# Data Analytics Framework for GAP Package Management

Thea Holtlund Jacobsen, 220024877
Supervisor: Dr. Olexandr Konovalov
School of Computer Science

This thesis is submitted in partial fulfilment for the degree of
**MSc in Computing and Information Technology**
at the University of St Andrews

**August 2023**

# Declaration

Date: August 15th 2023

[illegible]

# Abstract

Reproducible data analytics holds a pivotal role in software redistribution—a multifaceted task demanding careful consideration of numerous variables. This especially holds true for free, open-sourced software, as the nature of its development usually involves volunteers contributing their time and efforts without direct financial compensation. Consequently, it is imperative to streamline the redistribution process for efficiency, and employing tools for optimisation can significantly help reach this objective. This makes the study of such systems of great value to the software industry, by enhancing the quality of redistribution.

With the aim of easing the aforementioned process, this dissertation project introduces an analytical framework tailored for the GAP (Groups, Algorithms, Programming) programming language. The framework, created using Python and Jupyter Notebook, is comprised of notebooks for data extraction, analysis and visualisation based on GAP repositories hosted on GitHub. The project's main objectives revolve around generating outputs concerned with individual packages, version updates, testing patterns and contributor dynamics, to enable identification of trends, patterns and potential issues related to redistribution in a structured and user-friendly manner.

Built on the principle of separating code and data, the result of the project development efforts is a framework allowing for the interpretation of analytical outputs to facilitate better decision-making. Its application holds the potential of enhancing the efficiency of redistribution management, addressing a gap identified in existing field literature. The procedures for designing, implementing and testing the framework are detailed in this report, accompanied by code files that combined form the referenced analytical tool.

# Acknowledgements

The author wishes to acknowledge the contributions of the following individuals and institutions in the completion of this project:

- Dr. Olexandr Konovalov, for the time he has spent guiding me through the process, and for continuing to display such a genuine interest in the field. It has made the work of completing this project all the more motivating, and influenced my professional aspirations and vision for future career pursuits.
- My family, for all the love, patience and continued support. Thank you for encouraging me and putting up with me. My sincere gratitude for having a family like you grows every day, and you inspire me.
- The friends I have met here, for giving me a home away from home. The experience would not have been the same without you, and I will forever be grateful for sharing this journey with you. You have all made this year one I will never forget.
- Last but not least, to St Andrews for all the memories, and for giving me one of the best years of my life. The town, beaches, nature and people here are truly special. My time here has been so good for me, and has changed the way I see the world.

# Contents

# List of Acronyms

**API:** Application Programming Interface

**CI:** Continuous Integration

**CoP:** Communities of Practice

**EDA:** Exploratory Data Analysis

**FOSS:** Free and Open-Source Software

**GUI:** Graphical User Interface

**HTML:** HyperText Markup Language

**HTTP:** Hypertext Transfer Protocol

**JSON:** JavaScript Object Notation

# List of Figures

# List of Tables

# Chapter 1: Introduction

## 1.1 Background

In a world where numbers, figures and statistics drive decisions, and technological advancements in the scientific community occur at an unmatched rate, data analysis has become a focal point of academic, corporate and public interest. This has led to an industry dependence on high-quality analytical tools that consistently deliver accessible, reliable and reproducible results. Such frameworks are pillars in software distribution, allowing for optimised data management pipelines. This dependency is especially evident for FOSS (Free and Open-Source Software), where the focus lies on effective management and cost minimisation.

Evidently, the process of easing release management is thus of societal importance, further supported by the notion that universal access to software is key in driving innovation and encouraging all eager minds to contribute to technological advancements. Not only does universally accessible software imply that it must be user-friendly and open-sourced, it also means software should be available to individuals of all financial backgrounds [42]. Consequently, reliable tools to analyse data are core in securing efficient distribution and redistribution of free and available software, while ensuring transparency and fostering inclusive collaborative development [35].

One instance of such software is GAP, a computer system designed with a focus on group theory, algebraic computations and programming [48]. It is supported by a community of voluntary contributors collaborating on individual GAP packages. These packages, any developments, potential testing and the corresponding community are all variables of interest in the process of redistributing GAP.

Through the application of high-quality, reproducible data analytics tools, aimed at providing interpretive workflows yielding data which can be replicated by others using identical methods, the time and cost of the redistribution process can be reduced. Thus, for the field of scientific computations to move forward, there is a need for frameworks such as the one developed for this project. Not only will such methods aid in ensuring the quality of software products; it will also serve to make the work of software redistribution managers more efficient.

## 1.2 Dissertation Synopsis

This dissertation is based on the development of a framework specifically tailored to the GAP ecosystem, aimed at generating data structures that can be processed for analysis and visualisation pertaining to its packages, redistribution, testing procedures and the community of developers. By sourcing the necessary data from GitHub, a popular platform for code collaboration and version control, the framework holds the potential to optimise how GAP packages are shared, and support the values inherent to FOSS. More specifically, the product of the dissertation is a framework suitable for reproducing various analytical indicators and measurements using primarily Jupyter Notebook and Python, generating insight on the development, management and distribution of GAP through investigating its packages.

## 1.3 Dissertation Objectives

In the completion of this project, a set of primary, secondary and tertiary objectives were defined to measure the success of the framework. They serve as an indicator to assess whether the final product is providing what was intended to be delivered. The later sections of the project report will assess the degree to which each objective was achieved.

I. **Primary:** Allow a user to collect data, such as repository metadata, commit history, issue data, pull requests, and user information, on GAP packages from GitHub.

II. **Primary:** Allow a user to customise the data they want to extract to some extent. This will be done by leaving room for the user to in some way specify what information they want to extract, changing parameters that are used to call the data from GitHub, so that the overview they end up with corresponds to their specific needs.

III. **Primary:** Allow a user to get an overview of the current situation for GAP packages from GitHub, by exploring metrics such as the number of packages matching a set of given criteria, for the collected data.

IV. **Primary:** Allow a user to monitor packages redistributed with GAP, for purposes such as assisting a release manager or providing useful insight to the developer community. Metrics in this sense could include, for example, new releases or packages close to release.

V. **Primary:** Allow a user to explore any problems for GAP packages, related to metrics such as issues and pull requests, in a convenient way.

VI. **Secondary:** Create some useful insight and overview of the GAP package community. Metrics that could be interesting to explore in this fashion include actions, trends and behaviour of contributors, ranging from code authors to issue submitters and commenters.

VII. **Secondary:** Establish some processes for data cleaning and pre-processing, through measures such as accounting for call limitations, missing values and removing irrelevant entries.

VIII. **Secondary:** Allow a user to obtain some insight and understanding as to the history of GAP packages, their dynamics and the corresponding community, through historical trends and collaboration patterns.

IX. **Tertiary:** Ensure the code can be accessed and executed by all individuals to the greatest extent possible, using tools such as Docker containerising or the Binder software.

# Chapter 2: Context Survey

This chapter is dedicated to reviewing, analysing and summarising existing literature in the field, pertaining to the objectives of the project. It will point to relevant background information, important development considerations and related scholarly research, while critically investigating where current research differ from the approach taken in this project.

## 2.1 Project Scope

The distribution of FOSS through collaborative platforms such as GitHub, supported by development tools like Python and Jupyter Notebook, has been a subject of discussion by several researchers and data scientists. Scholarly literature emphasises the centrality of computational analysis in these workflows, advocating its use for the distribution of high-quality, user-friendly software [40].

Despite the well-documented necessity of software redistribution for FOSS existence, illustrated by Michlmayr et al. [33] and Perens [35], current literature reveals a deficiency in models, management guidelines and tools to assist release managers in the redistribution process [22]. This is particularly true for larger open-sourced software distribution processes [30]. In relation to certain distribution methods, LaBelle and Wallingford [26] attribute this deficiency to a typically high number of packages, often in the thousands, with interconnected dependencies. FOSS is sustained by entire communities, and while this has its advantages, a distribution style of this kind also comes with challenges related to release management [43] and investment [56]. Efforts to address these challenges can leverage platforms like Jupyter Notebook, underpinned by technologies such as Python, Binder, Docker and Git, to form analytical frameworks for FOSS redistribution [41]. Regarding data management for analysis, existing literature also explores optimal strategies for data sharing, management and documentation [9].

As such, this literature review aims to identify gaps in research and supporting frameworks for release managers redistributing FOSS, specifically focusing on GAP, a computational algebra system and programming language [48]. More specifically, this evaluation of previously published material will substantiate the need for the analytical framework proposed by this project, underpinning development decisions with corroborative studies and scholarly works. It will offer pertinent literary insights on project dependencies, development communities and critically discuss the dichotomy of code and data, as well as collaboration patterns and trends, from a human-centric perspective. A foundational understanding of computational themes and concepts is assumed.

## 2.2 Reproducibility in Computations

There is wide scientific acceptance for how software, by definition, is intended to "provide an automation solution to user problems and reduce the human effects" to optimise productivity [44]. Reproducible computational analysis is fundamental to better understand possibilities and limitations for a given software product, as indicated by extant literature on the subject [14]. In defining the essence of reproducibility, Rule et al. [40] states that reproducibility is about allowing for results to be recreated by other users. Chue Hong [9] underlines the distinction between replicability and reproducibility, suggesting that sustained usability and the ability to recreate with different data sets is what distinguishes reproducibility from replicability. This is of particular importance to this project, considering the aim of developing a framework clearly separating code and data, accompanied by tuneable parameters.

Sandve et al. [41] points to reproducibility as a cornerstone not only for analytical frameworks, but for the field of scientific research as a whole. This necessitates its accessibility and comprehensibility. While the establishment of a reproducible computational framework hinges on machine interpretability, existing literature asserts this claim through highlighting the importance of accompanying user documentation [40].

Honouring this principle of reproducibility not only enhances user interaction but, as Chue Hong [9] articulates, it is also crucial for the researcher in such continual work. Thus, with reproducibility in mind, the framework developed in this study should strive for reproducibility of analytical findings.

## 2.3 Distribution and Redistribution

In sharing software, distribution refers to the general provision of software packages to intended users, while redistribution is about the subsequent delivery after the software package has been collected from the original source [28]. The redistribution of FOSS such as GAP has been discussed by several scientific sources, pointing to the many associated considerations, all under the administration of a release manager. Previously published work underscores the critical role of ensuring continuous distribution of software packages, highlighting the benefits of knowledge sharing inherent in freely available software [42]. The process of redistributing FOSS in the context of GAP is exemplified in Figure 2.1, showing how it is based on volunteer contributions.

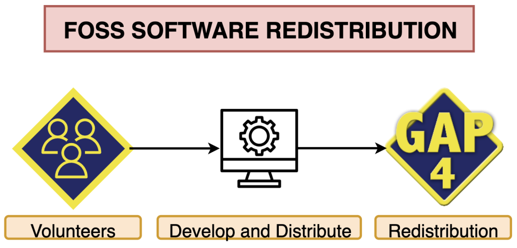


FIGURE 2.1: The redistribution of FOSS such as GAP is based on the individual package distributions of volunteers.

In existing literary definitions, it can be found that release management aims to "deploy release packages into target environments" [27] as intended, within the designated time frame [21]. Of particular interest to this project, current literature by Van Der Hoek and Wolf [52] discusses how the process of managing software releases is becoming increasingly complex, through changing components, dependencies and development environment. The complexity of release management imposes a significant workload, with associated risks, on the release manager [20]. Consequently, the demand for release management tools to navigate this evolving landscape underscores the value of the framework developed in this project [52].

Moreover, studies on the redistribution of FOSS like GAP highlight community-related challenges. Michlmayr [32] elucidates that volunteer development can result in inactive developers, thereby complicating management and distribution processes. Similarly, West and Gallagher [56] examines the challenges associated with numerous contributors, emphasising the difficulty in maintaining a comprehensive overview. They also highlight potential issues in redistributing FOSS, such as interdependencies, incomplete key functionality, testing, community communication and problem tracking [33].

## 2.4 Release Management

The complexities release managers must navigate during the redistribution process in computational fields have been covered by numerous scholars. Koch [23] posits that maintaining FOSS necessitates quality management, given that package delivery and management in this context pose unique requirements compared to traditional commercial software development, particularly considering the community of contributing programmers. Given the added complexity in coordination and component control, release management in redistributing software systems like GAP presents unique challenges. Treinen and Zacchiroli [51] highlight the need for seamless updates and automated dependency resolution as integral aspects of this process.

Functioning as a release manager require a certain skill set, Michlmayr et al. [33] points out. The authors' research asserts that capabilities of a release manager should include discipline, judgement and attention to detail. Hence, a tool easing the many responsibilities of a release manager for a system such as GAP will be of value in delivering quality software. Automated tools, like the framework developed in this project, can significantly reduce the cost of release management [19].

## 2.5 Framework Development Process

Academic literature suggests various strategies for developing a reproducible data analytics framework, such as the one underpinning this project, emphasising its iterative nature and evolving focus. Rule et al. [40] recommends starting by forming the project structure through storytelling, process documentation and defining section. Subsequently, the authors recommend developing the necessary code, before sharing the code and its documentation. This notion is also highlighted by Granger and Perez [14], emphasising the importance of storytelling and meaningful explanations of functionality.

Prior published material also puts emphasise on documenting all significant variables, versions and relationships [41]. Rule et al. [40] emphasise the importance of recording dependencies during development to ensure reproducibility. They further elaborate on the utility of pipelines for generalising processes, enabling varied data and parameters to yield different outcomes within a consistent framework. For this reason, it is recommended to "place key variable declarations, especially those that will be changed when doing a new analysis, at the top of the notebook" [40]. This principle is particularly pertinent to this project, given the critical importance of distinguishing between code and data.

## 2.6 Jupyter Notebook

Given its widespread use in computational research and extensive coverage in literature, the selection of Jupyter Notebook as the foundation for this project's analytical program is well-justified. This is exemplified by existing literature pointing to how "millions of users and tens of thousands of organisations use Jupyter on a daily basis" [14].

