INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




GIT Profiling

DIOGO FILIPE RIBEIRO SILVA
Outubro de 2022

POLITÉCNICO
DO PORTO

GIT Profiling

Diogo Silva

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering

Supervisor: Nuno Bettencourt, PhD

Evaluation Committee:
President:

Members:

Porto, October 15, 2022

Dedicatory

To my supervisor for having the patience and willingness to help and direct me through the writing of this dissertation.

To my girlfriend who never let me give up and always believed that I would be able to conclude this document and the all the others. Sorry!

To my parents who provided for me and made the not mandatory sacrifices that allowed me to study and get a degree.

Abstract

This dissertation was written with the objective of creating and objectively defining software developers profiles. In order to support the proposed profiles, data was extracted and transformed from GIT repositories by an automated process. This automation was achieved by having an application that runs a combination of commands from GIT and Git Quick Stats and allows the consumption of the transformed output of these commands via an Application Programming Interface (API).

A client application was also developed that would aid in the validation of the profiles and would provide a dashboard like User Interface (UI).

The client application is able to query a Representational State Transfer (REST) service endpoint to get the available information from the serve and run a keyword match algorithm counts the hits on a certain profile. The keywords serve as a dictionary of terms that can be found in commit messages, file names or comments.

The final dashboard is able to represent the repository and profiles information while also, providing a way to compare multiple repositories. The profiles are also presented as trends since the developer has more than just one type of contribution.

In an effort to increase the confidence of the results that were the outcome of this automated process, manual checks were made to ensure that the right conclusions were reached regarding the profiles definitions.

The design and architecture of the applications developed follows a traditional client and server approach which allowed for the separation of the responsibilities as was described above. In order to validate that the application was behaving correctly, metrics regarding the execution times and memory consumption were collected.

Limitations with the developed work were also described since there are external systems that are usually used in conjunction with GIT repositories that contain information that could be used to increase the accuracy of the profiles. On a more technical level, some improvements to the overall architecture were also suggested that could enhance the final experience.

Finally, some future work was also theorised that included the seniority or expansion of profiles by integrating external systems that contain more information.

Keywords: GIT, Developer Profiling, Commit, UI, API, REST

Resumo

Esta dissertação foi escrita com o objectivo de criar e definir objectivamente perfis de desenvolvedores de software. Com a intenção de suportar as afirmações sobre os perfis propostos, foram recolhidos dados de repositórios GIT.

Para o efeito, um algoritmo que permite extrair e categorizar informação de forma automatizada foi desenvolvido. Esta automatização foi conseguida com a criação de uma aplicação que consegue executar uma combinação de comandos GIT e uma biblioteca externa com o nome de Git Quick Stats. Esta biblioteca permite que seja obtida informação legível para leitura humana de forma eficiente e rápida.

A aplicação corre num determinado número de repositórios de diferentes tamanhos e, trata de todas as operações pesadas que envolvem a extracção e transformação dos dados. Os dados transformados são disponibilizados para o cliente através de uma API

Para o efeito de visualização e categorização dos dados extraídos de um determinado repositório, uma aplicação cliente foi desenvolvida com o intuito de ajudar na validação dos perfis e na visualização dos mesmos num dashboard simples.

Esta aplicação cliente consegue fazer pedidos a um serviço do tipo REST com o objectivo de obter os dados transformados. Após a obtenção destes dados, a aplicação executa um algoritmo baseado em verificação de palavras-chave e guarda o número de vezes que uma palavra de um determinado perfil é encontrada.

As palavras-chaves servem como um dicionário de termos que podem ser encontrados em mensagens de commit, nomes de ficheiros e comentários que, é expectável que, existam quando um perfil se encontra a ser avaliado. O número de perfis detectados e a sua precisão estão relacionados com o volume e qualidade deste dicionário de termos. Por consequência, este dicionário foi modificado ao longo do tempo de escrita desta dissertação.

O dashboard final, que foi desenvolvido para esta dissertação, suporta a visualização dos dados de repositórios de forma legível, representação de perfis e comparação de vários repositórios. A informação presente nas métricas GIT como o número de commits, número de colaboradores e o número de ficheiros é toda passível de ser visualizada. Em conjunto com esta informação, a possibilidade de verificar a distribuição de todos os perfis no repositório é assegurada por um gráfico do estilo radar. É também possível verificar estes perfis, por desenvolvedor, que contém os perfis que lhe foram atribuídos. Um desenvolvedor pode ter mais que um perfil visto que as suas contribuições podem ser em vários tipos diferentes e, por consequência, foi necessário criar uma divisão de perfis principais e

secundários. O perfil principal é identificado quando se e comparar quais dos perfis atribuídos tem a maior pontuação e os restantes são considerados secundários.

Para aumentar a confiança dos resultados obtidos pelo processos automáticos, foram feitas verificações manuais que garantem que as conclusões obtidas sobre os perfis são validas. A proposta final destes perfis foi criada tendo em conta também os papéis mais comuns dentro de projectos e equipas de software.

O desenho e arquitectura das aplicações segue um modelo tradicional de cliente e servidor que permite a separação clara das responsabilidades descritas acima. Com o propósito de validar o comportamento destas aplicações, foram recopiados dados e métricas sobre os tempos de execução e consumos de memória. Isto foi feito devido aos objectivos da dissertação serem a definição dos perfis e o código relacionado com a obtenção dos mesmos.

As limitações do trabalho desenvolvido estão descritas dado que existem sistemas externos que, normalmente, são usados em conjunto com repositórios GIT. Estes sistemas contem mais informação que pode ser usada para aumentar o grau de precisão dos perfis encontrados ou até auxiliar na criação de novos perfis. Foram também sugeridas alterações e melhorias as aplicações desenvolvidas.

Por fim, foi também abordado o trabalho futuro sobre este tema. A expansão dos conceitos presentes nesta dissertação pode evoluir no sentido de relacionar os graus de experiência de um desenvolvedor ou na integração dos sistemas externos para aumentar a quantidade de dados disponíveis.

Contents

List of Figures **xi**

List of Tables **xiii**

List of Source Code **xv**

List of Acronyms **xvii**

1 Introduction **1**
1.1 Problem . . . 1
1.2 Goals . . . 1
1.3 Research Questions . . . 2
1.4 Research Methodology . . . 2
1.5 Hypothesis . . . 4
1.6 Thesis Structure . . . 5

2 State Of The Art **7**
2.1 Value Analysis . . . 7
2.1.1 New Concept Development Model . . . 8
2.1.2 Value Network . . . 10
2.1.3 AHP Model . . . 10
2.2 Repository Analysis . . . 10
2.2.1 GitHub API . . . 10
2.2.2 Microsoft GHCrawler . . . 11
2.2.3 Apache Kibble . . . 11
2.2.4 CHAOSS . . . 12
2.2.5 GitCompare . . . 12
2.2.6 Git Inspector . . . 12
2.2.7 Git Quick Stats . . . 13
2.3 Issue trackers . . . 13
2.3.1 JIRA . . . 13
2.4 Front end Technologies . . . 13
2.4.1 React . . . 14
2.4.2 Angular . . . 14
2.5 Back end Technologies . . . 14
2.5.1 Node.Js . . . 14
2.6 Supply Chain Management . . . 15
2.6.1 Git . . . 15
2.6.2 SVN . . . 15

2.7 Summary 15

3 Analysis **17**
3.1 Business Value 17
3.1.1 Five Elements of a innovative activity 17
3.1.2 External Factors 18
3.1.3 Value 19
3.1.4 Perceived Value 19
3.1.5 Business Perspective 19
3.1.6 User Perspective 19
3.1.7 Business Canvas 20
3.2 Requirements Engineering 21
3.2.1 Functional Requirements 21
3.2.2 Non-functional Requirements 22
3.3 Summary 22

4 Design **23**
4.1 Architecture 23
4.2 Application Architecture 24
4.2.1 Web Application 24
4.2.2 Progressive Web App 24
4.3 Profile Analysis Algorithm 26
4.4 Profiles 27
4.5 Requirements 28
4.6 Summary 29

5 Implementation **31**
5.1 Technological Stack 31
5.2 Implementation details 32
5.3 Summary 33

6 Experiments **35**
6.1 Indicators 35
6.2 Assessment 35
6.3 Results 36

7 Conclusion **39**
7.1 Research Questions 39
7.2 Contributions 40
7.3 Limitations 40
7.4 Future Work 40
7.5 Personal Remarks 41

Bibliography **43**

List of Figures

1.1 Design Science Research (DSR) framework (Brocke, Hevner, and Maedche 2020) 3

2.1 Innovation Process (Koen et al. 2002) 7
2.2 New Concept Development (NCD) (Koen et al. 2002) 8

3.1 Canvas Business Model 20
3.2 Functional Requirements 21

4.1 Overall Deployment Diagram 23
4.2 Web Application Architecture 24
4.3 PWA Architeture with front end processing 25
4.4 PWA Architeture with back end processing 25
4.5 Profile Analysis Sequence Diagram 26
4.6 Data Extraction From Repository 28
4.7 Client application sequence diagram 29

5.1 Main dashboard 32

List of Tables

2.1 Repository tools comparison 16

6.1 Repository data extraction metrics 36
6.2 Profile match metrics 37
6.3 Profile manual checks metrics 38

List of Source Code

4.1 Profile Analysis Algorithm 27
5.1 Repository Commands 33

List of Acronyms

AHP	Analytic Hierarchy Process.
API	Application Programming Interface.
CHAOSS	Community Health Analytics Open Source Software.
CSV	Comma-separated values.
DSR	Design Science Research.
FFE	Fuzzy Front End.
ISEP	Instituto Superior de Engenharia do Porto.
NCD	New Concept Development.
NPD	New Product Development.
PEST	Political, economic, social, and technological.
PWA	Progressive Web App.
REST	Representational State Transfer.
SCM	Supply chain management.
SPA	Single Page Applications.
UI	User Interface.

