for example. The conclusion will always be simple and intuitive.

D. Client/Server implementation

Our tool need different elements to be functional. A NodeJS server which is in charge of the web page and the Python scripts. Internet access is required, obviously, mainly because our tool need to retrieve remote data from a distant API (GitHub API). You can put our tool on the web if you have a server, you just need a key to access the GitHub API without issues.

1) Server: The main point of the application, this is why we didn't neglect the technologies to use and it functioning. The NodeJS server use the TCP (Transmission Control Protocol 8080). This port is an alternative to the 80 port, it is regularly used to deploy some web applications prototypes.

To realize a server, the usage of the Connect framework (`https://github.com/senchalabs/connect`) allowed the implementation to be fast and easy. The Serve-Static framework (`https://github.com/expressjs/serve-static`) allow to remove files from the root of a given repertory. Since we use files in our web applications, it is important to have a good implementation of those structures before starting the realization.

The communication between the server and the web page is realized with socket.io (`https://github.com/socketio/socket.io`). Socket.io offers a simple and precise communication between a page and a server. Given that the communication will be an important point in the web application, it is something to be taken in consideration.

The server has to be able to launch python scripts when asked. Given that the scripts ask for differents entry parameters, those scripts can't be launched at the start of the program easily, they have to wait for the client to make the request. To do so, the using of Python-Shell (`https://github.com/extrabacon/python-shell`) offers the possibility to easily launch python scripts from the server with the good parameters.

2) Client: The purpose of the project is to offer to the client a way to realized a complex analysis, the easiest way possible. To do so, the user interface should not be neglected and need to allow the client to find his way really easily and to be the most intuitive possible.

For this project, we made the choice to use really common elements like ListBoxes, CheckBoxes, EditBoxes.... Thanks to them, the user is not lost when he first come on our site. On the arrival to the site, the user's cursor is directly in the main element of the application, which is the repository field where the GitHub link has to be pasted (or written if he knows it). Furthermore, the web application offers a repository example. When the repository is good, all the client needs to do is to choose from the different possibilities at his disposal and finish it by clicking the validate button.

The using of JQuery (`https://jquery.com/`) allowed us to offer to the client an enjoyable moment on the platform. Mainly by allowing us to use an animation when all the work is in progress in the server side.

When all the work is finished and to offer the client an easy way to analyze the result, we used the D3 library (`https://d3js.org/`). This library allow to display data but the graphical way. D3 is really popular and powerful, this is why we made the choice to use this library. We used mainly D3 for the Chord Diagram it offers, this diagram allowed us to make link the users which made commits on the same files.

E. Python Script

1) Commit retrieval: Now that we described the server, we can talk about the main piece of the program, the Python script. There are 2 scripts. The first one is for the commit retrieval. For this purpose, we used the GitHub API. The API allows a limited number of requests, we had to make a token to identify ourselves. Even with the identification, we are limited, so we have to check for every request, the number of API requests that we have left.

Commits number are generally really high, the API can't return everything in one request. The API use a page system, so we need to retrieve the results page per page, meaning that the number of requests made by the API can be a problem for really huge projects. We didn't meet this problem but it means that our project is not scalable.

When all the commits are retrieved, we need to make another request for each one of them, so we can have every detail of every commits, such as files modified and so on. All of these are returned by the API in a JSON and contain too much information so we don't store everything. Every commit and its details are stored is a .csv file (this file can be downloaded by the client at the end). We store the following informations :

- "idCommit" The commit ID, we need it because if a commit is on multiple files, it will appear many times in the list. This commit is local and not from the API, it is only for linking the files to a commit.
- "name" The user name of the person who made the commit.
- "date" Date and time of the commit
- "message" When a commit is done, a developer need to put a message, it is this message.
- "nfiles" The name of the modified file
- "changes" Number of changes made on this file.

A first filter is applied when we write the file, this filter removes all the commits done by the user "GitHub" because these commits represents no one so they are not relevant for this project.

For this part to work, all the script needs is the part of GitHub url of the repository to analyze. This url part looks like this : username/projectname. The API url used : `https://api.github.com/repos/username/projectname/commits`, this one allows the commit retrieval. For the detailed commits : `https://api.github.com/repos/username/projectname/commits/sha`, sha is the commit ID.

2) Commits analysis: When all of this is done, the csv file can be produced and analyzed by another Python script. This Python script also produces a .csv file.

This script will search for commits with two different person using the same files in a certain time frame. This time frame is determined by the user of our project and will be passed as a parameter to the script. This time frame is only in days.

To do so, the script starts by retrieving all the user names existing in the .csv file. Then, an interaction matrix is created. This matrix will allow us to count the interactions between the persons. It is a two entry array, one for the user names for the rows and column names. The intersection of them is a number, this number is the number of interactions between those 2 persons. So when the script find two people working on the same file in a given time frame, it will increment the value of this number.