Another aspect of consideration for deciding on the structure of the project framework includes how Jupyter Notebook is frequently employed in interdisciplinary settings [14]. This could be significant for a project focusing on GAP, as the computer system is commonly "used in research and teaching for studying groups and their representations, rings, vector spaces, algebras, combinatorial structures and more" [48]. This interdisciplinary consideration also underpins the principles of CoP (Communities of Practice), where communities dedicated to work on the same topics rely on sharing knowledge easily and efficiently [14]. Rule et al. [40] emphasise the utility of Jupyter Notebook in facilitating literate programming, where code delivery and user documentation coexist. This integrated structure, they argue, enhances tool usability by providing credibility, context and guidance through accompanying descriptive text.

Moreover, Granger and Perez [14] points out that there are three main dimensions of Jupyter Notebook, concerned with the principles of computing interactivity, creating computational narratives and the notion that Jupyter Notebook extends beyond being mere software. This paper's findings underscores the necessity of crafting quality, user-friendly code for the project, leveraging the comprehensive functionality of Jupyter Notebook while incorporating storytelling. Various methods for presenting the computational framework's contents are suggested by established academic sources. For presenting the contents of the computational framework discussed in this paper, established academic sources provides various methods to consider. Rule et al. [40] and Granger and Perez [14] point to NBviewer for sharing and visualising Jupyter Notebook. However, it is of importance to note that there are also other alternatives, such as GitHub and Google Colaboratory [14]. The optimal choice of presentation for this project must consider what would be the logical choice for a focus on separating code and data, which is principal to the framework.

Of further significance to this project, existing literature points to how version control with Jupyter Notebook can be challenging because code differences are stored in JSON (JavaScript Object Notation) format [40]. This presentation is less readable than its counterparts from a human perspective, as differences are compared based on the JSON files instead of notebook GUI (Graphical User Interface). This distinction is significant for the framework development process, as it prompt the search for alternatives to streamline version tracking and comparison.

## 2.7 Technological Infrastructure

In developing software products, existing literature offers guidance on the selection of complementary technological components, particularly from the perspective of reproducible analytics. Jupyter Notebook supports Python, a high-level programming language often used for its simplicity and readability as detailed by Vanderplas [54], and several assertions have been made by previously published works on creating Python code to optimise analytical outputs. For instance, utilising ipy-widgets, as suggested by Rule et al. [40], enables the change of parameters without necessitating code modifications. This functionality, facilitating analysis of various data points, could be beneficial for this project. Sandve et al. [41] similarly emphasise the importance of avoiding steps requiring manual code manipulation.

As to facilitate the concept of reproducibility, existing literature highlights the importance of using technologies that makes it easy for others to run the code. Ragan-Kelley et al. [37] emphasises how combining Python with other programs can further enhance such functionality. To incorporate such functionality, the authors point to software like Binder [4] - a platform for sharing computing environments based on Jupyter Notebook - and Docker [31], a program for deployment and scaling of applications within containerised environments. Other researchers also discuss this notion, pointing to how the use of such tools can facilitate environment sharing without mandating local software or dependency installations, while enable Python workflow automation [14]. The implementation of some tools like this could be beneficial for the computational analysis aspect of this project, as they enhance the reliability and accessibility of the code.

Highlighting the relevance of programs like Binder and Docker for this project, Sandve et al. [41] points to the necessity for users to reproduce results reliably through such a framework, even when faced with unexpected challenges. Such software additions would empower both the researcher and other users to reproduce the results on different devices and environments, a significant advantage for reproducible data analytics. Thus, using Python for coding in Jupyter Notebook, while fully leveraging its extensive functionality and synergising it with technologies like Binder and Docker, will be pivotal considerations in the development of the framework delineated in this paper.

## 2.8 Git and GitHub

GitHub, a renowned collaborative platform for code development, is highlighted in existing literature for its uses, utilities and several advantages. The data for this project, including repositories and associated metadata, is stored on GitHub and served through the GitHub API (Application Programming Interface) [10]. As the most popular platform for social coding, GitHub is based on the distributed development tool Git, commonly used for version control [17]. Despite its foundation on Git, GitHub offers a range of unique features for enhancing collaboration and promote social interactions within the development community [10]. Given its widespread use in software development, GitHub frequently serves as a primary data source for computational studies [7]. While earlier FOSS communities mainly had a focus on project management, supported by functionality such as issue tracking, version control and release management, scholars in the computing field highlight how GitHub rapidly gained popularity due to its user-centric approach [57].

Considering how version control, testing and a clear overview is critical to release management, as pointed out by Lahtela and Jäntti [27], version control tools are important means to releasing open-source software seamlessly. Efficient overview tools for release managers could benefit the GitHub community, as studies by Hata et al. [16] points to announcements and questions related to future release plans for software being among the most frequent discussions on GitHub. With the rising popularity of GitHub, there is also an increasing interest to analyse the social activities from the community [29]. With this scholarly insight in mind, it could be beneficial this project to further investigate the community behind GAP packages, as to provide some analytical insight for the community of programmers. The literary findings reinforces the utility of basing the analysis of GAP packages on GitHub for this project. They also propose the possibility of incorporating framework elements that make it possible to better interact with, cater to and understand the needs of the GitHub community.

## 2.9 Data Management

Numerous perspectives are provided by existing literature on how code and data should be handled, organised and shared in working with the development of reproducible analytics frameworks. The sharing of such code and data, as a core element to FOSS distribution, contributes to the sustained progress of the computing field through knowledge sharing [11]. In one instance, Sandve et al. [41] argues that the researcher should keep records of all data and results produced. While the overall concept of traceability and documentation is relevant to this project, it also has a distinct focus on the separation of code and data, which must be considered from a data collection perspective. On the contrary, Peng [34] points out that regardless of whether the data is made available, simply making code publicly accessible in an informative way can be powerful by itself. Alternative encouragement is given by Rule et al. [40], where the authors argue that researchers should aim to make at least parts of their data set available to potential users.

The notion that not all data must be shared, and that code by itself can be valuable, is the attitude adopted for the development process behind this project. Not only is this considered as the appropriate level to uphold from a usability perspective; an alternative approach of keeping and sharing records of all data produced through the framework would conflict with ethical commitments made for the project. At the outset of this endeavour, the researcher guaranteed that the products to be delivered would not directly share any identifiable information, but rather provide the tools for a user to generate this data independently. Data that would fall under this category includes GitHub usernames, GAP package names and records of similar nature. Consequently, the main points made by the authors in relations to distributing data and recording results ring true, with some important distinctions. Concepts like providing descriptions in the notebook to aid users in data processing applies to the project [40]. However, the research will maintain privacy by not sharing any identifiable information, for instance, through undertakings such as clearing Jupyter Notebook outputs before committing files to Git version control.

Of similar concern, the present work of field authors advocate for storing raw data behind figures, tables and plots used in research articles [9]. This notion will only be applicable to certain parts of the project, as it does not aim to be in possession of the complete raw data set for all instances of such illustrations. In the project's outset, the researcher determined that all GitHub usernames will be hashed and thus anonymised upon retrieval, so that the researcher will not have access to this data. Nonetheless, with the exception of data on GitHub usernames, this project will store the data referred to in plots and figures, including GAP packages where pseudonyms are provided when discussing the findings. On this notable differentiation of data, not sufficiently distinguished and elaborated on in existing scholarly literature, the project aims to keep records of most results produced, with the exception of those that fall under the category of identifiable data related to persons and packages. Analogously, field research claims that statements made in writing should be backed up by pointing to the specific results they represent [41]. While this might be beneficial for parts of this project, it does nonetheless aim for a separation of code and data, where the product provided is a means of generating some results, rather than the results themselves. As such, this would not necessarily be the case for all reproducible computational research and analysis, which should be recognised to a larger extent in existing literature.

## 2.10 Context Summary

As debated and assessed based on existing literature, there are a number of considerations to be made in the setting of developing a framework for reproducible data analytics based on data extraction from an online platform. Previously published research offer several outlooks on formatting, storytelling and data management when developing such structures. Reviewing a diverse selection of studies on the matter was helpful in forming opinions on an ideal design for the framework, particularly from the perspective of data management. Examining published works in the field provides a clear understanding of what components must be weighted in the overall project scope, while offering particular insight on the reproducibility, redistribution and release management functionality of the framework. This is essential insight in implementing the appropriate Jupyter Notebook infrastructure, and understanding Git and GitHub variables that may impact data extraction and handling.

# Chapter 3: Requirements Specification

This chapter will examine the requirements that were defined for developing the framework, along with the justification for their adoption. Through closely investigating potential users for the framework, their needs and the required data structure for analysis and visualisation yielded some core framework dependencies, which will be illustrated in this chapter.

## 3.1 Use Case

The overarching goal for the framework was to create a product that solves the user needs, as is the case for most software development processes. Potential users in this instance would be individuals involved in the development, management and redistribution of GAP, with a particularly aim at redistribution managers. Such distributions are defined as collections of packages, intended to be bundled and maintained in a coherent manner [12]. Subsequently, redistribution is concerned with release management of distributions, through the process of planning and supervising the deployment of software [21]. With this in mind, several considerations have to be made in the redistribution of open-sourced software, from licensing to source code integrity; all aspects that must be accounted for by a release manager [35]. Working with redistribution therefore requires the oversight and assessment of numerous variables, making it a complex process. This is especially true for FOSS release and redistribution [33]. To gain a better understanding of the software requirements based on the intended users, a use case diagram was developed, found in Figure 3.1.

Having a clear understanding of the users prior to initiating the development process enabled a more efficient design and implementation process. Delegating time to research existing literature, and drafting requirements based on this new-found knowledge of the audience, aided in the decision-making process of development because it clarified the understanding of needs.

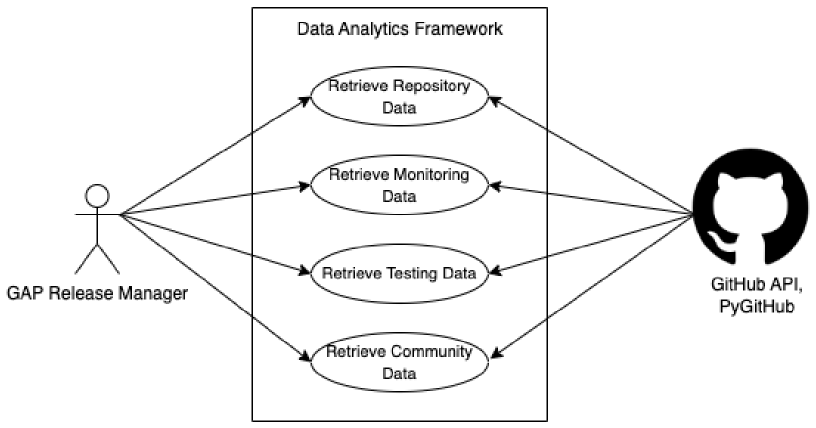


FIGURE 3.1: Example usage of the framework, where the intended user is a redistribution manager of GAP.

## 3.2 System Requirements

In formulating the requirements for the framework, the main focus was on the objectives referenced in §1.3. Accredited field literature was taken into consideration, emphasising how the focus of a framework like this should be to facilitate redistribution with the highest level of functionality and quality possible, considering resource limitations [22]. Moreover, of key importance, the framework had to accommodate to the nature and structure of GAP [48], for which it was to be designed.

As such, with GitHub as the basis for data extraction, the framework must consider what data will yield meaningful analysis. Based on studies describing the main areas of interest for such analysis [10], four categories for data extraction were defined. For each category, a set of data points were then selected. Illustrating what data would be required for each category, and how it should be formatted, was an essential step in the requirements formulation. The minimum data required for each category can be found in Table 3.1.

There are a number of concerns that must be considered in a release management process of sub-projects, which in this case would be the individual GAP packages. Weighing these aspects, including approval, testing and quality control, highlights how the system must incorporate broad functionality to be of value in redistribution management [20].

## 3.3 Ethical Considerations

The development of a framework intended to allow users to extract data based on GitHub usernames and repositories with package names is accompanied by a number of ethical requirements. As such, ethics approval was sought from the University of St Andrews

| Data Group | Properties | Structure | Format |
|---|---|---|---|
| Repositories | Age,<br>Total Releases,<br>Latest Release,<br>Last Activity Time,<br>Last Activity Type,<br>Total Pull Requests,<br>Open Pull Requests,<br>Closed Pull Requests | Per Repository | JSON |
| Monitoring | Different Versions,<br>Scheduled for the Next Release,<br>Last Release and Scheduled for Next | Applicable Repositories | JSON |
| Testing | Number of test files,<br>Total lines of test file code,<br>Versions of GAP tested on,<br>Required version of GAP | Per Repository | JSON |
| Community | Authors,<br>Submitters,<br>Author Repository Count,<br>Inactive Contributors,<br>Interactions | Applicable Repositories | JSON |

Table 3.1: The minimum data required from each analysis category, accompanied by the intended structure and format for storage.

School of Computer Science Ethics Committee, to ensure all ethical aspects had been thoroughly considered and accounted for, a claim validated in the form of an approved ethics application. The approval can be in Appendix A.

Requirements of ethical nature had to be considered and incorporated in the framework at the same level as other requirements. Detailed in the application behind the granted approval, ethical considerations for this project are primarily related to the processing of personal, though not sensitive, data. Despite the data being open-sourced and publicly available on GitHub, the researcher carefully considered the ethical implications of providing a framework that points to individual authors and named GAP packages. In doing so, it was determined that usernames extracted through the framework should be hashed upon retrieval, in the interest of providing anonymity to the individuals represented by the usernames. With a program structure like this, the true value of the usernames are never accessible, not even to the researcher. Furthermore, the project adheres to the principles of separating code and data, allowing users to run the code to collect data on individual GAP packages without providing any of this data. It will first be made available upon framework execution. Consequently, the names of GAP packages are pseudonymised in all printed components of the project, such as this report.

This approach aligns with the project goal of equipping users with the tools necessary to extract and interpret data, rather than supplying it directly. It aims to offer an application for accessing noteworthy package variables, trends and patterns, without directly provide users with the data.