Chapter 1

Introduction

This dissertation is outcome of the masters degree in Software Engineering at Instituto Superior de Engenharia do Porto (ISEP) with the title Developer Metrics and Profiling. The main themes of the document are related with collecting metrics from software repositories and using these metrics to identify and define developer profiles and comparisons between different projects.

In the first chapter of the dissertation the context and the problem addressed are presented so the reader becomes acquainted with the presented themes. The overall purpose of this dissertation, its goals, research questions and the hypotheses to prove are also present here. Finally, the structure of the document will be presented.

1.1 Problem

Most software development companies use some kind of versioning system to keep track of its codebase and developer's contributions.

Although some platforms allow for code inspection and analysis of certain metrics, there is not a unified platform that allows for more detailed approaches and categorization of the returned data. This includes not only the metrics that can be extracted from commit logs but also from software diagrams and overall comments that exist in the repository. However, properly categorizing for each kind of profile is a challenge because these profiles must first be identified and their definition must be based in metrics that can be collected inside a repository.

The data that leads to these profiles can also be used to create comparisons between different projects and help identify issues, success rate of team compositions and even check if document designs are aligned with current application architecture.

1.2 Goals

The main goal behind this dissertation is to create a clear definition of developer profiles that will categorize what kind of work a certain developer does. There

are some associated goals that are the main focus of this dissertation and are defined below.

- **G1.** This first goal is related to the idea that it is possible to reliably detect and categorize data so that it is possible to define developer profiles. These profiles provide the basis for this dissertation outcomes and analysis and will need a theoretical study since a standard regarding these does not exist.
- **G2.** The second goal approaches the way that the collected metrics can be used to objectively identify the developer profiles.
- **G3.** The third goal refers to the way that the data can be manipulated to generate comparisons between different data sets. Along with the developer profiles it is desired to check other metrics like the general repository information or the if the architecture of the software is actually compliant with the overall documented design.

1.3 Research Questions

The main objective of this dissertation is intrinsically related with its goals which means that it is possible to extrapolate various research questions that will sustain this dissertation content and outcomes. These questions are defined as follows:

- **Q1.** Did the analysis of the repository data lead to accurate and properly defined developer profiles?
- **Q2.** Can the developer profiles be used to accurately determine a developer contributions?
- **Q3.** Did the analysis between projects accurately identify team compositions?
- **Q4.** Was the collected data enough to create metrics that control the overall health of the project?

1.4 Research Methodology

The research methodology will be based on the Design Science Research (DSR) framework which provides a set of steps and guidelines that can be used from the concept to the final outcome of the research topic. The figure 1.1 show the process definition.

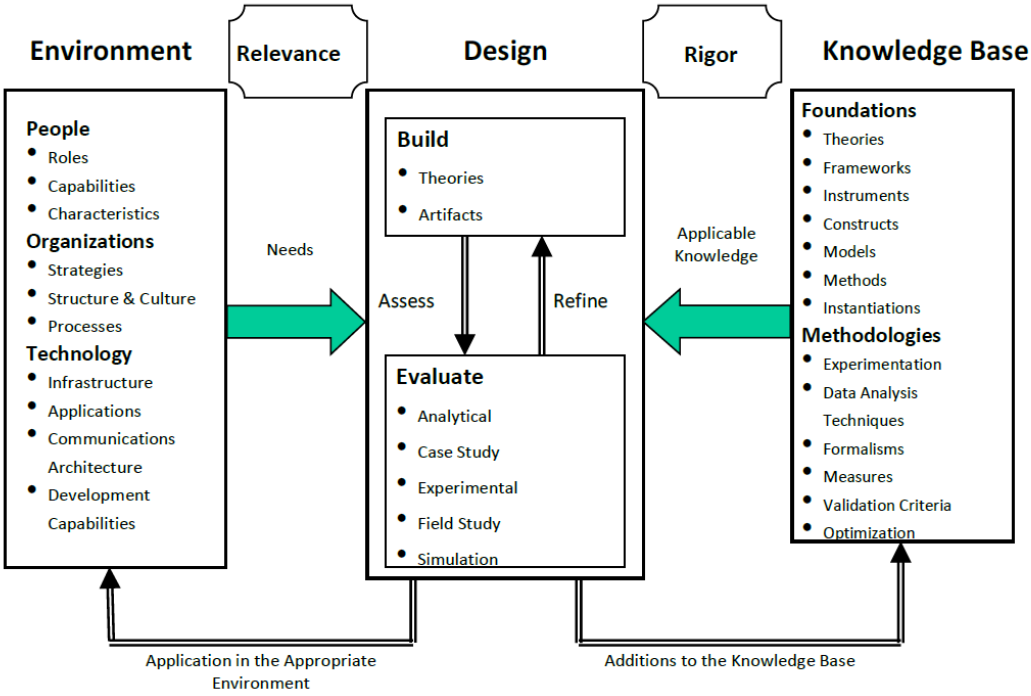

FIGURE 1.1: DSR framework (Brocke, Hevner, and Maedche 2020)

The performance of DSR projects has been based on several process models, such as Nunamaker, Chen, and Purdin 1990, Walls, Widmeyer, and Sawy 1992, Hevner 2007, and Kuechler and Vaishnavi 2008. The mostly widely referenced model is one proposed by Peffers et al. 2007 as referenced in Brocke, Hevner, and Maedche 2020, p. 5.

According to Peffers et al. 2007, there are six steps linked to the DSR process: problem identification and motivation, definition of the objectives for a solution, design and development, demonstration, evaluation, and communication. There are also four possible entry points: problem-centered initiation, objective-centered solution, design and development-centered initiation, and client/context initiation. Since this dissertation start at the problem identification and motivation the problem centered initiation was used and the six steps where defined as follows:

Problem identification and motivation This is where the research problems are defined and the value of the solution is proposed. This links to the problem presented in this dissertation and provides the basis for the motivation of the research into the dissertation themes.

Definition of the objectives The objectives allow the researcher to specify what can be accomplished with the current knowledge and problems definition. This is linked to the objectives present in this dissertation, these objectives can be quantitative as or qualitative as can be seen in the goals of this dissertation.

Design and development This step is where the DSR artifact is specified and is linked with the solution design and architecture and the outcomes of dissertation.

Demonstration In the demonstration step the experimentation methodologies are properly defined. This includes simulations, case studies or proof trough empirical data. In this dissertation the main idea is to prove trough empirical methodologies that the artifacts are valid and objectively correct.

Evaluation In this step the artifacts of the dissertation are evaluated and checked for how well they support the solution to the problem. This is related to the evaluation of the solutions trough statistical tests , questionnaires or other methods that can be used to grade the outcomes of this dissertation.

Communication In this final step, the results and all aspects of the problem are communicated to the relevant parties. In this case the this dissertation is the communication itself.

Taking into account what was said in the text above and the defined goals in 1.2, it is possible to specify how the evaluation of these goals will be approached. The goals **G1.** and **G2.** are linked to the theoretical study of the correlation between the collected metrics and the developer profiles. **G1.** mainly focus on the idea that the data can be categorized to create developer profiles. For this, when the general repository data has to be collected, a manual check must be done to understand what kind of profiles can theoretically be created and inferred.

After these profiles are specified, an approach to the manipulation and collection of the data must idealized and defined. At this time, only the GIT metrics are considered due to automation potential on gathering these as this automation will increase the speed in the data gathering processes. This whole process is used to fulfill the next goal (**G2.**)

Finally, the third and last goal, is related with the more technical part of the dissertation. This technical part includes some kind of dashboard that is used to actually represent and compare the collected data. This comparison is focused on specific project metrics that can be used to understand the difference between repositories.

1.5 Hypothesis

The hypotheses were formulated with the objective of testing the contents of this dissertation taking into account the goals and questions provided in sections 1.2 and 1.3, respectively. The hypotheses are defined as follows:

1. Validate that the identified developer profiles are based on specific metrics and can be applied to any project.

    a. **Null Hypothesis:** The hypothesis that it is not possible to create developer profiles based in information that is collected from a repository

should be reject. This would mean that it is not possible to objectively determine what kind of contributions the developer has the most.

b. **Alternative Hypothesis:** The developer profiles are successfully identified and are based in metrics that can extrapolated to any project. This means that it is possible to prove the validity of the profiles with concrete data.

2. Validate that is possible to correlate the data between repositories and generate comparisons between projects.

   a. **Null Hypothesis:** The hypothesis that the data collected from repositories cannot be used to compare different projects should be rejected. This would mean that the collected metrics have different interpretations depending on the project and would invalidate the work done in this dissertation.

   b. **Alternative Hypothesis:** The collected data can be used to compare different projects and is completely reliable.

1.6 Thesis Structure

This document is divided into three parts: introduction, body and conclusions. These parts are organized as follows:

Chapter 1 This is where the project context and the problem are introduced. The objectives and document structure are also presented here.

Chapter 2 This is the section includes the State the Art of the technologies and processes present in this document.

Chapter 3 This section contains the value analysis and the functional requirements.

Chapter 4 This is where the design for the chosen solution is presented.

Chapter 2

State Of The Art

In this chapter of the dissertation, the objective is to present the chosen technologies for tackling the issue and analyse what kind of valid alternatives exist in the current software landscape. The first part will be related to value analyses and will present the standards related to Fuzzy Front End (FFE), New Concept Development (NCD) and Political, economic, social, and technological (PEST) models. The second part will present the current state of the technologies that are relevant to this dissertation theme. These include Supply chain management (SCM) technologies, issue trackers and front end frameworks.

2.1 Value Analysis

This section will introduce the concepts of the presented value analysis that will be linked to this dissertation in chapter 3.