Example of an interaction matrix :

X	User 1	User 2	User 3
User 1	X	26	67
User 2	26	X	3
User 3	67	3	X

At the end, the interaction matrix is saved in a .csv file. The formatting is the following : User1, User2, Number of interactions, color

- User1 and User2 are the two names of the persons having interactions
- Number of interactions represents the number of interactions between those two.
- Color is the color that will be used in the Chord Diagram

F. Docker

When all of the project was done, we chose to make a pipeline of all of this. The idea was the following : every part that will be called need to be in its own Docker infrastructure. The pipeline would've looked like this :

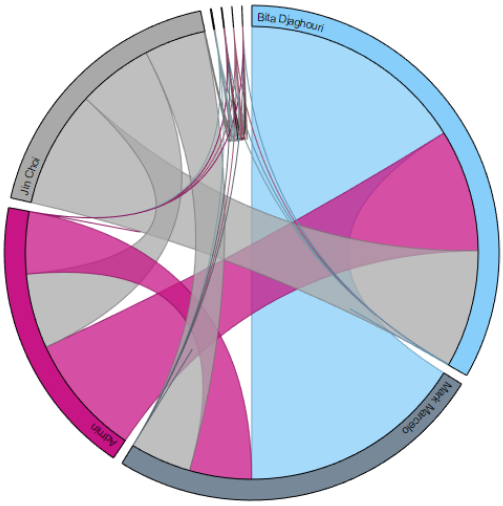
- The nodeJS server
- The first Python script (commits retrieval)
- The second Python script (commits analysis)

The logic behind it is that when a user click on the validate button, the NodeJS server starts the first python script container. When this Python script is finished, it starts the other Python Script container and end. The second Python Script do its job and end.

Unfortunately, we couldn't make it to work so we just made a simple Docker container containing all of our application.

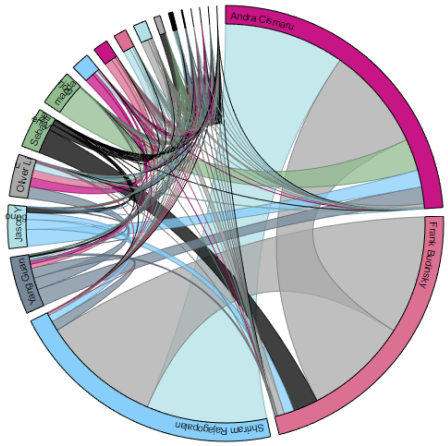
IV. RESULTS INTERPRETATION

When you run our application, after a little while, a Chord Diagram is put in the webpage. For the "reactide" GitHub (`https://github.com/reactide/reactide`), the diagram looks like this the following figure.



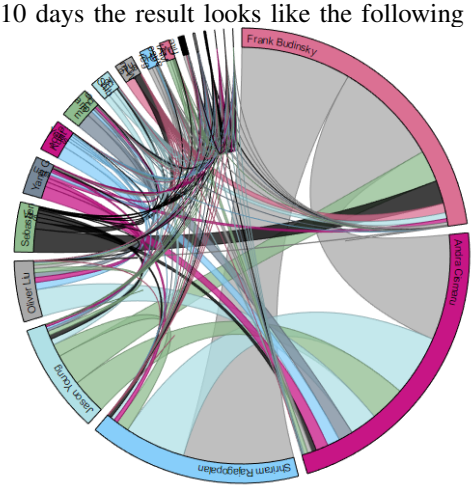
This interactions diagram tells us some really interesting things. We can start by saying that 4 people really work on ReactIDE. These 4 persons are working in the same office because you can see a lot of huge interactions between them. Some people tried to help but they are remote persons because you have less than 10 interactions for them. There is really high chance that they are people who made pull requests to correct some bugs. Another possibility is that they are working on really specific part of the project and only them are touching those files so they are not any interaction. For the main developers, you can see in a visual way the interaction them, we can see that there are really big interactions.

We need to take a lot to another project to really see some differences between a project with a little number of developers and one project bigger.



This diagram represents the ”istio” GitHub (`https://github.com/istio/istio`). This project is full of interactions, you can see that there is 3 ”central” developers who touch to pretty much everything in this project. Then you can see a good amount of ”small” developers who work with the 3 mains developers. The green one on the project, works exclusively with the pink/red one on the top right. When you see this one, you can see that this project is a big one because there is a lot of people working daily on it.

Those 2 graphs were with 1 day between the commits, if you put 10 days the result looks like the following :



When you look at the 10 days graphic, you can see some new interactions. Some developers become bigger ones by reducing the main ones interaction numbers. Before, you had 3 main developers and now you can really see 4 of them and the new one works a lot with the other 3. This allows us to see directly the evolution of the work within the project, and for example to show the communication of the developers over the days.

Our tool was designed to see the interactions between the main developers of each project. We can see that the visibility for developers who contribute less to the project is reduced. The tool aims to show directly the main actors of the project. This would allow, for example, a project manager to see directly the main actors contributing to the success of the project.

V. CONCLUSIONS

Even if we could not make the pipeline work, we think that our work is a good fit for the subject. You can really see the interactions and make your own conclusions about a project. All you need is to have the folder containing the docker configuration and follow the instructions contained in a pdf named ”Docker Container Utilisation”.

The scalability of this project is really not good so we advice to not deploy it to a real server, except if you can find a way to upgrade the number of requests to the GitHub API, then you can.

This project offers an interaction visualization of a GitHub project in a very explicit way. Furthermore, we decided to allow the user to remove some of the developers in the project. For example, if the user wishes to analyze interactions between only some main developers or to look the smallest ones, those who made only few functionalities.

The fact that the user can download the 2 csv files allows him to make more deep analysis about a project.