## 3.4 Data Handling

The framework's core functionality revolves around data: its extraction, processing, analysis, visualisation and even exclusion, in cases where sharing is not intended. The project incorporates a number of facets where data is accessed and distributed, and it was therefore necessary to obtain a clear understanding of the data distribution flow, to incorporate the appropriate requirements. As such, a dataflow diagram was created, and it can be found in Figure 3.2.

A better understanding of what the project data structure looked like eased the process of creating the framework. Given the numerous components, diverse requirements had to be considered. Continuing on the requirements formulation from an ethical perspective, all notebook outputs generated during development were cleared before committing the code to the GitHub repository, demonstrating responsible data handling. This was done in accordance with guarantees made in the ethical application, and to ensure compliance with the commitments that the researcher will not share any identifiable information generated through the framework. Moreover, raw data extracted by the researcher was stored on their personal computer for the duration of the development process, and backup was performed through Apple's iCloud service. Only the researcher had access to this data. In completing the development of the framework, data stored by the researcher is to be destroyed to circumvent any ethical concerns related to the continual storage of such data. Thus, the implications of data extraction, processing and distribution had to be considered in the process of drafting requirements.

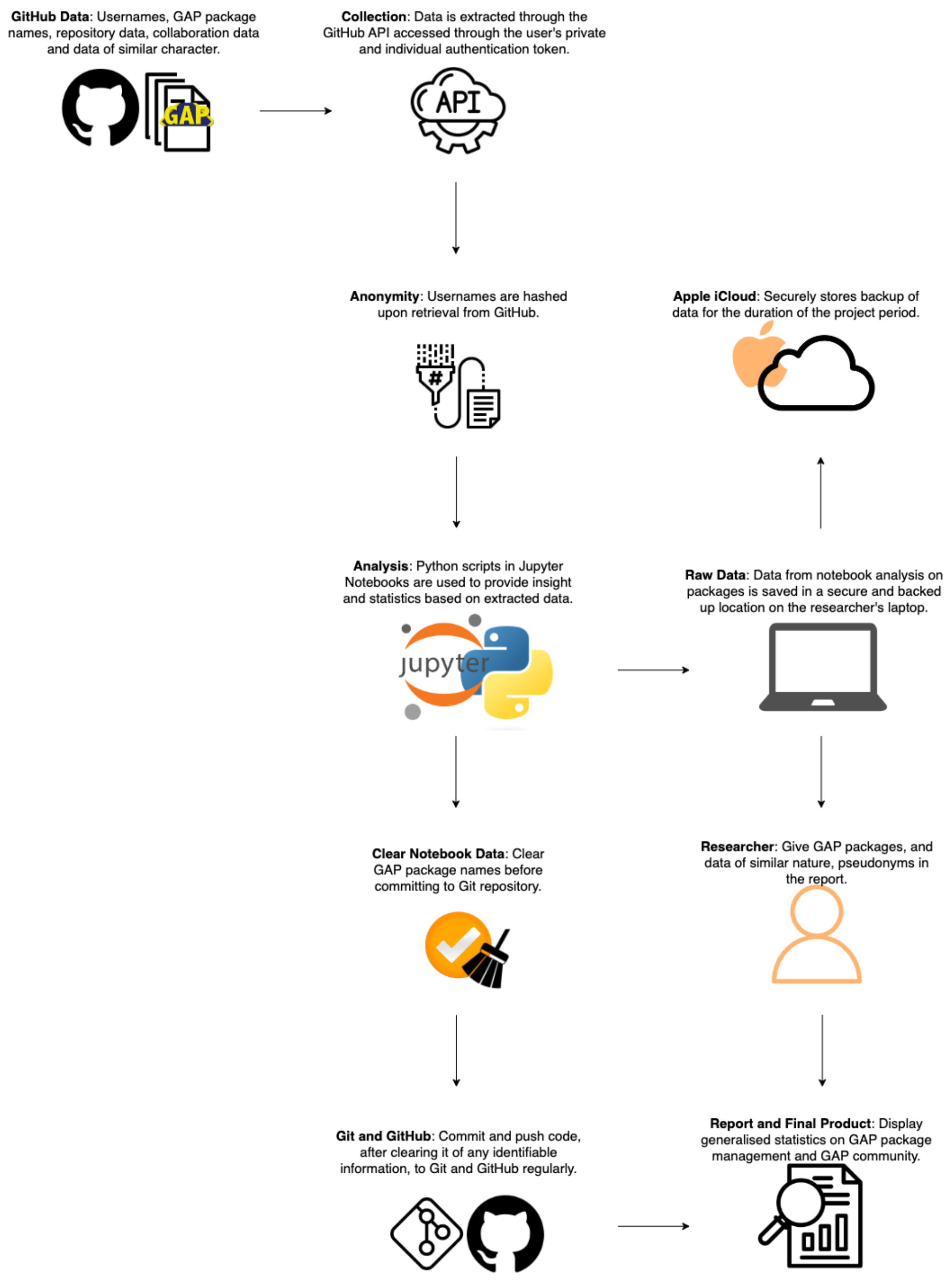

**GitHub Data**: Usernames, GAP package names, repository data, collaboration data and data of similar character.
GAP
**Collection**: Data is extracted through the GitHub API accessed through the user's private and individual authentication token.
API
**Anonymity**: Usernames are hashed upon retrieval from GitHub.
**Apple iCloud**: Securely stores backup of data for the duration of the project period.
**Analysis**: Python scripts in Jupyter Notebooks are used to provide insight and statistics based on extracted data.
jupyter
**Raw Data**: Data from notebook analysis on packages is saved in a secure and backed up location on the researcher's laptop.
**Clear Notebook Data**: Clear GAP package names before committing to Git repository.
**Researcher**: Give GAP packages, and data of similar nature, pseudonyms in the report.
**Git and GitHub**: Commit and push code, after clearing it of any identifiable information, to Git and GitHub regularly.
**Report and Final Product**: Display generalised statistics on GAP package management and GAP community.

FIGURE 3.2: The complete flow of data, depicting the numerous components of the framework and how they all relate to each other from a data handling perspective.

# Chapter 4: Software Engineering Process

This chapter will examine the process of creating the framework from an engineering perspective, and detail the considerations made. It will describe the Agile development approach taken, the reasoning behind its adoption and how the approach was implemented practically.

## 4.1 An Agile Approach

The choice of engineering process in software development could have developmental ramifications, best avoided through meticulous deliberations in selecting what strategy would best fits the situation. Though this project was completed independently by the researcher, several software development principles were studied, evaluated and compared, resulting in the adoption of selected principles from the Agile development framework. Rooted in four core values, the Agile Manifesto emphasises "individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; responding to change over following a plan" [3]. With some adaptation, and a focus on the overarching Agile mindset that prioritises learning, consideration and maintaining a flexible approach, the principles were incorporated in the framework development.

The intended audience for the framework, and how they would use it, was prioritised over the predetermined structure defined for the project, to stay open to changes that could improve the user experience. The application of Agile principles to a framework intended for FOSS is particularly fitting, as there are several similarities in their main principles [23]. Additionally, while notes were maintained throughout the process, the software itself was the main focus, fostering a mindset where documentation is intended to be supporting material, not a prerequisite to run the software. The third Agile principle was incorporated through being attentive to potential issues that arose during the process, instead of insistently staying with the initial ideas. Even though the framework was not developed in collaboration with any specific end-users, the target audience was kept in mind throughout the development phase, through evaluating various aspects of the release management process [22]. Finally, and perhaps of greatest importance to this project, adapting to problems and addressing changing perspectives of what would be beneficial to end-users was favoured over following the original plan mindlessly.

Reflecting on these core values from the Agile framework early on, along with revising the twelve principles accompanying the values [3], largely shaped the development process and how situations were handled along the way. Based on this paradigm, implementation jobs were broken down into smaller tasks, which were incrementally added and tested, and the approach taken at any given time was regularly evaluated to ensure the optimal strategy was applied. Through setting short-term objectives, and prioritising work accordingly, the development process remained flexible and efficient.

## 4.2 Incremental Implementation

Beyond an overall Agile development approach, describing attitudes and guidelines, several tools were adopted specifically to support the software engineering process. In terms of accountability throughout the development process, the researcher met with their supervisor weekly to discuss developments since last time, and address tasks for the upcoming work period. Additionally, the implementation of version control, ensuring changes were tracked and previous versions could be restored in the event of new code breaking functionality, was done through Git. Despite not relying on the collaborative aspect of version control, its use to periodically capture the state of the code while tracking new developments, was fundamental throughout the process. The use of tools like Git eases the process of testing new functionality, by providing the option of resetting the project to a previous state where everything was working as intended in situations where implementation attempts fail. This was core in supporting the software engineering approach chosen, providing a timeline for completed work.

# Chapter 5: Framework Design

This chapter is dedicated to the detailed exploration of the design behind the framework. Elaboration on the program structure, specifically looking into software chosen, notebook structure, code functionality and flow of execution, the design sections aims to explain and justify the approaches taken, and how they in union tie the framework together.

## 5.1 Overall Project Structure

The fundamental ideas behind the design of the framework are concerned with accessibility, readability and user-friendliness. In evaluating overall designs and file structures, various setups were assessed, and several options were tested before deciding on a final project scheme. The framework is composed of several Python and Jupyter Notebook files. As mentioned in §3.2, the framework is set to analyse four distinct areas of concern, namely repositories, redistribution monitoring, package testing and the contributor community. Additionally, it holds one notebook for data analysis and one notebook for data visualisation. There is also a folder for collected data, and various supporting files for containerisation and environmental variables in such instances. This design emphasises published research stating that release management tools should entail descriptive information, and that location transparency must be incorporated [52]. Assessing different design options, the code to fetch and analyse data gradually evolved in building the framework. The final framework structure is based on the notion of separating files for data extraction, data analysis and data visualisation, while keeping manual data manipulation steps to a minimum [41].

In designing the layout for the Jupyter Notebook files, the principles of not repeating code and organising notebooks in a structured manner were emphasised [41]. This is reflected in how the commit history of the project gradually moves repeated code into functions, defines constants and minimises the cost of GitHub API calls required. Through implementations such as utils.py and config.py files, the design allows for providing functionality code shared by several notebooks. These files are primarily concerned with user authentication through the GitHub token, as well as other functions that must be available and accessible to all notebooks. Additionally, a data_constants.py file was created to host constant variables and make them available to all notebooks. A design like this encapsulates the research aims of avoiding redundant code by moving out commonalities and making them accessible on a need basis [40].

Other noteworthy design choices of relevance to the project as a whole includes the use of Docstrings and function annotations. The framework heavily relies on functions to provide instructions on what data is to be extracted and how it should be interpreted. Consequently, it was imperative to design framework functions where the purposes and return values were clear. Whereas Docstrings detail the function's purpose, parameters and return values, function annotations provided type hints. The Google Python Style Guide [13] was followed for the implementation of Docstrings, fostering concise, verbose and structured annotations. An illustrated application of Docstrings and function annotations is shown in Figure 5.1.

```python
def get_repository_age(repo: Repository) -> tuple:
    """Get the age for a respository, measured in days.

    Args:
        repo (Repository): The GitHub repository.

    Returns:
        tuple: The repository name and the age of the repository
↪   measured in days.
    """
    age = (datetime.now().date() - repo.created_at.date())
    return repo.name, age.days
```

Figure 5.1: The Docstring for get_repository_age describes what operation it performs, its arguments and return value. Function annotation says that the repository is of type GitHub Repository, and the return value is of the type tuple.

## 5.2 Data Retrieval Design

GitHub is a popular source for providing raw data to analyse open-sourced software, due to its ease of access based on the API [10]. From a data extraction perspective, initial file versions placed all retrieval code in one file, based on the idea that only a single script would have to be executed to retrieve data. Later, as the code developed and the project implications were better understood, the decision was made to change the structure. To clearly indicate this separation of concerns, providing structured and easy-to-use Jupyter Notebook environments following industry best practice advice [40], four individual notebooks were generated to extract data for each category. A structure like this also enables users to recollect data selectively, instead of having to recollect all data if they wish to reload some data file.

Simultaneously, the files are prefixed with a numeric value to represent the order in which they are intended to be executed, no special circumstances taken into account. The design of this naming convention was done to enhance readability and user-friendliness, as to clearly communicate the intended order of execution. This choice was based on recommended practices, and to facilitate storytelling [14]. Moreover, this formatting of the pipeline allowed for restricting the length of files, provided a clear visual separation of areas of concern and aided in the modularisation of code [40]. Further considering the data collection design, the aim was to have all files generated through data retrieval ignored by Git, while displaying the collected_data folder to clearly communicate where data would be stored. As such, the design incorporates a README file in the folder, to ensure it was not empty, and then instructs Git to ignore all files in the folder, except from the README file. This design component also allows for specifications regarding the data that will populate the folder, such as how users are not intended to modify these files directly.

The data retrieval process is largely shaped by the GitHub API limitations. Through deployment of the Python sleep function, the framework allows the user to begin the process of data extraction, and if they run out of calls in the process, the program will automatically pause while waiting for the call rate to be renewed. When it is, the program will resume its flow of execution. This implementation required several attempts characterised by trial and error, before a final working version that would not crash the program upon an empty API call limit was discovered. The code for the function is included in Figure 5.2.

## 5.3 Novel Monitoring Features

With high industry competition and the access to more advanced tools, software today evolves rapidly. This creates the need for high-quality monitoring tools [44]. In designing monitoring functionality, much time was spent drafting, testing and revising potential solutions. The majority of the functionality in the notebook is based on the gap-system "PackageDistro" repository on GitHub, where the design idea was to output what the next GAP release would look like by extracting data on its current state. This analysis was largely based on repository metadata files, which are files containing fixed details regarding the package [51]. As a result, the notebook produces results on what packages have an updated version since the last GAP release, packages scheduled for the next GAP release and packages scheduled for the next release which were also in the last GAP release.

```python
def wait_until_reset(reset_time: int) -> None:

    # Convert UNIX timestamp to a datetime object
    reset_datetime = datetime.fromtimestamp(reset_time)

    # Calculate the time difference between current time and reset time
    current_time = datetime.now()
    time_difference = reset_datetime - current_time

    # Sleep only if the reset time is in the future
    if time_difference.total_seconds() > 0:
        sleep_time = time_difference.total_seconds()
        sleep_time_minutes = sleep_time / 60
        print(f"Reached GitHub API rate limit. Sleeping for
         ↪  {sleep_time_minutes} minutes until the limit resets.")
        time.sleep(sleep_time)
```

Figure 5.2: The wait_until_reset function takes the reset time provided by GitHub, calculates the time difference between the current time and provided renew time, and pauses the program execution for that time interval.