Value analysis is a structured and systematic process that takes into account the client requirements and the product purpose[1].

The main purpose of this analysis is to add value to a determined service or product while the lowest possible cost while maintaining an optimum balance between function, performance, quality, safety and cost[1].

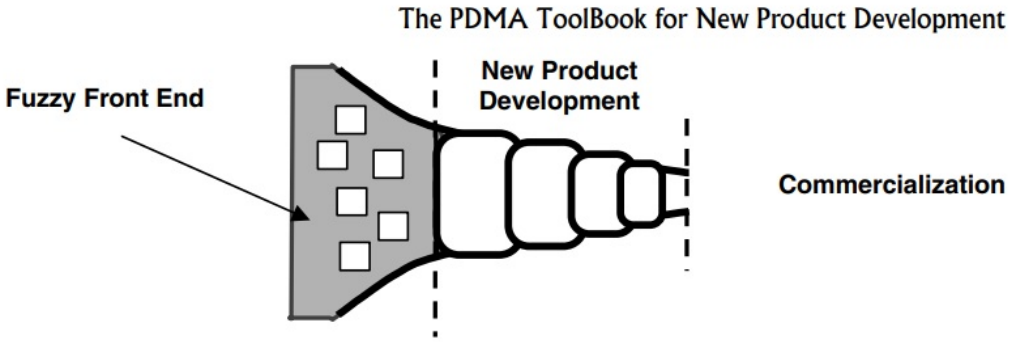

FIGURE 2.1: Innovation Process (Koen et al. 2002)

[1]https://www.value-eng.org/page/AboutVE

The process described in figure 2.1 can be divided into three parts:

FFE - Described as an environment of uncertainty related to time management, analysis of ideas and identifying opportunities.

New Product Development (NPD) - A contrast from FFE where the environment is stable and focused in the objectives and development of the product or service.

Commercialization - Promotion and selling of the product or service.

This dissertation uses the NCD model notation created by Koen (Koen et al. 2002) that is used to describe the various processes of innovation form the problem at hand to the proposal and selection of ideas and solutions.

2.1.1 New Concept Development Model

As can be seen in figure 2.2 the NCD model consist of an engine, five essential elements and the influencing factors. The arrows define that the process can start in the ideas or opportunity identification stage.

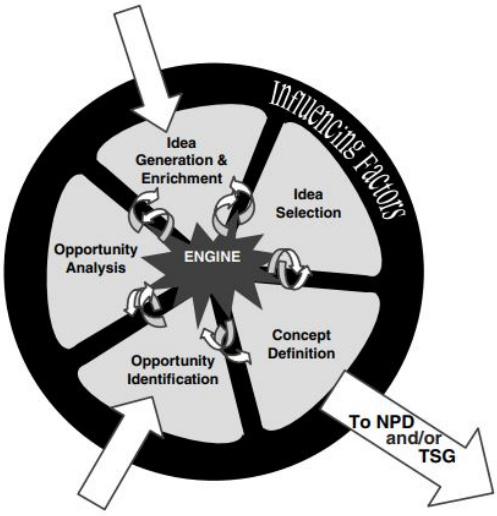

Figure 2.2: NCD (Koen et al. 2002)

Engine

The engine, which represents senior and executive-level management support, powers the five elements of the NCD model (Koen et al. 2002). The definition of these five elements can be seen below.

Opportunity Identification

This element is related to identifying large or incremental business and technological opportunities. These sources and methods for these can range from formal approaches like problem solving methods to less formal approaches like conversations with individuals.

Opportunity Analysis

In this element the gathering of additional information is done in order to convert the identified opportunities into business opportunities. The methods for this can involve focus groups, market studies or scientific experiments.

Idea Generation and Enrichment

Within this element, an iterative and refinement process is defined that contains brainstorming sessions and idea banks with the goal to generate build and reshape ideas. Due to the nature of these processes, some new ideas might come up during the process from internal or outside inputs.

Idea Selection

The fourth step is the idea selection, which defines the process of selecting and prioritizing opportunities or ideas based on their costs. These costs are related to monetary funding and time invested in some determined project.

Concept Definition

This element allows the business case to developed based on the estimates of other activities, customer needs, investment requirements and project risks and potential. The uncontrollable factors are defined by outside world conditions and rules and involved sciences maturity.

This model is supposed to be revisited and reevaluated since it is a circular based strategy where ideas must flow trough the 5 elements.

External Factors

This analysis was created using the PEST model. This model is a framework that allows for strategic planning when doing market research for some project.

Political

These factors are related with decisions, rules or laws that the government creates that affect the economy. The political stability of a government is directly linked with a country or nation infrastructure.

Economic

This factor governs economic growth, inflation rates and interest rates that have drastic effects on how a business function

Social

These factors include the cultural aspects, population age and social trends. Companies are required to have strategies to adapt to these factors and, in turn, these strategies affect costs and organisation structure.

Technological

Then technological Factors include all the technology aspects like research activity, available technological incentives and products. These define entry barriers

for a specific product or project and take into account technological shifts that affect costs and quality.

Canvas Business Model

This model is a strategic management template with the focus for developing and document business models. These business models help understand and assess the business position, values, customer base, revenue streams and dependencies.

2.1.2 Value Network

As stated by (V.Allee), "People naturally network as they work so why not model itself as network".

The above citation is from author Verna Allee and it means that people naturally tend to work as part of a network so the work itself can be modelled to work as network.

A value network is a complex web of dynamic relationships that compose a group or organization focused in exchanging tangible or intangible values (Allee 2008).

2.1.3 AHP Model

The Analytic Hierarchy Process (AHP) model was created by (Saaty 1990) and helps in the process of evaluation and decision making by allowing the usage if different qualitative and quantitative criteria.

2.2 Repository Analysis

Most of the metrics that compose the basis for this dissertation themes will come from tools that allow the analysis of the information contained inside a repository. The table 2.1 shows a comparison between the described repositories.

2.2.1 GitHub API

GitHub offers a public Application Programming Interface (API) that can be used to query project metrics. Unfortunately, there is a limit to the hourly number of requests that can be made so using it to query large projects is not recommended. It can be used to select a single project contributor to be observed and supports subscribing to project events that can be defined as needed.

Using this API, it is possible to view all the information that GitHub shares in it's web interface. However, the access to specific Git metrics like changed lines of code or repository users is limited.[2].

Advantages

- Robust API maintained by GitHub

[2]https://docs.github.com/en/rest

- Can be used with other tools

Disadvantages

- Imposed limits in the amount of queries
- Limited access to git repository metrics

2.2.2 Microsoft GHCrawler

GHCrawler is a robust GitHub API crawler that walks a queue of GitHub entities transitively retrieving and storing their contents[3]. This crawler is basically a program that simulates a user behaviour in a certain website and allows for multiple queries using GitHub API since it has a system that allows the management of various access tokens.

Advantages

- Uses the same API provided by GitHub
- Has a management system that controls several access tokens
- Can be used with other tools

Disadvantages

- Has the same limitations as GitHub API.
- It is bound to GitHub API technology

2.2.3 Apache Kibble

Apache Kibble is a suite of tools for collecting, aggregating and visualizing activity in software projects[4].

This tool is a more of full suite of tools and scanners that run and collect information, this information is then fed to the central Kibble server. These scanners allow for collection of data not only related to git repositories but other technologies like JIRA or mailing lists.

As long as there is data in the Kibble server, it is possible to create a dashboard with several widgets that allows for the presentation of the collected metrics.

Advantages

- Mostly technology agnostic since it depends on the scanners
- Works as a full suite that allows for the creation of a dashboard
- Open source and completely free

Disadvantages

[3]https://github.com/Microsoft/ghcrawler
[4]https://kibble.apache.org/

- Not as flexible to customize since it has baked in solutions

2.2.4 CHAOSS

The project Community Health Analytics Open Source Software (CHAOSS) is focused on sharing data and metrics that classify a projects health[5].

This project has several tools that allow for the collection of metrics contained in different repositories. The tools include Augur which is a web application that presents metrics on a specific project[6], Cregit[7] to get information on code changes and GrimoireLab[8] which is an open source platform to handle data gathering.

Advantages

- Provides a list of useful metrics
- Allows access to several tools

Disadvantages

- Some tools are not as mature as others

2.2.5 GitCompare

GitCompare is a tool that allows for extraction of various metrics of open source repositories. [9]

The type of metrics collected include the amount of commits, forks, contributors, issues and open pull requests on a certain project.

Advantages

- Allows collection and comparison of several projects.
- Provides a list of collected metrics.

Disadvantages

- It is a suite with a very specific purpose and it cannot be modified.

2.2.6 Git Inspector

Gitinspector is a statistical analysis tool for git repositories. The default analysis shows general statistics per author, which can be complemented with a timeline analysis that shows the workload and activity of each author[10].

Advantages

[5]https://chaoss.community/
[6]https://github.com/chaoss/augur
[7]https://github.com/cregit/cregit
[8]https://chaoss.github.io/grimoirelab/
[9]https://gitcompare.com/faq
[10]https://github.com/ejwa/gitinspector

- Allows collection and comparison of several projects.
- Provides a list of collected metrics.
- Can scan for different file types.

Disadvantages

- Only shows cumulative work by each author.

2.2.7 Git Quick Stats

Git Quick Stats is a command line interface tool that allows for the consultation of metrics related to all collaborators in a project. These metrics include the amount of insertions, deletions and updates that a developer pushed to the remote repository and the complete commit history of the repository [11].

Advantages

- Simple and easy to use command line interface.
- Easy to integrate with any technology that supports running terminals.
- Is able to export GIT log information.

Disadvantages

- More of a consultation tool than a full blown application.
- Limited to local repositories.

2.3 Issue trackers

This section includes a description of the issue trackers that are used in conjunction of other technologies present in this dissertation.