The notebook logic is based on the conclusion that all pull requests merged into the main branch of the repository is guaranteed to be in the next GAP release, as they can be regarded as approved changes. Subsequently, the current main branch version would be the version in the next GAP release at the time in which the code is ran. The notebook also incorporates functionality based on the conclusion that open pull requests might be included in the next GAP release, but this is not guaranteed as they have not yet been merged. A flowchart representing the approach can be found in Figure 5.3.

Determining what code would yield the desired functionality was one of the more challenging aspects of the implementation, but once it was accomplished, the analytical outputs provide valuable information for a redistribution manager.

## 5.4 Novel Testing Features

The trustworthiness and thorough documentation of package dependencies, such as version compatibility, are crucial in ensuring the redistribution of open-source software [26]. Additionally, industry research points to a lack of or insufficient testing as a core challenge in release management [27]. In designing functionality to analyse individual package tests and compatibility information, several attempts were made with the aims of creating code that would account for different formatting in repository version testing

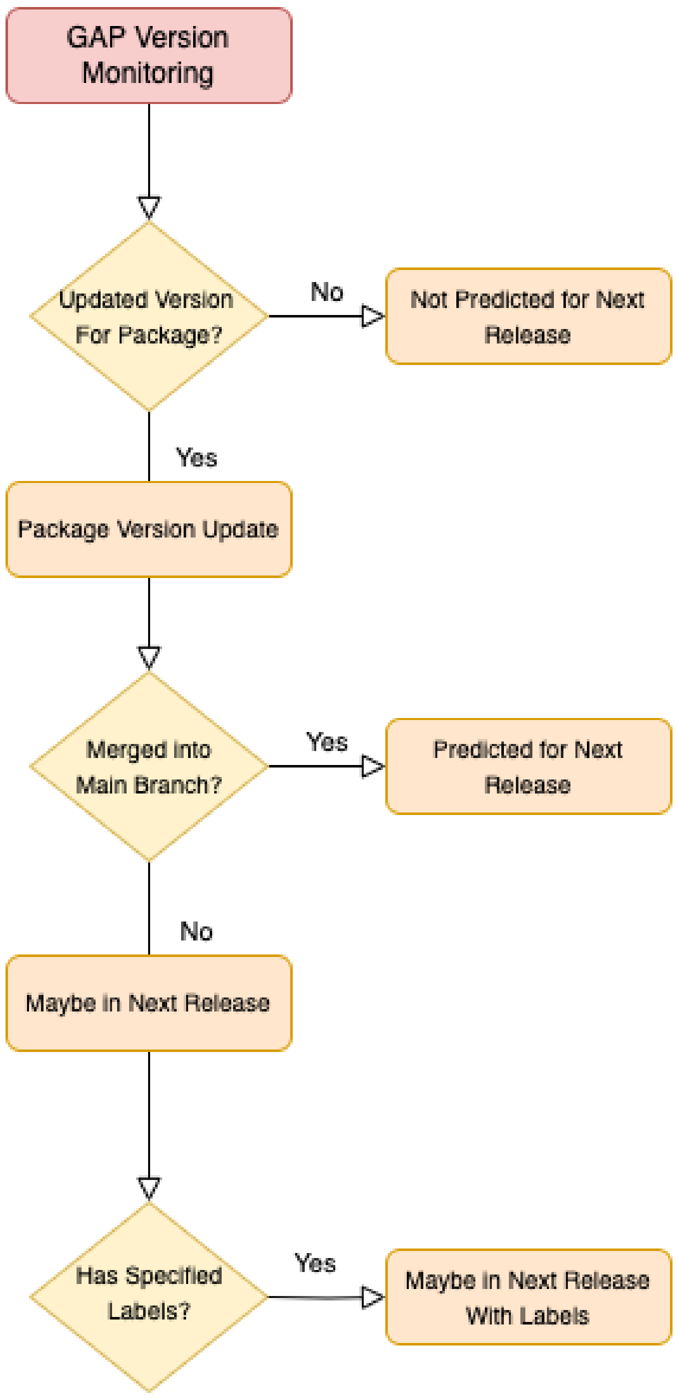


FIGURE 5.3: The workflow checks for new versions merged into the PackageDistro main branch, new versions under open pull requests and new versions under open pull requests with labels "Automatic PR", "New Package" or "Update Package".

files. The structure of the notebook is based on the acknowledgement that a GAP package, and its corresponding repository, will have one GAP version listed as the version required to guarantee package compatibility, which is the version listed in the repository PackageInfo.g file. This version will be referred to as the required version. Additionally, package administrators can choose to deploy CI (Continuous Integration) through

GitHub Actions, found in the CI.yml in each repository. This file states all the different GAP versions a package has been tested on. These versions will be referred to as the tested versions. Finally, a package repository can have a test directory with test files, used to confirm that functionality works as expected. Based on this knowledge, the aims of the testing notebook was to analyse the versions listed in the various files, their compatibility and any contradictory information.

In comparing the required version to the tested versions, complications and incorrect outputs emerged in the first code iterations. Upon further investigation, this was the result of varying formatting in how repository authors listed the version information. For example, some required versions had greater than or equal to `(>=)` symbols in front of the version, while other repositories did not have any comparison operators preceding the specified version, only an equal to `(=)` sign specifying exact version. Additionally, some few packages did not have any tested versions, inferring that the package had not been tested for compatibility on existing GAP versions. As a solution to the problem, regular expression patterns were implemented. With the final implementation, an inclusive pattern to capture all variations of version specifications was incorporated through a searching pattern specified as a raw string, looking for a string literal `stable-` followed by a capture group matching a version number of one or more digits, a dot `(.)` and then one or more other digits. Using this search pattern allows the program to account for different variations in the formatting of version numbers. The implementation of the search pattern can be found in Figure 5.4.

```
if pkginfo_file:
        pkginfo_content =
         ↪  pkginfo_file.decoded_content.decode("utf-8")
        version_pattern = r'GAP\s+:=\s+"[^"]*?([\d.]+)"'
        version_match = re.search(version_pattern,
         ↪  pkginfo_content)
        if version_match:
            gap_version = version_match.group(1)
            pkg_tested_version.append((repo_name, gap_version))
```

FIGURE 5.4: Taking advantage of the powerful data processing tools in Python, a version search patterns accounting for differences in version formatting was incorporated.

Further considerations had to be made in determining the design for processing test directories, to compute the number of test files and lines of test code for each repository. With formatting variations similar to the ones for the CI files, individual repository administrators store test files in various ways. While some repositories have all test files stored in one directory, others have multiple folders where some included additional sub-folders. Creating code accounting for these differences was accomplished through the implementation of a recursive function to process directories. When called, the function will first check if the item for which it is being called is a directory or a test file. If the item is a test file, the function will increment the count of test files by one, and count the lines of code for that given file. On the contrary, if the item for which it is being called is a directory, the function will call itself to process the sub-directories, continuing the search for test files and lines of test code. The code described can be found in Figure 5.5.

```
def process_tst_directory(repo: Repository, directory_path: str,
↪   tst_file_info: dict) -> tuple:
    contents = repo.get_contents(directory_path)
    num_tst_files = 0
    total_lines = 0

    for item in contents:
        if item.type == "file" and item.name.endswith(".tst"):
            tst_file_content = requests.get(item.download_url).text
            lines = tst_file_content.splitlines()
            num_tst_files += 1
            total_lines += len(lines)

        elif item.type == "dir":
            subdirectory_path = f"{directory_path}/{item.name}"
            subdir_num_tst_files, subdir_total_lines =
            ↪   process_tst_directory(repo, subdirectory_path,
            ↪   tst_file_info)
            num_tst_files += subdir_num_tst_files
            total_lines += subdir_total_lines

    return num_tst_files, total_lines
```

FIGURE 5.5: The function allows for processing test files and lines of test code in repositories with differences in how files and directories are structured.

## 5.5 Novel Community Features

For FOSS, the very core of the system is composed by the individual contributions of community developers [43]. Understanding how they work is therefore important to a redistribution manager. The notebook for community data retrieval is concerned with extracting data on the community of GAP collaborators on GitHub; a popular platform both for developers and researcher aiming to analyse the community of developers [7]. In computing functionality to analyse community interactions, several factors had to be considered.

The primary design concerns were related to what information should be extracted. Early stages of the code attempted to extract all unique authors, unique submitters and unique commenters for a given repository. While this provided a relatively complete data set, with several possibilities for analysis and visualisation, the time and resources to load the data outweighed its benefits. Due to its sheer volume of data, it was also challenging to make any sensible visualisation with the data, and the analytical value did not justify the added strain on the program. The situation exemplifies a frequent occurrence in the design and development phase of the framework, where the cost of extracted data had to be considered in relations to the analytical value it provided, to find some sensible middle-ground. In this evaluation process, speed and ease of running program was given emphasis.

There are numerous ways in which community members can collaborate. The approach chosen had to allow for some sensible analysis, visualisation and conclusion to be extracted. As such, the notebook incorporates functionality to examine how repository authors and issue submitters collaborate, through outputs pairing them for individual repositories. Finding an appropriate format for storing this information resulted in the use of a dictionary, where the author's hash value is the key and the hash values of users who have submitted issues to any of their repositories are the values.

Initially, the functionality of this computation would store multiple hash values for the same issue submitter as a way of indicating the collaboration between the users on multiple occasions, but later revisions concluded that it was sufficient to know that the two users had interacted. The number of interactions did not add sufficient value to be included in the output, as it negatively impacted the readability and user-friendliness of the program.

## 5.6 Analysing the Data

With data appropriately extracted, processed and exported, the design process shifted to combining the outputs for analysis. For concerns related to versions, compatibility and consistency, the code design is primarily concerned with comparing version numbers in the search for inconsistency or in other ways noteworthy findings. As highlighted in §5.4, there are several ways in which a version can be specified for a given package. Having extracted versions, these challenges resurfaced in comparing them. While some packages have a single decimal point, such as "4.9", other package versions entail two decimal points typified by "4.10" and yet others have more decimal portions, as seen in "4.11.0". To account for this varying syntax, the packaging [8] library was used to access the version module, and the parse function. Incorporating these tools aided in the handling of version numbers, through parsing version strings and converting them into Version objects, to determine ordering and ease the comparison process. A portion of this functionality is exemplified in 5.6.

```
if ci_versions and package_version:
    if not all(version.parse(ci) >= version.parse(package_version[0])
    ↪  for ci in ci_versions):
        packages_with_mismatch.append(package)
```

FIGURE 5.6: The comparison of version numbers with different formatting was made possible through employing various Python tools.

Additionally, functionality was added to check whether the tested versions were lower than the required version, for each individual package. The outputs from the evaluation are important in terms of being conscious of how the CI file is used and how often it is run. From an ethical and sustainable perspective, every computational process has an environmental cost, and determining where tests are being conducted for versions that are lower than the required version is an important step in efforts to reduce unnecessary runs of the CI environment.

## 5.7 Visualising the Data

The Python libraries implemented for visualisation purposes allow the framework to display a wide array of functionality, in different ways [54]. When it came turn to design the visualisation functionality, the question of how code could be implemented to fulfil the objectives of the project arose. The visualisation aspect of the project did not only aim to solve user problems, but it also focused on the core Jupyter Notebook principle of storytelling [14]. As such, various bar charts, histograms, pie charts and graphs were designed. An example, analysing repository data, can be found in Figure 5.7. The actual framework output, showing repository names, have been replaced with pseudonyms as to uphold commitments made in the ethical evaluation.

Obtaining a good understanding of the community of contributors on GitHub is of importance to the redistribution process, as the two are strongly related [57]. However, the long hash values representing GitHub usernames reduced the readability of certain plots and graphs, due to their sheer length. To account for this issue, a label generator function was implemented in the visualisation notebook, to allow for alternative representations with better readability. Given the hashing function's purpose of ensuring anonymity for GAP community members, substituting their representational value does

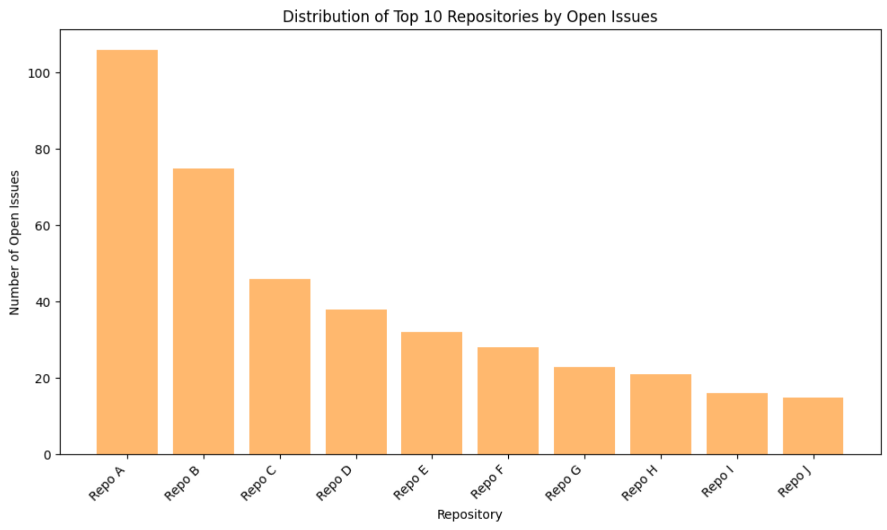


FIGURE 5.7: Distribution of repositories by number of open issues (the actual repository names have been pseudonymised due to ethical considerations).

not affect the analytical value of the program. As such, replacing them with shorter, more concise identifiers improves the program's usability.