2.3.1 JIRA

JIRA is an issue tracking tool that helps teams plan, manage, and report on their work. This tool has several different packages depending on what is necessary to a specific project, these packages range from management focused with control over what is being done to a documentation repository that allows the users to share the documentation in a wiki styled environment.

2.4 Front end Technologies

In this section the current technologies used for front end development are presented and described. Although there are a few solutions available only React

[11] https://git-quick-stats.sh/

and Angular where considered due to their growing adoption and the author knowledge.

2.4.1 React

React is a JavaScript based library that allows for the development of web applications [12]. This library is widely used and has support of a huge community and, due to that, has a lot of packages that range from simple User Interface (UI) packages to complex libraries of components with complex interactions. It has very good performance it is and easy to maintain if the general recommendations are followed like using composition to create components or taking advantage of the provided helper functions.

It can be used to create a Progressive Web App (PWA) as well, which allows the flexibility of create an application that can run in an offline environment.

2.4.2 Angular

Angular is an application design framework and development platform for creating efficient and sophisticated single-page apps [13]. Since it is framework, it provides a lot of functionalities that can be used to maintain proper code formats and application architectures. There are set of rules on how to properly take advantage of the framework and it is pretty common to develop project with heavy usage of observable patterns.

Like React it can also be used to create a PWA.

2.5 Back end Technologies

In this section the considered technologies for the back end component of the solution. In this case only one technology was considered since just a small and simple script was needed to support the client application.

2.5.1 Node.Js

Node.js is a JavaScript-based platform used to develop online applications that include Single Page Applications (SPA) and other complex systems like streaming platforms or complex business functionalities. This technology is open-source, completely free and, at the moment of writing, used by thousands of developers around the world [14].

[12] https://reactjs.org/
[13] https://angular.io/docs
[14] https://nodejs.org/en/about/

2.6 Supply Chain Management

This section contains the current considered technologies for file versioning.

2.6.1 Git

Git is a version control system designed to allow the introduction of content inside a repository from various users while maintain a history of editions and source control[15].

2.6.2 SVN

The Apache subversion is an open source version control system and supports full versioning of files in a centralized system. It has widespread adoption and has a rich community of developers and users [16].

2.7 Summary

As described before that are several tools that allow for the collection of metrics present in repositories and all the other associated technologies. These tools are available as standalone projects to be integrated with other software or as full suites with predefined data and metrics for collection as can be seen in the comparison table below:

[15]https://git-scm.com/
[16]https://subversion.apache.org/

Table 2.1: Repository tools comparison

Repository tools							
	GitHub API	Microsoft GHCrawler	Apache Kibble	CHAOSS	GitCompare	Git Inspector	Git Quick Stats
Is free to use?	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Has limitations with tokens?	Yes	Yes	No	No	No	No	No
Is just based on GIT repositories?	No	No	No	Yes	Yes	Yes	Yes
Is a standalone tool?	No	No	Yes	Yes	Yes	Ye	Yes
Can be extended or incorporate into other solutions?	Yes	Yes	No	No	No	Yes	Yes

Chapter 3

Analysis

In this chapter, the concepts that were introduced in chapter 2 will be linked to the project described in this dissertation. The functional and non functional requirements will also be present.

3.1 Business Value

This section will link the concepts introduced in subsection 2.1 to this dissertation content and will add more information. The value, perceived value and customer value are also present here.

3.1.1 Five Elements of a innovative activity

This section provides a contextualization the concepts introduced in subsection 2.1.1 in this dissertation.

1. **Opportunity Identification**

   With the normalization of remote work in the developer space, the pool of contributors to a software project is now tied to the global market(Ford et al. 2021).

   Taking this into account and the openness to remote work we can identify an opportunity to build on the current documented work with not only project metrics comparisons but also the profiling of the developers.

   There are several tools that look trough different open source repositories to check and log specific projects health metrics. These metrics can also be used to predict how the project will evolve and the possible outcomes(Xia et al. 2020).

   The profiling of the developers will give more information on the factors that affect project outcomes since it will allow for comparison between team compositions and can be expanded to understand what are the most successful ones.

2. **Opportunity Analysis**

   In order to validate this opportunity a research was conducted on the current market tools and what is offered to a user of these platforms. This research led to the conclusion that there are several platforms that monitor open source projects and that these are of high interest due to the metrics they produce.

   These metrics can easily be used to compare different projects statuses and conditions. Taking this into account it was inferred that the logical next step would be to also take into account the type of developer that participates in the project.

3. **Idea Generation and Enrichment**

   The main idea is to correctly and objectively categorize the profiles that are associated with each developer but it was expanded to a more complex use case where the developers profiles are taken into account when collecting the metrics from a repository.

4. **Idea Selection**

   Following the discussion between the proposed ideas and after taking a look at what exists in the current landscape, an investigation was proposed that will look into current solutions and products to evaluate if the main idea behind this dissertation is valid.

5. **Concept Definition**

   After studying the results of this dissertation it will be possible to make a case for investment in the proposed solution. These results will take into account the commercial and technical risk factors since it is possible that some new software will increase the cost or affect the reliability of the product.

3.1.2 External Factors

This section provides a contextualization the concepts introduced in subsection 2.1.1 in this dissertation.

Political The global market of developers due to remote work conditions allows for the study and categorization of metrics and profiles.

Economic Taking into account the cost of development of a software, it would be beneficial to understand the rate of success of projects and their team compositions with more detail.

Social The dominance of technological forces and the need to develop and disseminate innovations in several markets are characteristics of international industries in the modern world(Shkalenko and Fadeeva 2020). This means that the trend is to have a software associated with an industry and having

a way to identify the teams needs is beneficial to the overall success of the project.

Technological Taking into account the amount of technologies that are available on the market there are several ways to developer a tool or software that can integrate with many systems.

3.1.3 Value

As stated by (Nicola, E. P. Ferreira, and J. J. P. Ferreira 2012, p.1), 'Value has been defined in different theoretical contexts as need, desire, interest, standard-/criteria, beliefs,attitudes, and preferences. Value is, therefore, very dependent on perception.'

This dissertation will be based on expanding the current set of tools with new functionalities and definitions. The value will be achieved by all the architectural and tools advancements that will be created.

3.1.4 Perceived Value

Value can be perceived differently by different customers and organizations. In fact, there is no direct correlation between the costumer perceived value and the suppliers perception of the costumers value delivery (Ulaga and Eggert 2006).

The proposed approach and advancements will allow for more information to be gathered and used in comparisons between projects. This information and categorization of profiles can be used from comparison of similar projects to the composition of teams which can be linked to overall project costs.

3.1.5 Business Perspective

This dissertation will propose a solution that addresses the problem described in section 1.1.

The proposed solution can lead to a set of tools that allow for a more deep understanding of project needs and could lead to a bigger advancements or interactions that could be introduced in the market as a commercial product.

3.1.6 User Perspective

Woodall (Woodall 2003, p.3) states that value for customer is 'an ambiguous appendage that can be used to represent both what the customer perceives/receives and also what the customer can deliver'.

The customer must analyze and decide if the proposed method can indeed lead to improvements into their own projects or teams. This dissertation will help this decision by providing metrics and data based on the conclusions of the dissertation.

3.1.7 Business Canvas

As can be seen in figure 3.1 a canvas business model was created for accessing of the value of the solution. It is important to note that there are no direct revenue streams since it is a project will just work as a platform to consult the information.

FIGURE 3.1: Canvas Business Model

3.2 Requirements Engineering

This section includes information related to the engineering of the solution. The functional requirements and non-functional requirements are listed here. The functional requirements identify what the system must support in order to be considered successful. If one of these requirements is not met the system will fail. In contrast with this, the non-functional requirements describe how the system works. These requirements usually govern speed, availability, reliability or usability which affect the system performance but are not needed for it to work.

3.2.1 Functional Requirements

In the creation of the new process the following functional requirements where defined:

FR1 The application must allow collection of information from repositories;

FR2 The application must show the developer profiles data;

FR3 The application must allow for comparison of repositories;

FR4 The application must allow for the creation of dashboards;

FR5 The application must allow for the definition of intervals for consultation;

To satisfy the requirement defined above, a use case diagram was created to specify how a specific user would use a client application go get and map the repository information. These interactions and systems can be seen in figure 3.2.

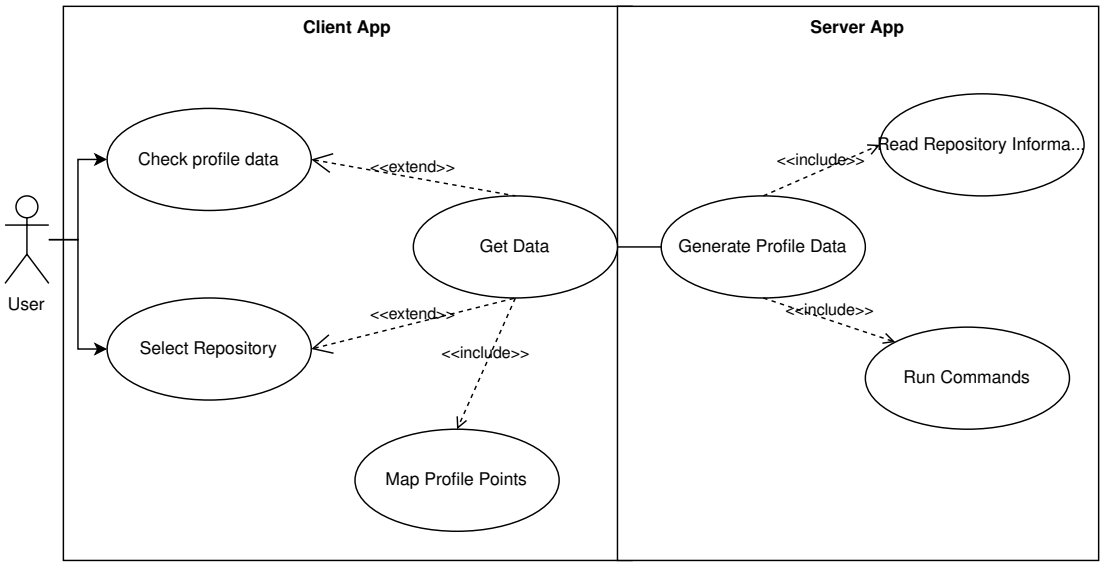

FIGURE 3.2: Functional Requirements

3.2.2 Non-functional Requirements

In the creation of the new process the following non functional requirements where defined:

NF1 The application should gracefully fail if a request has errors;

NF2 The application should handle large requests without blocking;

NF3 The application should be intuitive to use;

NF4 The amount of data will be huge so a solution with a MapReduce approach might be needed to process the data sets, this means that a client only application might not be the best solution;

3.3 Summary

There is not much information in the area that is being studied in this dissertation, in consequence, there is the possibility of creating a base case study for future reference. The value of the dissertation is not only linked to this but also to the validity of the defined profiles. These profiles can only be considered valid if the predetermined requirements are satisfied by the created solution.