The visualisation section of the framework has also been designed to provide outputs of a more statistical nature. An example can be found in Figure 5.8. While the outputs of these visualisations are not likely to directly affect the redistribution process, they are important in providing a better understanding of the state of GAP and its packages.

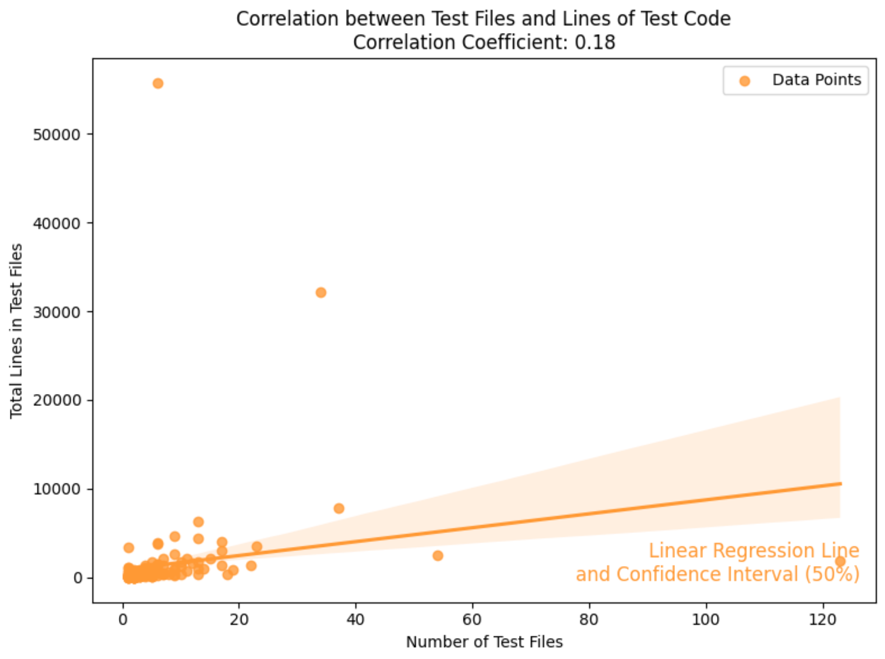


FIGURE 5.8: The x-coordinates show the number of test files for a package, while the y-coordinate show total lines of file code. The linear regression indicates a slight positive correlation between the two.

# Chapter 6: Implementation

The contents of this chapter will focus on exploring facets of implementation, including the flow of implementation, how functionality testing was conducted and other noteworthy implementation decisions.

## 6.1 Software Implementations

The main structure of the framework is based on Python [53] and Jupyter Notebook [2]. For accessing data, the Python library PyGithub [50] was incorporated to ease the process of interacting with the GitHub API.

Additionally, several libraries and packages were installed to accomplish the functionality aimed delivered by the framework. Beautiful Soup is a tool used for web scraping [39]. Incorporating this Python library allowed for data extraction from websites, to provide some statistics on GAP packages hosted in locations other than GitHub. Matplotlib [18] is an interactive visualisation tool, which coupled well with the Python library seaborn [55], which is based on the framework of matplotlib, to create more statistical data visualisation. Further supporting the visualisation process, NetworkX [15] is a comprehensive Python package focused on depicting complex networks, which was useful in providing visuals for the GAP community. Other Python libraries employed includes numpy [1], which is a versatile package for scientific programming, and packaging [8], providing functionality related to computations on version numbers. Pandas [49] is a Python tool commonly used for data analysis and manipulation, and through combining its capabilities with the ydata-profiling [6], the two libraries allowed for nicely formatted analytical outputs. Additionally, nbformat [46] was implemented to manipulate the contents of the analysis and visualisation notebooks, before nbconvert [45] was employed to create HTML (HyperText Markup Language) files only including the outputs of the files, to facilitate readability. Other libraries incorporated for the framework includes dateutil [47], an extension of the existing Python datetime module, and requests [38], facilitating an easy and elegant way of handling HTTP (Hypertext Transfer Protocol) requests.

## 6.2 Testing Code and Functionality

Testing the code after implementation led to several discoveries, which in some cases prompted changes to the framework. For example, attempting to analyse repository events led to the finding that only GitHub activities that took place within the last 3 months are available through the API. As such, this data came with time limitations that had to be accounted for [17]. Additionally, in implementing the framework, several structures were tested for graphs. From a visualisation perspective, some relationships were more difficult to depict due to their complexity. Especially for NetworkX graphs, numerous attempts were made before finding an implementation that displayed some sensible relationship. Examples of discarded attempts can be found in Appendix B. These many rounds of testing, characterised by trial and error, were fundamental in finding good, working solutions for the framework. For this particular example, the final graph resulting from the testing can be found in Figure 6.1.

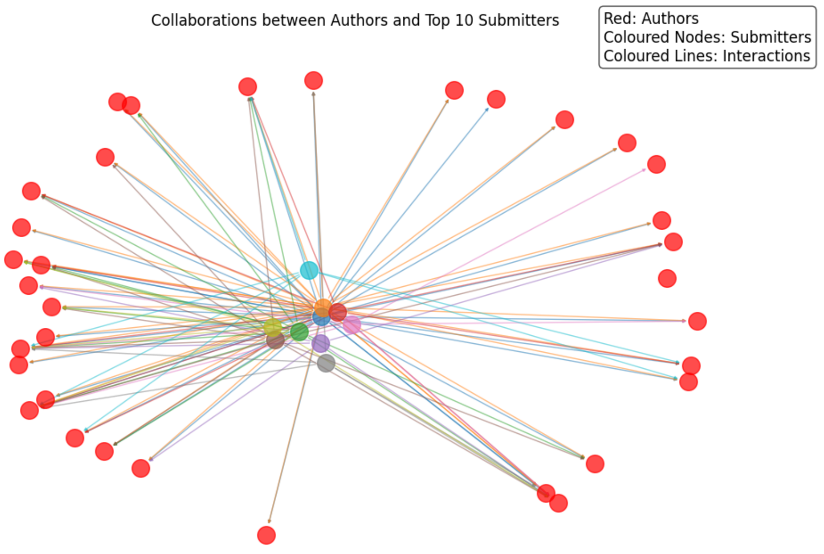

Collaborations between Authors and Top 10 Submitters
Red: Authors
Coloured Nodes: Submitters
Coloured Lines: Interactions

FIGURE 6.1: An iterative process of testing various solutions for the NetworkX graph eventually yielded some sensible and interpretable output.

This pattern of testing a variety of solutions, and correcting the approach taken in the next code iteration, was a common theme throughout the implementation phase of the project. Testing code is particularly important for analytical frameworks of this kind, as they form the foundation for decisions and further work. As such, much time was devoted to testing implementations, both for quality and in order to find the best solution possible.

## 6.3 Implementation Aids and Protocols

Reconsidering the project's ethical aspect during the implementation phase, initiatives were taken to ease the process of omitting notebook outputs from the Git history. While upholding this promise was initially done by consistently clearing out all notebook outputs before committing project updates, the pre-commit routine was later obsoleted through the discovery of nbstripout, a software tool functioning as a "git filter or pre-commit hook" [25]. The utility of nbstripout is displayed in Figure 6.2.

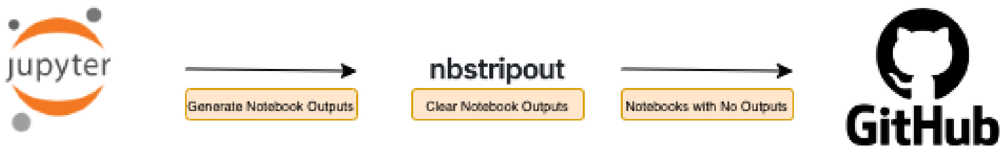


FIGURE 6.2: The employment of nbstripout eased the process of upholding the commitment made in the ethical application for the project of not sharing any outputs.

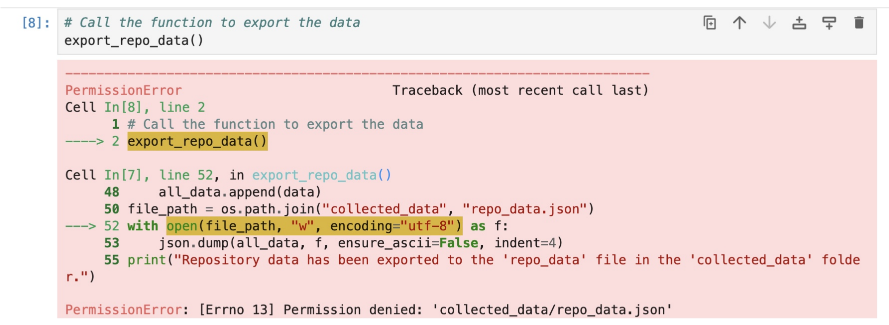
The version control aid found in nbstripout ensures that even if Jupyter Notebook outputs are mistakenly left uncleared, they will not be included in the Git log, as they will be automatically clear upon committing them. The decision of implementing nbstripout helped in prevented any unintentional publishing of outputs, easing the process of ethics compliance from an implementation perspective.

## 6.4 Reproducibility: Docker and Binder

From the project's onset, the researcher proposed the use of some industry software to create an alternative environment for accessing the framework, enhancing its accessibility and reproducibility. More specifically, Binder and Docker were mentioned, as research conducted at the time pointed to these environments as appropriate for this setting. However, as a common facet to software development, discoveries made later in the implementation phases somewhat altered this notion.

Upon finalising the project code and initiating the implementation of these tools, challenges related to Binder's primary use for public repositories were considered. MyBinder [4] allows users to turn any repository into an interactive, executable environment, but it is only available for public repositories. There are however solutions in place to allow for the application of Binder to private repositories, such as BinderHub, which is the technology on which MyBinder is based [5]. It has become a favoured tool for reproducibility in the data science community, with functionality for both technical and practical reproducibility [37]. Nonetheless, the final choice of a tool to ensure a consistent, independent and reproducible environment needed to balance user-friendliness with the need for manual configuration to accommodate alternatives for user-provided GitHub tokens. Considering the project's use case, implementing both Binder and Docker seemed excessive at this point. After evaluating the options, Docker was chosen as the preferred alternative to running the framework in a local Jupyter Notebook environment, over Binder.

Docker, a software providing containerised environments focused on isolation, reproducibility and easy deployment, was implemented after the Jupyter Notebook code was completed. Having added the Docker file and generated a Docker image to test the program in a containerised environment [31], some errors occurred when running the project in JupyterLab. The problems were related to users and permissions, as `\joyvan"` is the default user when running the containerised environment this way, and this user did not have the necessary permissions to create files when extracting data. A representation of what this permission denial looks like can be found in Figure 6.3.



```
[8]: # Call the function to export the data
     export_repo_data()
```

```
---------------------------------------------------------------------------
PermissionError                           Traceback (most recent call last)
Cell In[8], line 2
      1 # Call the function to export the data
----> 2 export_repo_data()

Cell In[7], line 52, in export_repo_data()
     48     all_data.append(data)
     50 file_path = os.path.join("collected_data", "repo_data.json")
---> 52 with open(file_path, "w", encoding="utf-8") as f:
     53     json.dump(all_data, f, ensure_ascii=False, indent=4)
     55 print("Repository data has been exported to the 'repo_data' file in the 'collected_data' folde
r.")

PermissionError: [Errno 13] Permission denied: 'collected_data/repo_data.json'
```

FIGURE 6.3: Permission complications that occurred when implementing Docker.

After several attempts to resolve the issue, the information necessary to create a working solution was found reading the documentation on common problems when employing Docker with Jupyter Notebook [36].

Considering how there is no home directory in the traditional sense when the framework is accessed in a non-local environment, the token must be supplied elsewhere. As such, the function to get the GitHub token from the user also had to be modified, to allow for its specification either through the home directory, or located as an environment variables in the project. The README file for the project was also updated accordingly. The final execution workflow for running the framework with Docker can be found in Figure 6.4.

The implementation of tools like Docker for the framework opens doors for its wider use, by supporting processes of scalability, portability and rapid deployment. The Docker framework is commonly used in software development, and making the framework available in this framework enables its wider use [31].

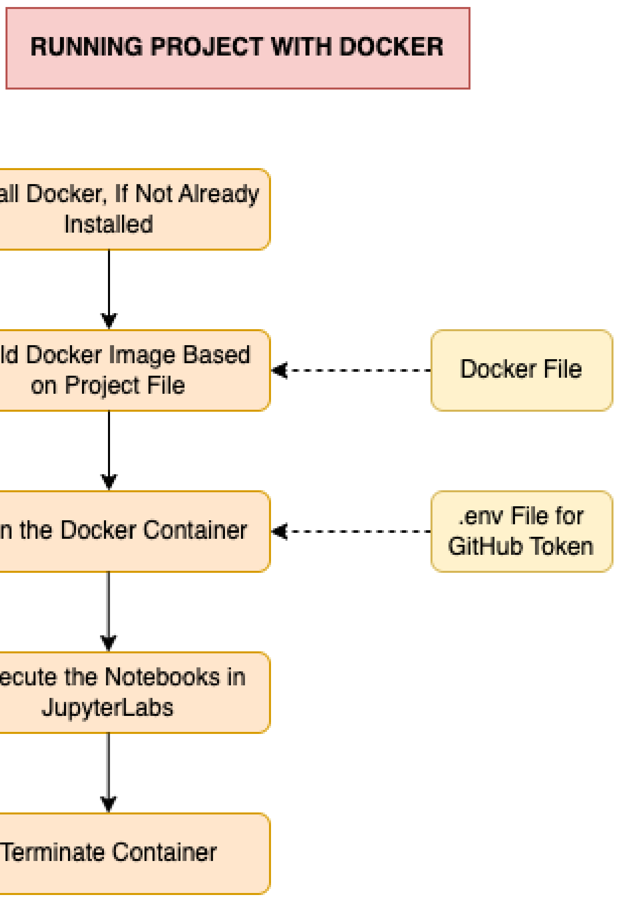

RUNNING PROJECT WITH DOCKER
Install Docker, If Not Already Installed
Build Docker Image Based on Project File
Docker File
Run the Docker Container
.env File for GitHub Token
Execute the Notebooks in JupyterLabs
Terminate Container

FIGURE 6.4: The framework can be executed through Docker to facilitate compatibility and efficient use of resources.

## 6.5 Facilitating Framework Usability

A key attribute of a robust framework is its ease of use, navigation and comprehension. This should be backed by clear documentation detailing the software's requirements. In finalising the implementation, the discovery of automatic requirements generation was found through pip-tools, specifically using pip-compile. First, the pip freeze command was used to generate a list of all packages installed in the virtual environment, along with their versions, in a file called requirements.in. Next, having quality controlled the file contents, the requirements.txt file was generated. The commands executed to perform this operation, compiling a requirements.txt file based on the requirements.in file, can be found in Figure 6.5.

```
$ pip freeze > requirements.in
$ pip-compile --output-file=requirements.txt requirements.
   in
```

FIGURE 6.5: The commands show how program requirements can be automatically generated based on installations in the virtual environment.

Creating the requirements.txt file in this way ensured no package dependencies were forgotten, as all packages installed for the virtual environment were included in the output.

Additionally, to enhance framework usability in the implementation phase, several alternatives were considered to make analytical outputs more readable. The options considered were pandas DataFrames, Rich DataFrame, the Python built-in Pretty Printer and the IPython display module. In choosing what option to move forward with, the program use case was emphasised. As a GAP redistribution manager might want to share the data, for example by copying it and including it in an e-mail, any formatting adding friction to such information sharing should be avoided.

The Rich DataFrames primarily comes with functionality for colours, highlighting and styling, which in this context would clutter the outputs without adding any considerable value. As such, this option was discarded. As most of the framework's data has a tabular structure, and thus best represented as a table consisting of rows and columns, the pandas DataFrame was chosen as the best option. This tool has functionality well-suited for manipulating and analysing tabular data.

Furthermore, considering any potential future implementations of analytical enhancements for the framework, such as data aggregation, merging and filtering, the pandas DataFrame is well-suited for these tasks, unlike Pretty Printer. The final choice was therefore to go with the pandas DataFrame for readability, and use the IPython display module in some instances where there were large outputs, to facilitate readability.

Finally, the decision was made to incorporate functionality for the framework to generate HTML files containing only the outputs of the data analysis and visualisation files. Despite implementing a design that separates the notebooks for data extraction from the files intended to interpreting analytical results, the outputs of the analysis and visualisation notebooks are still displayed in between code blocks necessary for computations. As such, incorporating functionality to output the HTML files provides framework users with the possibility of examining only the file outputs.

# Chapter 7: Evaluation

This chapter aims to critically evaluate the results of the development efforts for this project, as well as the quality of the framework created. It will review what has been accomplished in relation to the original objectives, while considering the use cases in comparison to existing alternatives.

## 7.1 Framework Outputs in Notebooks

Upon completing the project, there are a total of four Jupyter Notebook files for data extraction, three Python files for setup, configuration and data constants, a Dockerfile and other supporting files enabling a variety of framework features. After retrieving the data from GitHub, the data is processed and manipulated for analysis and visualisation that can be interpreted by framework users. The outputs provided for repository data in the analysis notebook can be found in Table 7.1, defining the area of study along with the data type for the output.

| **Repositories: Description of Output** | **Data Type** |
|---|---|
| Packages from GAP on GitHub | Text |
| The latest version of GAP | Text |
| GAP packages hosted elsewhere on GitHub | Text |
| Total Number of Repositories | Text |
| Total Number of Releases | Text |
| Number of Open Issues | Text |
| Number of Open Pull Requests | Text |
| Total Bug Count | Text |
| Total Enhancement Count | Text |
| Inactive Repositories | DataFrame |
| Repository Statistical Analysis | ProfileReport |

Table 7.1: Processing the data extracted from GAP repositories provides a variety of analytical outputs.

Additionally, the analytical part of the framework provides outputs related to the monitoring data extracted. An overview of these outputs can be found in Table 7.2, describing the areas of interest.

| Monitoring: Description of Output | Data Type |
|---|---|
| Updated package version on main branch compared to last GAP release | DataFrame |
| Packages with unmerged pull requests | DataFrame |
| Packages with unmerged pull requests, and some specified release related labels | DataFrame |

TABLE 7.2: Running the script for data analysis provides predictions on what the next GAP release might look like.

The execution of the script for data analysis also provides outputs for the testing data collected in the extraction process of the framework. These outputs are referenced in Table 7.3.

| Testing: Description of Output | Data Type |
|---|---|
| Repositories with files in test directories | Text |
| Number of test files for all packages | Text |
| Number of repositories with CI file | Text |
| Number of repositories with PackageInfo file | Text |
| Number of packages with tested versions but no required version | Text |
| Number of packages with required version but no tested versions | Text |
| Number of packages with both tested versions and required version | Text |
| Overview per individual package on test file count, tested versions, required version, whether the CI file has data and whether it has a PackageInfo file | DataFrame |
| Number of repositories with the latest version of GAP in their tested versions | Text |
| Packages where some tested versions are not greater than or equal to the required version | DataFrame |
| Packages with gaps in tested versions and the required version, or where required version is higher than some tested version | DataFrame |

TABLE 7.3: The testing data extracted through the framework provides a number of analytical outputs of relevance to redistribution.

Finally, the analytical component of the framework processes the data extracted on the GAP community, to provide relevant outputs. These can be found in Table 7.4, along with the type of output.

| **Community: Description of Output** | **Data Type** |
|---|---|
| Number of authors for all GAP packages | Text |
| Number of submitters for all GAP packages | Text |
| Number of authors who were also submitters for all GAP packages | Text |
| Number of inactive contributors for all GAP packages | Text |

TABLE 7.4: The framework outputs provided on the GAP community provides a deeper understanding of its composition, function and interactions.

Finally, the visualisation notebook in the framework is focused on creating plots, graphs and charts based on a combination of the extracted data. The outputs it produces can be found in Table 7.5, listed in chronological order based on there in the notebook the visualisations appear.

| **Visualisation: Description of Output** | **Data Type** |
|---|---|
| Distribution of repositories by age | Histogram |
| Distribution of repositories by total releases | Bar Chart |
| Distribution of top N repositories by open issues | Bar Chart |
| Distribution of pull requests, open vs. closed | Pie Chart |
| Next release predictions based on PackageDistro repository | Bar Chart |
| Number of GAP packages with tested versions and required version | Bar Chart |
| Correlation between test files and lines of test code | Scatter Plot |
| Number of authors, issue submitters and author-submitters | Bar Chart |
| Top N authors and their repository contributions | Bar Chart |
| Top N authors with most issue submitter interactions | Bar Chart |
| Number of contributors by year of first commit | Bar Chart |
| Interactions between authors and top N issue submitters | NetworkX Graph |

TABLE 7.5: In combining and manipulating the extracted data, the framework provides a wide array out visualisation outputs.

Combining, manipulating and interpreting data for analysis and visualisation had yielded a framework that produces a wide variety of outputs, to the benefit of potential users wanting to obtain a better understanding of the current state of GAP, as well as the community responsible for its continued development.

## 7.2 Assessing Dissertation Objectives

Some key objectives, referenced in §1.3, were defined at the project's outset. Concluding the framework implementation, these objectives were evaluated to assess their attainment. An overview of the appraisal, concluding on the extent to which the objectives were achieved, can be found in Table 7.6.

| **Category** | **Description** | **Achieved** |
|---|---|---|
| Primary | I. Data Collection from GitHub | Yes |
| Primary | II. Customise Data Extracted through Framework | Partially |
| Primary | III. Obtain Overview of Current GAP Situation | Yes |
| Primary | IV. Monitor Packages Redistributed on GitHub | Yes |
| Primary | V. Explore Issues Related to Packages | Yes |
| Secondary | VI. Provide Insight on Package Community | Yes |
| Secondary | VII. Process for Data Cleaning and Pre-Processing | Yes |
| Secondary | VIII. Facilitate Historical Trends and Patterns | Partially |
| Tertiary | IX. Reproducibility through Docker and/or Binder Tools | Partially |

Table 7.6: All objectives were reached with the exception of those for which unforeseen implications occurred, and alternative strategies were chosen.

In reviewing the objectives, it becomes evident that the final product succeeds in fully meeting most of the goals defined for its implementation. Most of the objectives were fully achieved, and a description of what specific implementations were made in their fulfillment can be found in Table 7.7. Some of the fulfilled objectives were completed in a number of instances, exemplified through objectives concerned with getting data or allowing for monitoring.

While the majority of the objectives were fully achieved, some objectives were only partially implemented. This was largely due to time constraints, because alternative solutions were prioritised or due to discoveries taking away from their implementation need. Details on the extent of implementation, along with comments on where components intended to be implemented are missing, can be found in Table 7.8. The final product is a reflection of the acknowledgement that it was acceptable for these objectives to only be partially fulfilled, and that development efforts could be prioritised towards other areas.

| Objective | Description of Achieved Objectives |
|---|---|
| I. | Data is automatically collected by running the scripts for data collection, and outputs are stored in a designated project folder. Collected data include metrics for individual repositories, repository distribution monitoring, package testing and the contributor community. |
| III. | A wide range of functionality has been implemented in the pursuit of this objective. The final product incorporates numerous metrics for the current situation of GAP packages, such as age, last event and number of issues for a given package, to highlight a selection. |
| IV. | Individuals operating the framework can access key redistribution monitoring metrics, primarily based on the GAP "PackageDistro" distribution repository on GitHub. This repository is currently used for releasing new GAP versions, so monitoring its contents will give an indication of what the next release will look like. Analysing discrepancies between the main branch version number and the latest released version indicates an imminent package update in the next GAP release. |
| V. | The framework incorporates functionality to access an overview of potential issues related to individual GAP packages. To cite a few examples, the program enables analysis identifying packages with differing test numbers, issues with specific problem-related labels and pull requests requiring further investigation. |
| VI. | Users can extract data on the total number of authors and issue submitters, and how these relate to one another. They are also able to see statistics on the rate of contribution, inactive contributors identified through some given threshold and the first contribution of authors, to give a few examples. |
| VII. | The framework accounts for missing and irrelevant variables by removing them entirely, or filtering them out through keywords for exclusion. Data cleaning and formatting are also part of the design, through accounting for variances in how input can be formatted, such as pattern seeking. |

Table 7.7: The specific implementations incorporated in the framework in fulfillment of the achieved project objectives.

## 7.3 Comparing Existing Solutions

While GAP offers a variety of tools to support package authors, including PackageManager, the Example package, ReleaseTools, GitHubPagesForGAP and Docker containers, their functionality differ from what is provided through this framework [24]. These solutions all contain functionality that would be useful to a redistribution manager, but compared to the framework developed for this project, they do not have the complete analytical functionality. Even though there are several benefits to be had from the use of tools such as the GAP PackageManager, existing literature in the field points to how

| Objective | Description of Partially Achieved Objectives |
|---|---|
| II. | Users are able to customise the data they want to analyse to some extent, illustrated by input variables specifying the number of users to include for visualisation plots. The original plan was however to incorporate such functionality to a larger extent, and in a way that did not require users to modify notebook code, but there was not enough time for this endeavour. |
| VIII. | The objective of allowing a user to analyse historical trends was incorporated through providing visual analysis on when authors made their first contribution. However, the original intent was to incorporate such functionality to a larger extent. |
| IX. | The objective of implementing tools for containerising, through the implementation of a setup to run the framework with Docker, was achieved for the framework. The project originally debated the inclusion of Binder, but due to the program's private nature and the limited benefits to be had, this was later on decided against. |

Table 7.8: Objectives that were only partially achieved, along with comments on where intended implementations are missing.

such alternatives are lacking in more specific functionality, especially as the distribution volume increases [30].

While GAP ReleaseTools - a GAP package intended to assist in the process of creating GAP releases by automating redistribution tasks - is undeniably useful, it does not provide detailed analysis in the way the framework created does. Simultaneously, the framework cannot perform any redistribution actions, nor did it ever intend to. Consequently, considering how the framework is superior at providing analytical and visual outputs for decision-making, applying it in the union with existing tools such as GAP ReleaseTools could be beneficial to a redistribution manager. This way, the framework could be in charge of providing the necessary data analysis, while other solutions could perform automated tasks related to redistribution actions. A comparison of the two redistribution tools, showing their differences in functionality, can be found in Figure 7.1.

## 7.4 Value and Framework Importance

In the field of computer science, published research points to open-sourced software and its developers as the future of global development [11]. Improving the accessibility of open-source software, a key driver of innovation and business competitiveness, holds societal significance [56]. Existing literature also points to how organising the group efforts that make up open-sources software is a complex process, due to the numerous developers working independently, before their contributions are packaged together in

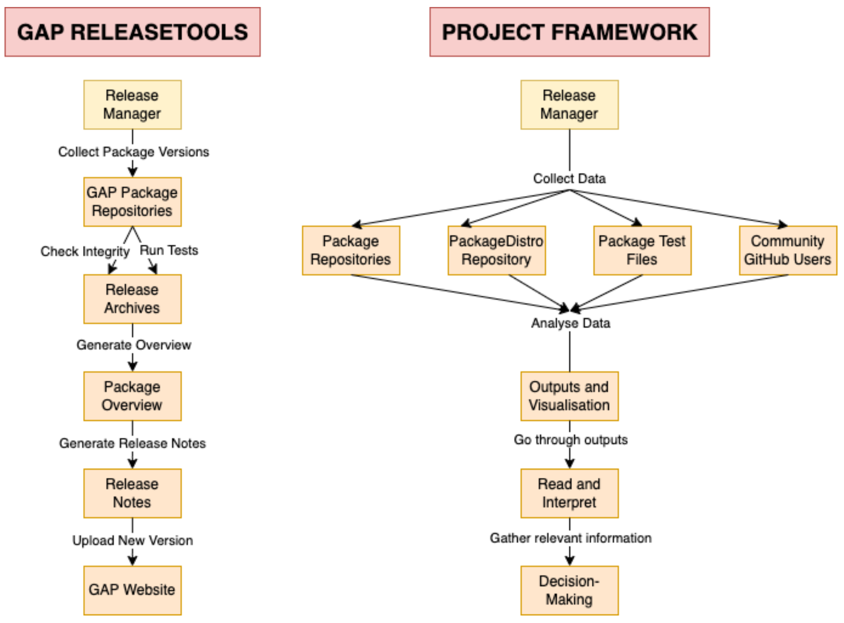


FIGURE 7.1: Comparison between the GAP release workflow and the framework created for this project, highlighting their differences in functionality.

one product, as is the case for GAP [28]. Any problems in the distributions should be dealt with promptly, and not postponed for later releases [19]. As such, a well-organised distribution pipeline is especially important for FOSS. Any inefficiencies threatens use, access and learning opportunity for the developer community [42]. Providing GAP release managers with a tool to facilitate the monitoring and analysis of the package community in the way this framework allows for is therefore an important step in the redistribution process, because it creates the foundation for sound decision-making.