Chapter 4

Design

This chapter presents the chosen design that will define how the app will be constructed and its alternatives. It is also here that the design requirements will be introduced and described.

4.1 Architecture

The architecture of the application is directly linked to how the information is available and how it it can be consumed. It is possible to create a client application that behaves differently depending on how the information is obtained and preserved. The overall design of the system can be seen in the diagram below.

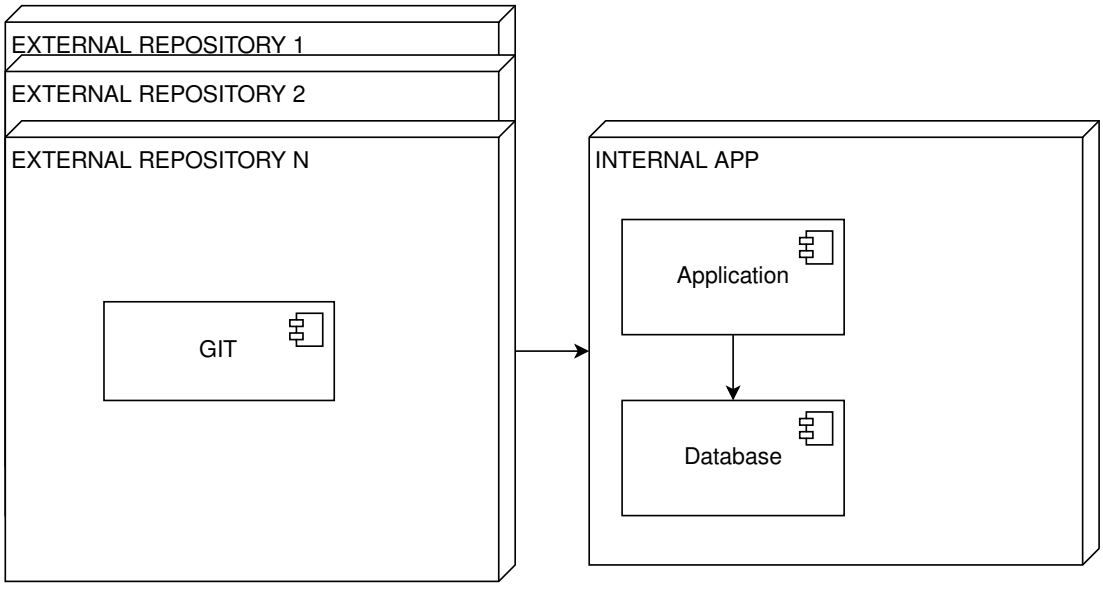

Figure 4.1: Overall Deployment Diagram

As can be seen in figure 4.1 the data will be collected from any external repository that uses GIT as its versioning system. The repositories will be queried for their information taking advantage of several commands that GIT supports. These commands will allow for the creation of metrics that will then be used in the internal app and will be saved in some kind of database. There are different

approaches to how this kind of behaviour can be implemented as can be seen in subsection 4.2.

4.2 Application Architecture

In this subsection the main approach to the application development and its alternatives will be described and detailed.

4.2.1 Web Application

This is a very common approach and usually has support from a back end component to handle the most complex functions. The overall design for this architecture can be seen in the next figure (4.2).

In this approach a back end application exists that will consume the information from the repositories. This information will then be available for consultation in the front end application through an API. The user will then be able to create a dashboard from the metrics which are basically a set of filters on the available information.

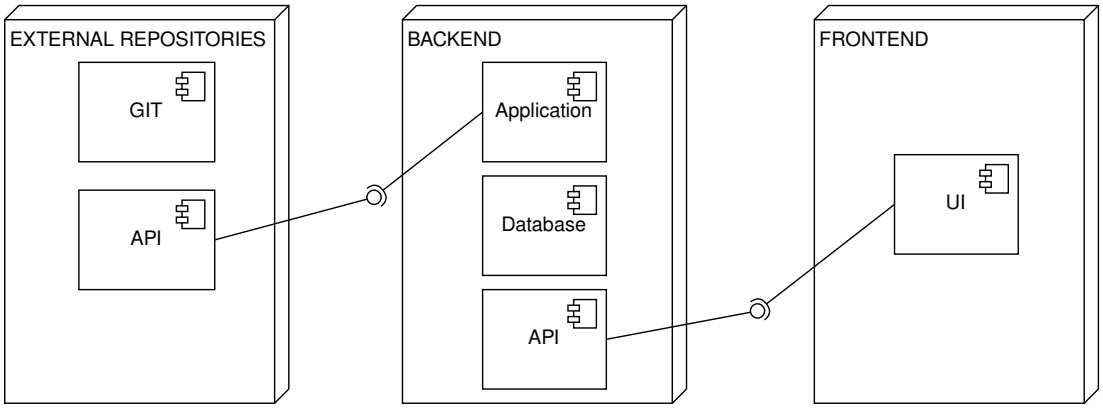

Figure 4.2: Web Application Architecture

4.2.2 Progressive Web App

As the title indicates, this is an approach where the application works without connection to the internet since it uses a local database.

PWA with front end processing

In figure 4.3 it is possible to see an example where the back end would just have a sync service to receive information and handle it. The information would be obtained by the front end part of the application and would be save locally. It is important to note that, this is not mandatory as the overall behaviour of the application can be very similar to the one present in figure 4.2 where we will need to have a some sort of sync service logic if we want to save the metrics in the back end.

The most important change in this approach is that, if the front end is actually querying information directly from the repositories, a token management system for the repository and all the complex logic will need to be present in the client application which would make it consume a lot of system resources. This performance challenge would not be the only issue with this approach, security issues while handling the information from the repository could also arise. These issues are related to the handling of possible private information that could include specific user data that could be leaked if the client system is compromised.

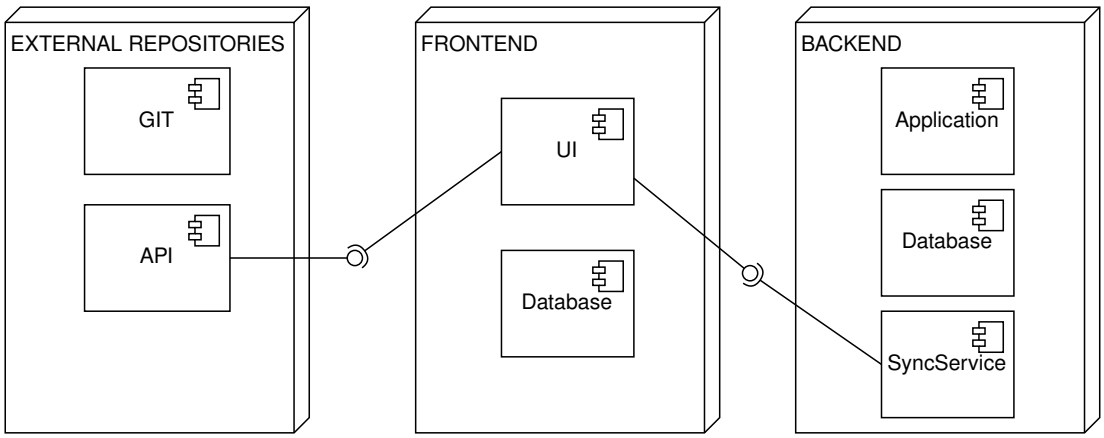

FIGURE 4.3: PWA Architeture with front end processing

PWA with back end processing

In this approach the logic is mainly the same as seen in the approach described before this one, however, the main difference would that the heavy operations would still be done in the back end part of the application. This means that the front end application would consume data form the back end like a conventional web application but would have a local database that would permit the app functions to work normally until the next synchronization. This different can be visualized in 4.4.