Pointing to specific areas of concerns to highlight this concept, industry literature confirms the necessity of several metrics generated by the framework. For example, data given on inactive contributors is important to the redistribution process, as inactive contributors in distributed systems are "hard to quickly identify" [32]. Continuing on the spectrum of contributor outputs, the framework also enables the extraction of analysis regarding collaborator interactions: a subject debated and attempted understood by several acknowledged, existing publications [29]. Consequently, it is reasonable to claim that the analysis generated by the framework is of communal interest.

The numerous facets of new software releases are also of wider societal importance, as it is information frequently sought by the community of developers [16]. Simultaneously, there is a growing industry interest for reproducible research, and tools aiding in reproducibility in the field of computer science on a general basis [34]. This further attests to the value of the framework developed for this project.

# Chapter 8: Conclusion

This chapter will summarise the contents of the project, with an emphasis on the main achievements of the framework. It will highlight any shortcomings of the project, detailing what could be done to improve its functionality, followed by a review on possible future work that can be completed based on the project outcome thus far.

## 8.1 Dissertation Summary

The project report consists of eight chapters, where the introduction presents the project contents, followed by a literature review to examine existing publications in the field. Next, the requirements that were specified for the project are demonstrated, followed by the software engineering process applied during development, and the considerations made during the design phase, followed by the implementation phase. Finally, the project is evaluated and before conclusions are made in this final chapter.

The final version of the software product that is found in the analytical framework tailored for GAP analysis consists of four Jupyter Notebook files for data extraction, one for outputting analytical findings based on the data, and a final notebook for data visualisation. Both the analysis and visualisation notebook holds functionality to export the findings only to an HTML file. The notebooks are supported by various Python utility and configuration files, and the project directory also contains a pre-made Docker file to run the framework in a containerised environment.

The ultimate outcome of the project not only achieves most of the defined objectives; it also includes additional functionalities beyond the main goals. A framework for data extraction, analysis and visualisation was developed, to study the current state of GAP packages hosted on GitHub from a redistribution perspective. The report concludes that all objectives were fully achieved, with the exception of objectives II., VIII. and IX.. However, as emphasised in §7.2 during the framework evaluation, these three objectives are labeled as partially achieved either due to an original intention to implement their functionality to a greater extent, or because implementation discoveries reduced their initial purpose. Overall, the final product allows for the output of relevant computational analysis, separating code and data, and works as intended.

## 8.2 Dissertation Limitations

While the project for the most part delivers on the pre-defined objectives, it is still defined by certain shortcomings that could be improved upon. First and foremost, the objectives labeled as partially achieved could be further explored, to fill any gaps in functionality that is still desired implemented. This could include adding more options for customising data outputs in relations to objective II., and implementing code that outputs historic outputs to a greater extent in relations to objective VIII..

Another limitation of the framework includes how it for the most part only is applicable to GAP packages hosted on GitHub, despite there being packages hosted in other locations as well. Additionally, the usability of the framework could be strengthened by moving all data extraction functionality to one location, so that only one script would have to be executed to fetch the data. In doing so, a user would be able to run the code, let the program take its time in extracting the data, and come back knowing all data has been extracted and is now available for analysis and visualisation.

Furthermore, the execution speed is constrained by the GitHub API, though the design minimises the number of required calls and includes functionality to pause execution if calls run out. As the framework relies on extracting information from the Internet, it is also subject to limitations and constraints of application layer protocols such as HTTP. While these are common limitations for programs of this nature, and do not pose any obstacle that necessitates correction, it is still of importance for users to be aware that such restrictions exist in both using and further developing the framework.

Lastly, in discussing the limitations of the project, several measures have been implemented to ensure the privacy of the GAP community. Consequently, the framework does not provide users with any data on specific users. This could have been beneficial in terms of collaboration or further investigations based on the system outputs, but would have been in violation of the ethical commitments discussed in §3.3.

## 8.3 Potential Future Work

There are numerous potential extensions for the project that could be investigated in the future. As mentioned in §8.2, the framework mainly provides analytical outputs for GAP packages hosted on GitHub, and any future work could explore the inclusion of analysis for packages hosted elsewhere. Additionally, if the framework is made public in the future, it could also be interesting to investigate the incorporation of Binder to fulfill objective IX. to a greater extent, or do a new assessment on the utility of employing both Binder and Docker for the framework.

Moreover, additions can be made for the outputs in the project's analysis and visualisation files. The data gathered opens up numerous potential areas of investigation, but the framework created in this project prioritised only those found to be relevant due to time constraints. The possible outputs that can be generated based on the GAP community found on GitHub are many, and as the importance of providing such collaborator collectives with reliable analytical tools has been accentuated throughout this paper, adding to the framework functionality overall could be engaging future endeavours.

# Bibliography

[1] Abbasi, Hameer et al. *Array programming with NumPy.* Version 1.23.5. 2020. DOI: 10.1038/s41586-020-2649-2.

[2] Abdalla, Safia et al. “Jupyter Notebooks - a publishing format for reproducible computational workflows”. In: (2016). DOI: 10.3233/978-1-61499-649-1-87.

[3] Beck, Kent et al. *The Agile Manifesto.* 2001. URL: http://agilemanifesto.org/.

[4] Binder. *Turn a Git repo into a collection of interactive notebooks.* 2023. URL: https://mybinder.org/.

[5] BinderHub. “BinderHub - BinderHub documentation”. In: (2023). URL: https://binderhub.readthedocs.io/en/latest/.

[6] Brugman, Simon. *Pandas-profiling: Exploratory Data Analysis for Python.* Version 4.1.2. 2019. URL: https://github.com/pandas-profiling/pandas-profiling.

[7] Caldeira, João et al. “Software Development Analytics in Practice: A Systematic Literature Review”. In: *Archives of Computational Methods in Engineering* 30 (3 2023), pp. 2041–2080. ISSN: 1134-3060. DOI: 10.1007/s11831-022-09864-y.

[8] Cannon, Brett et al. *Packaging.* Version 23.1. 2023. URL: https://pypi.org/project/packaging/.

[9] Chue Hong, Neil. “Better Software, Better Research: Why reproducibility is important for your research”. In: (Aug. 2014). DOI: 10.6084/m9.figshare.1126304.v1.

[10] Cosentino, Valerio, Canovas Izquierdo, Javier L., and Cabot, Jordi. “A Systematic Mapping Study of Software Development With GitHub”. In: *IEEE Access* 5 (2017), pp. 7173–7192. ISSN: 2169-3536. DOI: 10.1109/access.2017.2682323.

[11] Dempsey, Bert J. et al. “Who is an open source software developer?” In: *Communications of the ACM* 45.2 (2002), pp. 67–72. ISSN: 0001-0782. DOI: 10.1145/503124.503125.

[12] Di Cosmo, Roberto, Zacchiroli, Stefano, and Trezentos, Paulo. “Package upgrades in FOSS distributions”. In: (2008). DOI: 10.1145/1490283.1490292.

[13] Google. “Google Python Style Guide”. In: (2023). URL: https://google.github.io/styleguide/pyguide.html.

[14] Granger, Brian E. and Perez, Fernando. “Jupyter: Thinking and Storytelling With Code and Data”. In: *Computing in Science & Engineering* 23.2 (2021), pp. 7–14. ISSN: 1521-9615. DOI: 10.1109/mcse.2021.3059263.

[15] Hagberg, Aric A., Schult, Daniel A., and Swart, Pieter J. *Exploring network structure, dynamics, and function using NetworkX.* Version 3.1. 2008. URL: https://networkx.org/.

[16] Hata, Hideaki et al. “GitHub Discussions: An exploratory study of early adoption”. In: *Empirical Software Engineering* 27.1 (2022). DOI: 10.1007/s10664-021-10058-6.

[17] Hu, Yan et al. “Influence analysis of Github repositories”. In: *SpringerPlus* 5.1 (2016). DOI: 10.1186/s40064-016-2897-7.

[18] Hunter, John D. *Matplotlib: A 2D graphics environment.* Version 3.6.3. 2007.

[19] Jansen, Slinger and Brinkkemper, Sjaak. “Ten Misconceptions about Product Software Release Management explained using Update Cost/Value Functions”. In: *2006 International Workshop on Software Product Management (IWSPM’06 - RE’06 Workshop)* (2006). DOI: 10.1109/iwspm.2006.8.

[20] Jansen, Slinger et al. “Integrated development and maintenance for the release, delivery, deployment, and customization of product software: a case study in mass-market ERP software”. In: *Journal of Software Maintenance and Evolution: Research and Practice* 18.2 (2006), pp. 133–151. ISSN: 1532-060X. DOI: 10.1002/smr.330.

[21] Jäntti, Marko and Järvinen, Julia. *Improving the Deployment of IT Service Management Processes: A Case Study.* Springer Berlin Heidelberg, 2011, pp. 37–48. ISBN: 978-3-642-22205-4. DOI: 10.1007/978-3-642-22206-1_4.

[22] Kajko-Mattsson, Mira and Fan, YuLong. “Outlining a Model of a Release Management Process”. In: *Trans. SDPS* 9 (2005), pp. 13–25.

[23] Koch, Stefan. *Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion.* Springer Berlin Heidelberg, 2004, pp. 85–93. ISBN: 978-3-540-24853-8. DOI: 10.1007/978-3-540-24853-8_10.

[24] Konovalov, Olexandr. “Open source software system for discrete computational mathematics”. In: *Conference of Research Software Engineers* (2019). URL: https://dx.doi.org/10.6084/m9.figshare.7581551.v1.

[25] Kynan. *nbstripout.* Version 0.6.0. 2022. URL: https://github.com/kynan/nbstripout.

[26] LaBelle, Nathan and Wallingford, Eugene. “Inter-package dependency networks in open-source software”. In: *Journal of Theoretical Computer Science* (2005). URL: `https://arxiv.org/pdf/cs/0411096.pdf`.

[27] Lahtela, Antti and Jäntti, Marko. “Challenges and problems in release management process: A case study”. In: *2011 IEEE 2nd International Conference on Software Engineering and Service Science* (2011). DOI: `10.1109/icsess.2011.5982242`.

[28] Laurent, Andrew M. St. “Understanding Open Source and Free Software Licensing”. In: (2004). URL: `https://people.debian.org/~dktrkranz/legal/Understanding%20Open%20Source%20and%20Free%20Software%20Licensing.pdf`.

[29] Lima, Antonio, Rossi, Luca, and Musolesi, Mirco. “Coding Together at Scale: GitHub as a Collaborative Social Network”. In: *arXiv pre-print server* (July 2014). DOI: `10.48550/arXiv.1407.2535`.

[30] Mancinelli, Fabio et al. “Managing the Complexity of Large Free and Open Source Package-based Software Distributions”. In: *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE’06)*. IEEE, 2006, pp. 199–208. ISBN: 0769525792. DOI: `10.1109/ase.2006.49`.

[31] Merkel, Dirk. *Docker: Lightweight Linux Containers for Consistent Development and Deployment.* 2014. URL: `https://link.springer.com/content/pdf/10.1007/978-0-387-72486-7.pdf#page=315`.

[32] Michlmayr, Martin. “Managing Volunteer Activity in Free Software Projects”. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference.* Boston, MA: USENIX Association, 2004, p. 39.

[33] Michlmayr, Martin, Hunt, Francis, and Probert, David. “Release Management in Free Software Projects: Practices and Problems”. In: *Open Source Development, Adoption and Innovation: IFIP Working Group 2.13 on Open Source Software* 234 (2007), pp. 295–300. URL: `https://link.springer.com/content/pdf/10.1007/978-0-387-72486-7.pdf#page=315`.

[34] Peng, Roger D. “Reproducible Research in Computational Science”. In: *Science* 334.6060 (2011), pp. 1226–1227. ISSN: 0036-8075. DOI: `10.1126/science.1213847`.

[35] Perens, Bruce. *Open Sources: Voices from the Open Source Revolution.* O’Reilly, 1999. URL: `https://www.oreilly.com/openbook/opensources/book/perens.html`.

[36] Project Jupyter. *Using the Jupyter Docker Stacks - Troubleshooting.* Jupyter Docker Stacks Documentation. 2023. URL: `https://jupyter-docker-stacks.readthedocs.io/en/latest/using/troubleshooting.html`.

[37] Ragan-Kelley, Benjamin et al. “Binder 2.0-Reproducible, Interactive, Sharable Environments for Science at Scale”. In: *Proceedings of the 17th Python in Science Conference*. 2018, pp. 113–120. URL: `https://pdfs.semanticscholar.org/c043/bef741a9616d1144e0205ac21ceae881485d.pdf`.

[38] Reitz, Kenneth. *requests*. Version 2.28.2. GitHub, 2023. URL: `https://github.com/psf/requests`.