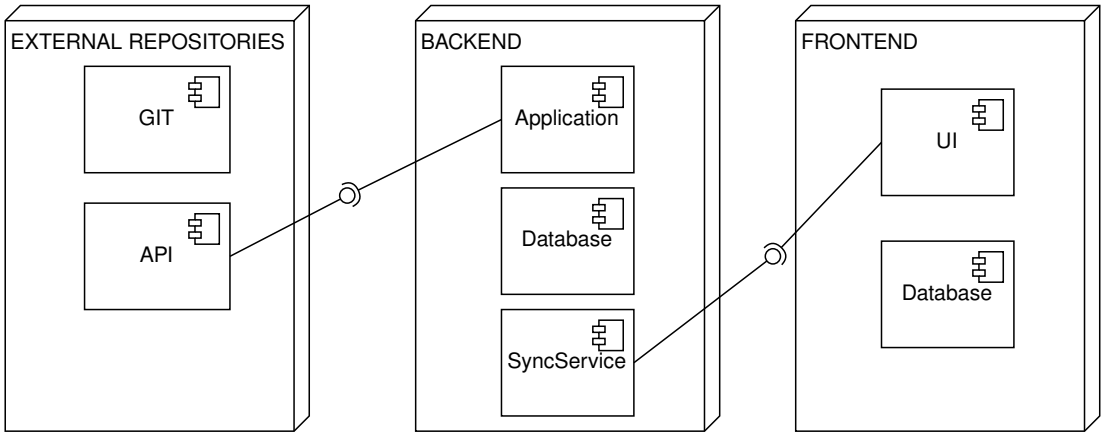

FIGURE 4.4: PWA Architeture with back end processing

After taking a look at each of the above approaches and the specific goals of this dissertation the web application approach was chosen. This means that the client app will only show and map the data that is available from a back end API. The back end application will handle all the heavy operations including reading and transforming the repository information into the expected format. A system like the chosen one can scale pretty well since most of the heavy logic is in a controlled environment.

4.3 Profile Analysis Algorithm

In order to properly handle the transformation and analysis of the collected data, an algorithm was created. This algorithm is based of keyword matching and will search the commit messages and changed file names for specific words that are represent in commits that are linked to the specified profiles.

The developer profiles are attributed by a point system that takes into account the number of hits that the keywords have with the commit history of a specific developer. This point system allows for the creation of a profile trend which enables the identification of the collaborator major and minor roles.

The sequence diagram 4.5 details how the interactions will work between the two layers of the application.

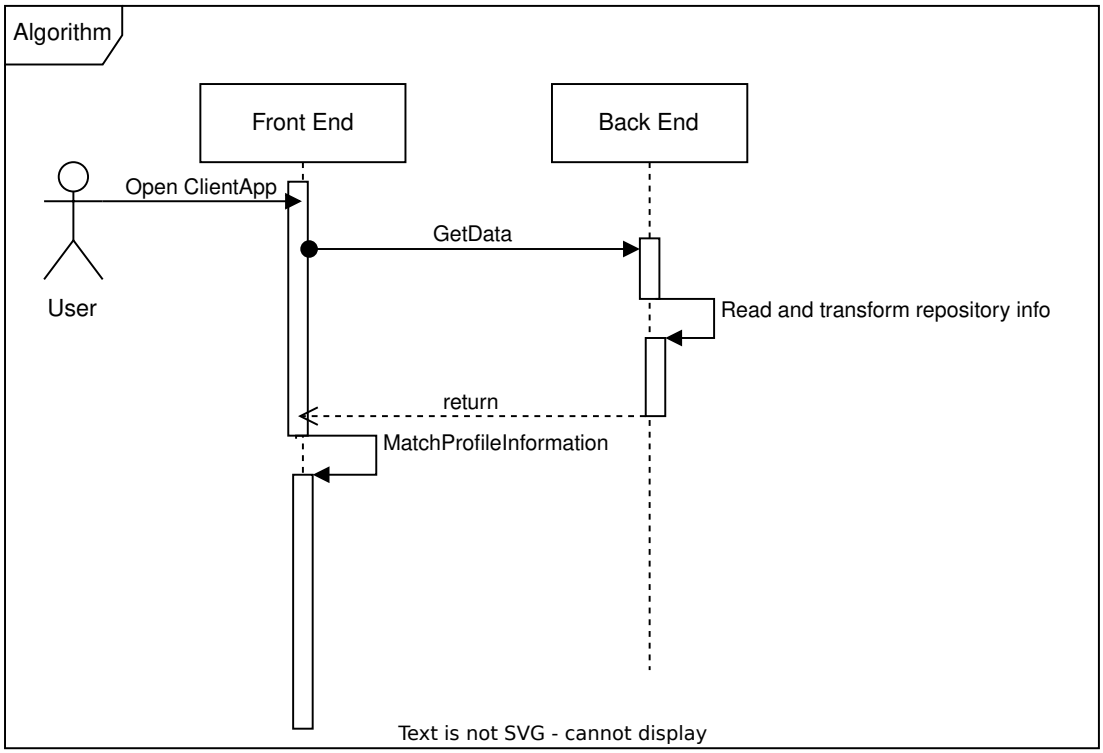

Figure 4.5: Profile Analysis Sequence Diagram

The pseudo code related to the profile match algorithm can be seen in 4.1.

```
1  //After preparing the data to be read by the algorithm
2
3  data = getBackendData();
4
5  function matchProfiles(){
6  profilePoints = 0;
7          loop(profileDefinitions)
8              shouldIncrementProfilePoints = commitMessage has
       profileDefinitionKeyword OR (filename has
       profileDefinitionKeyword AND is not file deletion);
9              if(shouldIncrementProfilePoints)
10                 increment profilePoints
11
12 return profilePoints;
13 }
14
15 loop (data)
16     if (data entry has information)
17     matchProfiles();
```

Listing 4.1: Profile Analysis Algorithm

4.4 Profiles

In order to validate this dissertation proposal it was decided that the profiles that were to be validated should be defined as such:

- **Coder:** This profile is the one where the most of the contributions are done by just committing code into the repository and should be present in most analyses.
- **Tester:** This profile measures that the contributions of the developer are mainly related to tests on the available code base.
- **Documenter:** The documenter profile measures if the developer produces a considerable amount of documentation in the code, this documentation is related to comments or descriptions of functions.
- **Arquitect:** To measure this profile it is necessary to check if the developer produces versioned diagrams or architectural artifacts.
- **Infrastructure Engineer:** The infrastructure engineer profile checks if the contributions done by the team member are mainly based around versioning of infrastructure related artifacts. These artifacts are related to pipeline files or infrastructure configurations of the project.

The above profiles are the base of this dissertation work and are linked to the information present in GIT repositories. This information is categorized as committed files extensions, commit messages, number of changed files and lines and the actual content of the commit. These dimensions can be used to infer the

above listed profiles as possible definitions of each developer or collaborator. In a mathematical approach a profile can be defined by the formula below.

$$\sum = MatchedCommits + MatchedFileNames$$

It should be possible to apply each profile to the specified formula. The formula can be interpreted by saying that some profile will basically be a sum of points based on matched keywords on commits and filenames. These keywords should help isolate specific words in a commit or filenames, in turn, linking them with a profile.

4.5 Requirements

The functional requirements defined in sub section 3.2.1 defined the design of the applications to collect, map and show the data from the repositories. The first functional requirement is related to the collection of the data from the repository and is handled by a back end application. The diagram 4.6 shows how the extraction of information works.

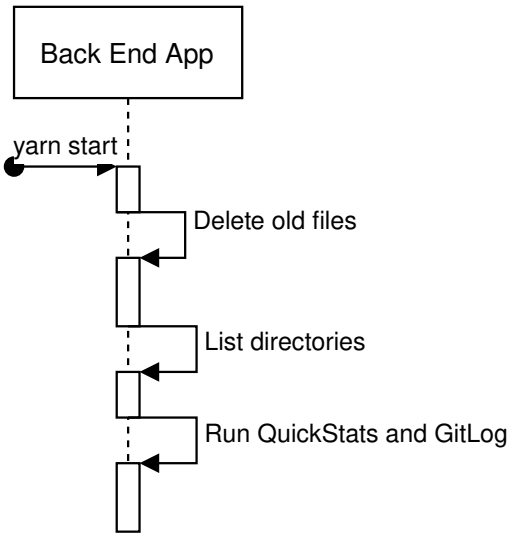

Figure 4.6: Data Extraction From Repository

According to the diagram above the extraction algorithm should start by deleting the old files that were generated in previous run, in case of updates. After this task completes, the script should list all available directories and run the commands into each of them. The information must be persisted somehow, in this case the proposed approach was in memory.

The rest of the functional requirements specify what kind of actions the user is allowed to do in the client application.

The client application will support an endpoint call that returns the data from the repositories. After getting this information, the profile matcher algorithm runs

on this data and categorizes it according to the defined rules. Finally, the user is able to interact with the application according to the predetermined functional requirements. The diagram 4.7 describes the expected behaviour.

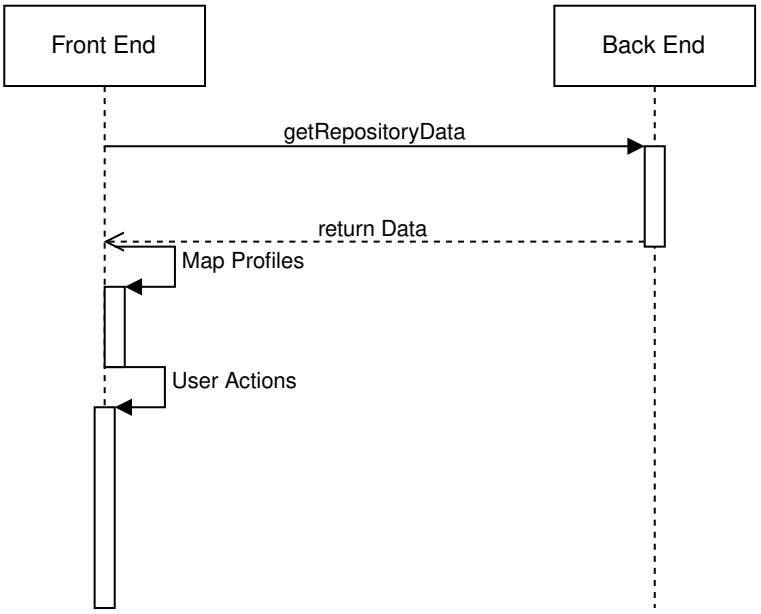

FIGURE 4.7: Client application sequence diagram

4.6 Summary

There are some requirements that were taken into account when the different approaches were elaborated. Since this is mainly a front end focused application, the need for performance and clearly separated responsibility layers was necessary. This means that, even if the same functionalities could be achieved with just a client application, the end experience and usability were considered.

The application itself should not do the heavy complex operations in the information due to it being dependant on the client specifications. This approach means that the application design must contemplate some kind of communication with a back end environment.

Since the front end web applications properly support offline modes the design should also contemplate these approaches as a valid solution.

Chapter 5

Implementation

The main objective of this dissertation is to create and define profiles based on GIT information and other enveloping systems like issue trackers, wikis and content delivery systems.

5.1 Technological Stack

There are several technologies present in the proposed solution that allow for the collection and representation of the experiments conducted in this dissertation. These technologies are related to the final expected outcome which is a dashboard that is be served as an web application. The technological stack can be defined as follows:

Back End The back end is composed by a Node.Js application that allows for the automation of the commands present in git quick stats and provides endpoints to be consulted by the client app.

Front end A React app that can request the back end for information and provides a dashboard that displays the collected data.

Git Quick Stats This tool provides a command line interface to consult a git repository for several stats and collaborator information.

Issue Trackers These include any technology that allows for software issues tracking and wikis about a certain project.

The implementation must satisfy the defined requirements present in subsection 3.2.1. The technology stack was chosen to implement the predefined requirements and can be linked as follows:

FR1 Is implemented to the back end application and the git quick stats tool in order to collect the repository data.

FR2, FR3, FR4, FR5 Are implemented by the React application which will allow the user to visualize and interact with the information.

NFR1, NFR2, NFR4 Are implemented in conjunction by the front end and back end layers.

NFR3 Is implemented by the front end layer in order to have a good use experience using the application.

The final artifact of this dissertation must satisfy all the functional requirements and the implementation details are shaped by them.

5.2 Implementation details

In order to extract the information about some collaborator, a combination of the tool Git Quick Stats and the command git log since the tool does not return all the required info. This combination will be handled by a small Node.Js application which will handle the logic to execute the commands and provide the API that the client application will call.

To keep the back end application simple, the Node.Js script expects that the repositories are downloaded into a specific folder and it executes specific commands sequentially for each of the present repositories and it saves read stats in memory storage. This is not ideal but, since the main focus of this dissertation is to find out which developer profiles exist, this implementation was chosen and can be easily expanded and modified to integrate other technologies which include supporting a database.

The client application allows for the visualization of the data in a dashboard, as can be seen in 5.1.

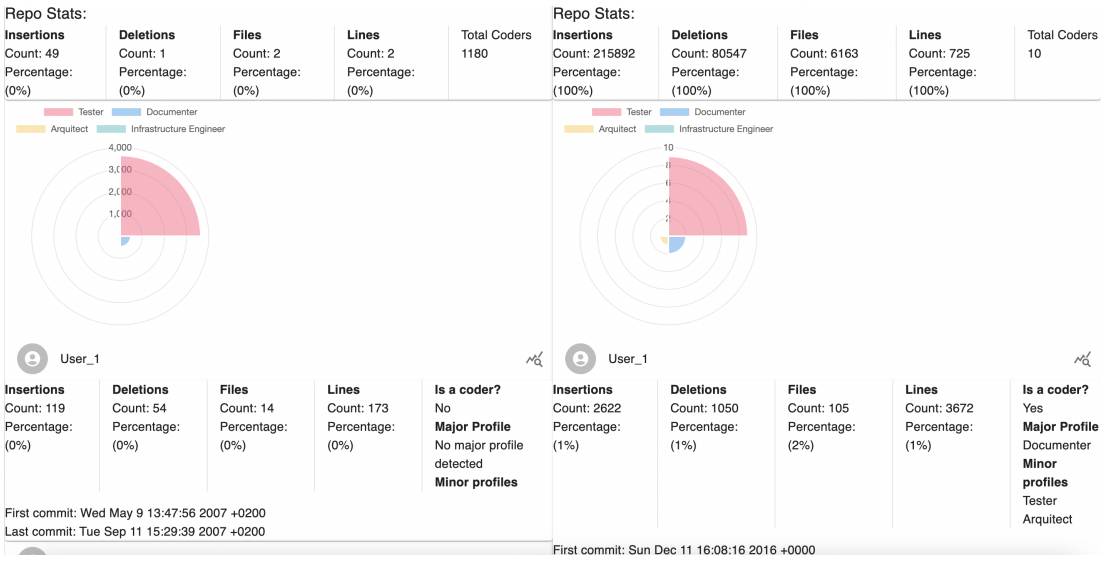

FIGURE 5.1: Main dashboard

This enables the user to use the dashboard to check the developer profiles, the stats of the consulted repository and can be used to compare different collected information about any repository.

It is also necessary to understand that GIT repositories have limitations on what kind of information can be extracted and, due to these limitation, the inferred

developer profiles are mainly based around these limitations. The information that is collected can be enriched with systems or tools that usually are present with GIT repositories, as a result, the accuracy of the defined profiles will increase. These system include issue trackers, wikis and software pull requests and will require manual interaction to extract the extra data. Due to the objective behind this dissertation being the definition of the developer profiles, no time will be invested into automating these consultations and instead the client application will provide a way to enrich or override the developer profiles with the data extracted from these external systems.

Like it was described, the client app and server app are mainly used to automate and support the process of getting and showing the stats from the repositories. The expected golden path of these consists in the user opening the client app and waiting for the information to be synced. This information is synced by calling a Representational State Transfer (REST) API endpoint that signals the back end application to start processing the available repositories. Each repository is consulted and the information is extracted using a combination of the Git Quick Stats tool and GIT commands. This is possible by saving the outcome of the commands in the terminal to a file and then reading them with a file reader routine. The detailed command definition can be seen below:

```
1 git-quick-stats -T > generalStats.txt
2 git log --name-status --no-merges --pretty=format:"newEntryInLog
    :%an,%ae,%s" > fullLog.csv
```

Listing 5.1: Repository Commands

The first command gets the general stats of the repository which consists in listing the authors and their total commits, insertions, deletions, changed files, first and last commit.

The second command outputs the user changelog with to a Comma-separated values (CSV) file. This output contains some format changes to make it easier to read and parse the detailed information on what files were changed, deleted or inserted in the commit.

The combination of all these stats are the basis for the conclusions that will be reached at the end of this dissertation.

5.3 Summary

The implementation described in this chapter was linked to the predetermined requirements. This includes choosing the technological stack and the logical approach to the automation of the collection and transformation of data.

The final outcome is a web application that is able to call an API endpoint that returns the information that can be mapped and analysed. The responsibility of reading and transforming the data sets into the expected format is delegated to the back end application. This means that the most expensive operations are

done in a controlled environment which is easier to scale and that it is possible to better protect the information about a determined repository.

Chapter 6

Experiments

In the course of this dissertation several experiments were made to test the algorithm resilience and the accuracy of the inferred profiles. To test the resilience of the algorithm, experiments were conducted taking into account the repository size. This size is directly linked with the amount of collected information which can lead to blocks, out of memory exceptions, long execution times or timeouts. To test the accuracy of the profiles, a certain number of developers were selected from four repositories and manual checks were made to understand if the developer was in the boundaries of the specific profile. The total number of chosen developers varied in relation to the repository number of collaborators.

6.1 Indicators

In order to conduct the experiments some indications were defined that would support the assessment of the chosen solution presented in this dissertation. The information collected was intrinsically correlated with the experiment that was being done.

The indications are divided into two different categories. The repository size, algorithm execution time and memory consumption were used to measure the performance and resilience of the solution that was developed to help define and collect information for the developer profiles. Excluding these technical indicators, there were other indicators collected to aid in the definition of the developer profiles like the number of developer commits, insertions, deletions, changed lines, file names and commit messages.

6.2 Assessment

At the beginning of this dissertation four research questions 1.3 were defined. To fulfill the objectives of this dissertation, these research questions need to be answered. In order to answer these questions, experiments were conducted and analysed.

The first part of this work was centered on collecting data for analysis and transformation and, after the collection, manipulation and transformation of all data, it was possible to check that using similar data would lead to the same developer

profile. While using these data sets it was possible to check what kind of files the developer committed and what were the profiles of each member of a team. Due to this fact, the first 3 questions were able to answered fully since most of this information was handled by the algorithm and the matching keywords system. It is important to note that, it is possible to enrich these answers with external systems and more information not present in GIT repositories. Finally, for the last research question, it was not possible to determine with confidence the project health with the collected data from GIT repositories alone. This metric would benefit greatly of the collection of that of external systems like issue trackers and pull requests as it would better allow for the correction between the information obtained in the GIT repository and these external systems.

6.3 Results

The results of these experimentation were obtained by running the same code and applying the same rules to the different repositories and can be seen in table 6.1.

Table 6.1: Repository data extraction metrics

Data Extraction Results				
	Repository 1	Repository 2	Repository 3	Repository 4
Number of contributors	11	753	2015	> 5000
Repository Size	190.9 MB	2270 MB	709.7 MB	5990 MB
Average QuickStats Consumed Memory	11.79 MB	1230 MB	268.91 MB	1318.25 MB
Average GitLog Consumed Memory	11.47 MB	271.8 MB	158.63 MB	1330.03 MB
Average QuickStats Execution Time	0.01s	0.17s	0.32s	4.28s
Average GitLog Execution Time	0.35s	8.16s	13.4s	64.4s
Number of commits	725	26078	92773	1040149
Number of files	6163	304618	361950	2423811

The metrics on the execution time and consumed memory were measured by running the scripts ten times for each repository. The values were extracted by taking advantage of the response time in the client application and the memory consumption information in the server.

The `perfomance.now()` command was used to measure the executions times in milliseconds. Taking a look at the results the average memory consumption is directly linked to the repository size which means that increasing the size of the GIT repository will also increase the consumed memory.

In opposition, the execution times were dependent on the number of commits and files which means more cycles were needed do transform the data. This is a similar behaviour as transferring a big amount of smaller files versus one big file, the overhead of writing and reading the small files will make the operation take longer.

Metrics on keyword match algorithm were also collected and can be seen in table 6.2

Table 6.2: Profile match metrics

Profile Match Results				
	Repository 1	Repository 2	Repository 3	Repository 4
Commit Match Average Execution Time	< 0.1s	< 0.1s	0.1s	0.1s
Filename Match Average Execution Time	< 0.1s	0.2s	0.4s	0.4s

Since the client application is based on React, it can handle mapping the data to the profiles effectively. The metrics in the table above show that the operations in the match algorithm are not creating a bottleneck in the execution time if the amount of data increases. It was also observed that, the rendering times of the application increased with the bigger data sets. This is due to the amount of information that is rendered.

Finally, ten developers were chosen randomly and manual checks were done to test the accuracy of the mapped profiles and their reliability. These checks include investigating the selected developer history of commits, file changes and any other contributions that fall outside the scope of the automated process.

For the purpose of validating these of scope contributions, manual checks were done on available external systems like issue trackers and pull requests platforms. The conclusions reached from these checks were taken into account on the final proposal of the profiles and affected the keyword match algorithm.

As can be seen on table 6.3, issues were found in each repository with more prominent issues appearing on the bigger repositories.

Table 6.3: Profile manual checks metrics

Profile Manual Checks				
	Repository 1	Repository 2	Repository 3	Repository 4
Number of checked developers	10	10	10	10
Issues	Mismatched profile	Mismatched profile, Incomplete profile	Mismatched profile, Incomplete profile, Unknown profile	Mismatched profile, Incomplete profile, Unknown profile

The three issues seen in the table above can be defined as follows:

- **Mismatched profile** The mapped major profile does not match the developer role in the project.
- **Incomplete profile** The mapped profiles are missing keywords in the matching algorithm.
- **Unknown profile** The developer contributions did not lead to a specified profile.

The first two issues are mainly related to the keyword dictionary no containing the correct search terms. In an effort to correct these issues, more terms were added to the keyword dictionary that were the result of manually checking the developer commits and file changes.

The last issue is related to a limitation that was found during the testing of the larger repositories. This limitation means that there is not enough information in a GIT repository to cover all the possible developer profiles. It was found out that other profiles could theoretically exist by taking a look at external systems. These profiles are mainly based on contributions on external systems that are usually linked with any software repository.

Chapter 7

Conclusion

At the end of the work, it was possible to confirm that, using only GIT metrics, the developer profiles can be generated and defined. These same metrics cannot be used to infer the overall health of the project by themselves and would need some kind of external system or analysis.

It is important to note that, the profiles can be limited in scope and accuracy due to lack of external systems that complement the information like issue trackers or software documentation wikis. Using keyword matching to get these metrics is a valid approach to categorize and segment the collected data, however, it might lead to performance issues and needs to be carefully managed when dealing with large data sets.

To conclude, there is definitely value in further pursuing this work and this dissertation presents the base work to understand if it is possible to create more technology oriented profiling to developers and collaborators inside projects.

7.1 Research Questions

At the beginning of this dissertation, four research questions were proposed. During the development of this dissertation it was possible to answer all of them. However, it was found out that the accuracy the answer can be influenced by external systems. This is due to the reality of software development while using these version control systems. There are several pieces of technology that are used in conjunction with the GIT repository to manage and enrich a project development process. These services and applications include more information that can be used to enrich the profiles that were identified in this dissertation.

The fourth question, which links the GIT metrics to the overall health of the project, was answered negativity. This means that the GIT metrics alone do not correlate, by themselves, with the health of the project. Some external system and analysis would need to be done to infer these kind of conclusions.

7.2 Contributions

One of the main contributions of this dissertation is the study and validation of the metrics that lead to the specified profiles. This means that, using the same type of repositories, other researchers should be able to reach the same developer profiles. These profiles are the basis of any future work as they were objectively defined and identified trough experimentation on different repositories. The other contribution is all the code developed in this dissertation which is available at: `https://github.com/DiogoSilvaP/git-profiling`.

7.3 Limitations

There are some limitation of the developed work. The first one would be that, due to the lack of externals system enriching the current profiles, it is only possible to infer that the developer as a specific trend in the profile. This means that the developer will get hits on a specific profile that could lead to false positives, for example, in the case that the developer did not contribute directly into the issue but did some commit about it.

The second limitation is more technical and is related with the way the algorithm to get the information about the repositories was implemented. This approach uses an sequential reader and returns an in memory data set to the client application. Due to this, the bigger the size of the repository the worst the performance of the solution is. The execution times and memory consumption values never reached levels that would be impracticable but can be improved dramatically with what is talked in the section 7.4

7.4 Future Work

This thesis can be further expanded in a few directions. The most relevant in the short terms would be to add external services support like issue trackers and wikis to further enrich the current developer profiles and create new ones. These could include sub denominations of current profiles like new types of testers that could include end to end testers and unit testers.

The second expansion would be to add the capability to analyse the content of the modified files, this would add insight in what the developer itself did in each commit. This information is vital to further refine the current developer profiles and add new ones like code commenters.

Finally, another interesting direction to take this work, would be to add the capability of defining the seniority level of a certain developer. This could include checking into systems that include pull requests and checking the number of comments, code smells and issues found with each pull request of a certain developer.

7.5 Personal Remarks

This work was a challenge due to its initial ambiguity in the beginning. Since it is not a very well documented area some concessions had to be made in the scope of the dissertation. A lot of conclusions were reached by experimenting with the data and could be disproved in the future by a better methodology or overall approach to the problem. Overall, i feel that this dissertation has the necessary ground work done to prove that it is indeed possible to implement a system that can take GIT metrics and transform their values into new information.

Bibliography

Allee, Verna (Jan. 2008). "Value network analysis and value conversion of tangible and intangible assets". In: *Journal of Intellectual Capital* 9, pp. 5–24. DOI: 10.1108/14691930810845777.

Brocke, Jan vom, Alan Hevner, and Alexander Maedche (Sept. 2020). "Introduction to Design Science Research". In: pp. 1–13. ISBN: 978-3-030-46780-7. DOI: 10.1007/978-3-030-46781-4_1.

Ford, Denae et al. (Dec. 2021). "A Tale of Two Cities: Software Developers Working from Home during the COVID-19 Pandemic". In: *ACM Trans. Softw. Eng. Methodol.* 31.2. ISSN: 1049-331X. DOI: 10.1145/3487567. URL: https://doi.org/10.1145/3487567.

Hevner, Alan (Jan. 2007). "A Three Cycle View of Design Science Research". In: *Scandinavian Journal of Information Systems* 19.

Koen, P. et al. (2002). "1 Fuzzy Front End : Effective Methods , Tools , and Techniques". In.

Kuechler, William and Vijay Vaishnavi (Jan. 2008). "The emergence of design research in information systems in North America". In: *Journal of Design Research* 7, pp. 1–. DOI: 10.1504/JDR.2008.019897.

Nicola, Susana, Eduarda Pinto Ferreira, and J. J. Pinto Ferreira (2012). "A NOVEL FRAMEWORK FOR MODELING VALUE FOR THE CUSTOMER, AN ESSAY ON NEGOTIATION". In: *International Journal of Information Technology & Decision Making* 11.03, pp. 661–703. eprint: https://doi.org/10.1142/S0219622012500162. URL: https://doi.org/10.1142/S0219622012500162.

Nunamaker, Jay F., Minder Chen, and Titus D.M. Purdin (1990). "Systems Development in Information Systems Research". In: *Journal of Management Information Systems* 7.3, pp. 89–106. DOI: 10.1080/07421222.1990.11517898. eprint: https://doi.org/10.1080/07421222.1990.11517898. URL: https://doi.org/10.1080/07421222.1990.11517898.

Peffers, Ken et al. (2007). "A Design Science Research Methodology for Information Systems Research". In: *Journal of Management Information Systems* 24.3, pp. 45–77. DOI: 10.2753/MIS0742-1222240302. eprint: https://doi.org/10.2753/MIS0742-1222240302. URL: https://doi.org/10.2753/MIS0742-1222240302.

Saaty, Thomas L. (1990). "How to make a decision: The analytic hierarchy process". In: *European Journal of Operational Research* 48.1. Desicion making by the analytic hierarchy process: Theory and applications, pp. 9–26. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(90)90057-I. URL: https://www.sciencedirect.com/science/article/pii/037722179090057I.

Shkalenko, A.V. and E.A. Fadeeva (Jan. 2020). "Analysis of the Impact of Digitalization on the Development of Foreign Economic Activity During COVID-19 Pandemic". In: DOI: 10.2991/aebmr.k.200502.197.

Ulaga, Wolfgang and Andreas Eggert (Mar. 1, 2006). "Relationship value and relationship quality: Broadening the nomological network of business-to-business relationships". In: *European Journal of Marketing* 40.3-4, pp. 311–327. ISSN: 0309-0566. DOI: doi:10.1108/03090560610648075.

Walls, Joseph G., George R. Widmeyer, and Omar A. El Sawy (1992). "Building an Information System Design Theory for Vigilant EIS". In: *Information Systems Research* 3.1, pp. 36–59. ISSN: 10477047, 15265536. URL: http://www.jstor.org/stable/23010780.

Woodall, Tony (Jan. 2003). "Conceptualising 'Value for the Customer': An Attributional, Structural and Dispositional Analysis". In: *Academy of Marketing Science Review* 12.

Xia, Tianpei et al. (2020). *Predicting Project Health for Open Source Projects (using the DECART Hyperparameter Optimizer)*. arXiv: 2006.07240 [cs.SE].