[39] Richardson, Leonard. *Beautiful Soup Documentation*. Version 4.12.2. 2007. URL: `https://www.crummy.com/software/BeautifulSoup/`.

[40] Rule, Adam et al. “Ten Simple Rules for Writing and Sharing Computational Analyses in Jupyter Notebooks”. In: *PLOS Computational Biology* 15.7 (2019), e1007007. ISSN: 1553-7358. DOI: `10.1371/journal.pcbi.1007007`.

[41] Sandve, Geir Kjetil et al. “Ten Simple Rules for Reproducible Computational Research”. In: *PLOS Computational Biology* 9.10 (2013), e1003285. ISSN: 1553-7358. DOI: `10.1371/journal.pcbi.1003285`.

[42] Stallman, Richard M. *Why Software Should Be Free*. Free Software, Free Society: Selected Essays of Richard M. Stallman. GNU Press, 2002.

[43] Stamelos, Ioannis et al. “Code Quality Analysis in Open Source Software Development”. In: *Information Systems Journal* 12.1 (2002), pp. 43–60. ISSN: 1350-1917. DOI: `10.1046/j.1365-2575.2002.00117.x`.

[44] Tabassum, Mujahid and Mathew, Kuruvilla. “Software Evolution Analysis of Linux (Ubuntu) OS”. In: *2014 International Conference on Computational Science and Technology (ICCST)* (2014). DOI: `10.1109/iccst.2014.7045194`.

[45] Team, IPython Development. *Nbconvert*. Version 5.9.1. 2023. URL: `https://pypi.org/project/nbconvert/`.

[46] Team, IPython Development. *Nbformat*. Version 5.9.1. 2023. URL: `https://pypi.org/project/nbformat/`.

[47] The Dateutil Community. *Dateutil*. Version 2.8.2. 2023. URL: `https://github.com/dateutil/dateutil`.

[48] The GAP Group. *GAP - Groups, Algorithms, and Programming - A System for Computational Discrete Algebra*. Version 4.12.2. 2022. URL: `https://www.gap-system.org/`.

[49] The Pandas Development Team. *Pandas-dev/pandas: Pandas*. Version 1.5.3. 2020. URL: `https://doi.org/10.5281/zenodo.3509134`.

[50] The PyGithub Community. *PyGithub*. Version 1.58.2. 2023. URL: `https://github.com/PyGithub/PyGithub`.

[51] Treinen, Ralf and Zacchiroli, Stefano. “Solving Package Dependencies”. In: (Aug. 2008). URL: `https://www.mancoosi.org/papers/debconf8.pdf`.

[52] Van Der Hoek, André and Wolf, Alexander L. “Software Release Management for Component-based Software”. In: *Software: Practice and Experience* 33.1 (2003), pp. 77–98. ISSN: 0038-0644. DOI: `10.1002/spe.496`.

[53] Van Rossum, Guido and Drake, Fred L. *Python 3 Reference Manual.* Version 3.11.3. CreateSpace, 2009. ISBN: 1441412697.

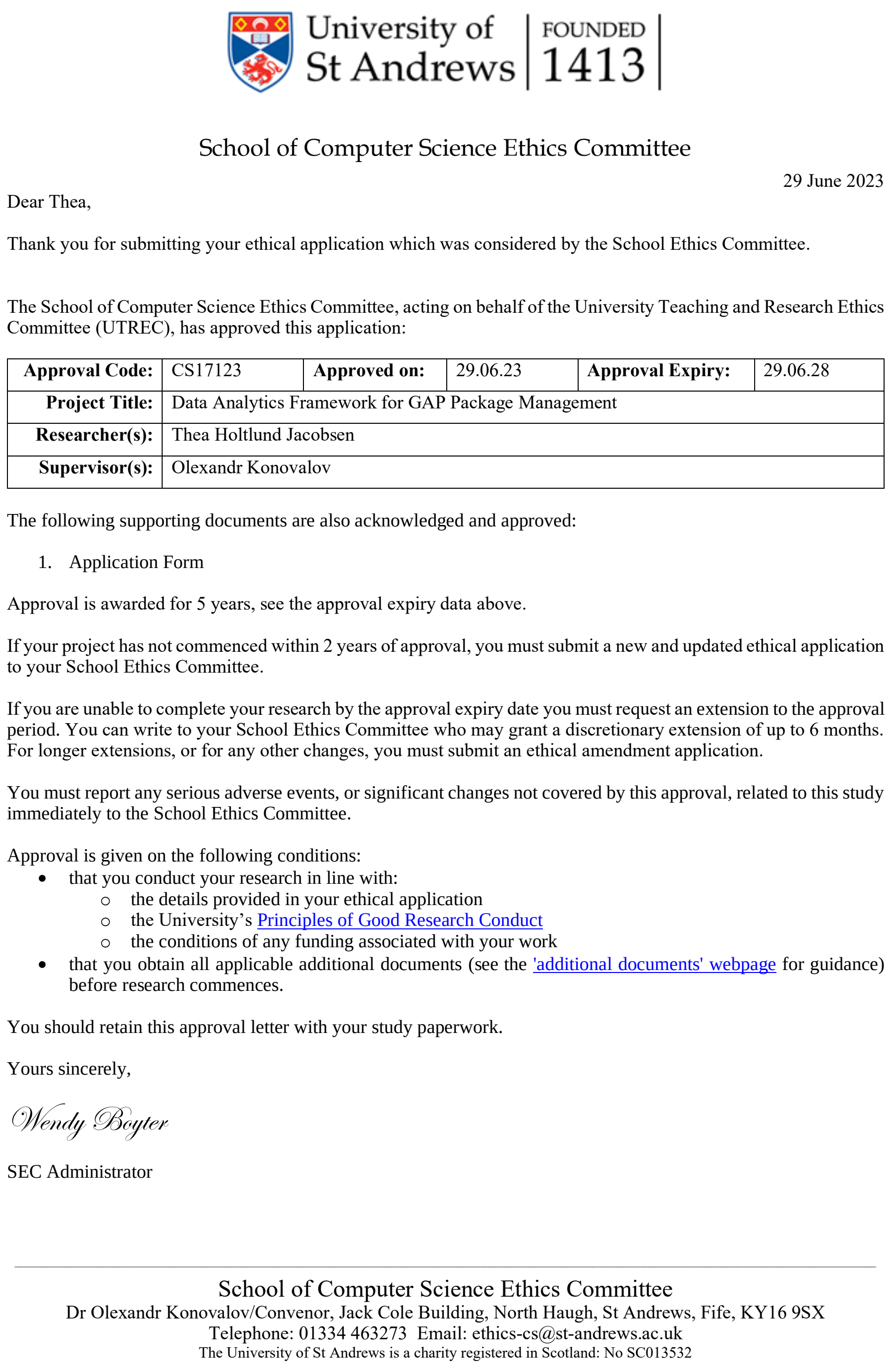
[54] Vanderplas, Jake. *Python Data Science Handbook.* 2016. URL: `https://jakevdp.github.io/PythonDataScienceHandbook/`.

[55] Waskom, Michael L. *Seaborn: Statistical Data Visualization.* Version 0.12.2. 2021. URL: `https://doi.org/10.21105/joss.03021`.

[56] West, Joel and Gallagher, Scott. “Challenges of Open Innovation: The Paradox of Firm Investment in Open-Source Software”. In: *R and D Management* 36.3 (2006), pp. 319–331. ISSN: 0033-6807. DOI: `10.1111/j.1467-9310.2006.00436.x`.

[57] Yu, Yue et al. “Exploring the Patterns of Social Behavior in GitHub”. In: *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies* (2014). DOI: `10.1145/2666539.2666571`.

# Ethical Approval Form



School of Computer Science Ethics Committee

29 June 2023

Dear Thea,

Thank you for submitting your ethical application which was considered by the School Ethics Committee.

The School of Computer Science Ethics Committee, acting on behalf of the University Teaching and Research Ethics Committee (UTREC), has approved this application:

| | | | | | |
|---|---|---|---|---|---|
| **Approval Code:** | CS17123 | **Approved on:** | 29.06.23 | **Approval Expiry:** | 29.06.28 |
| **Project Title:** | Data Analytics Framework for GAP Package Management | | | | |
| **Researcher(s):** | Thea Holtlund Jacobsen | | | | |
| **Supervisor(s):** | Olexandr Konovalov | | | | |

The following supporting documents are also acknowledged and approved:

1. Application Form

Approval is awarded for 5 years, see the approval expiry data above.

If your project has not commenced within 2 years of approval, you must submit a new and updated ethical application to your School Ethics Committee.

If you are unable to complete your research by the approval expiry date you must request an extension to the approval period. You can write to your School Ethics Committee who may grant a discretionary extension of up to 6 months. For longer extensions, or for any other changes, you must submit an ethical amendment application.

You must report any serious adverse events, or significant changes not covered by this approval, related to this study immediately to the School Ethics Committee.

Approval is given on the following conditions:

- that you conduct your research in line with:
  - the details provided in your ethical application
  - the University's Principles of Good Research Conduct
  - the conditions of any funding associated with your work
- that you obtain all applicable additional documents (see the 'additional documents' webpage for guidance) before research commences.

You should retain this approval letter with your study paperwork.

Yours sincerely,

Wendy Boyter

SEC Administrator

School of Computer Science Ethics Committee
Dr Olexandr Konovalov/Convenor, Jack Cole Building, North Haugh, St Andrews, Fife, KY16 9SX
Telephone: 01334 463273 Email: ethics-cs@st-andrews.ac.uk
The University of St Andrews is a charity registered in Scotland: No SC013532

FIGURE A 1: Confirmation that the Ethics Application submitted for this project was approved by the University of St Andrews School of Computer Science Ethics Committee.

# NetworkX Graph Testing

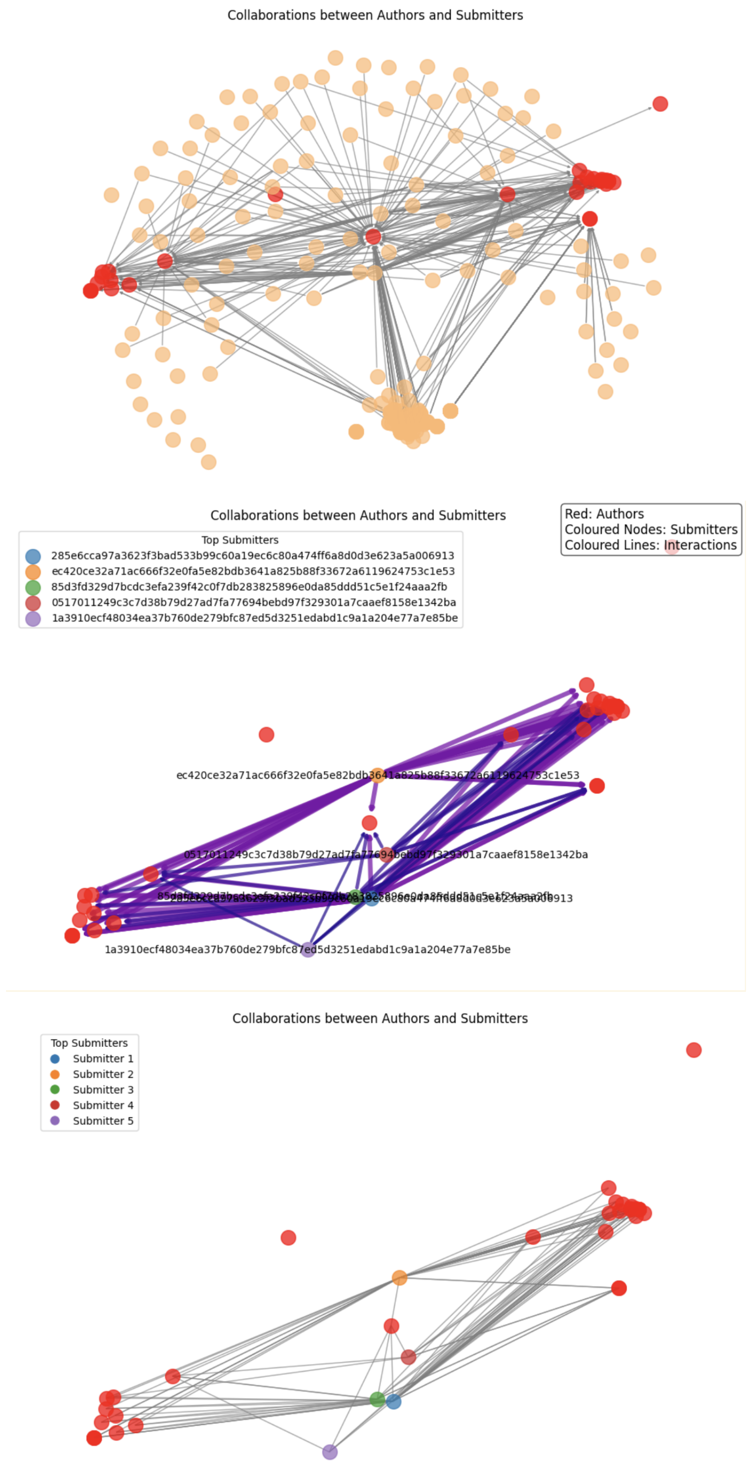

Collaborations between Authors and Submitters
Collaborations between Authors and Submitters
Red: Authors
Coloured Nodes: Submitters
Coloured Lines: Interactions
Top Submitters
285e6cca97a3623f3bad533b99c60a19ec6c80a474ff6a8d0d3e623a5a006913
ec420ce32a71ac666f32e0fa5e82bdb3641a825b88f33672a6119624753c1e53
85d3fd329d7bcdc3efa239f42c0f7db283825896e0da85ddd51c5e1f24aaa2fb
0517011249c3c7d38b79d27ad7fa77694bebd97f329301a7caaef8158e1342ba
1a3910ecf48034ea37b760de279bfc87ed5d3251edabd1c9a1a204e77a7e85be
ec420ce32a71ac666f32e0fa5e82bdb3641a825b88f33672a6119624753c1e53
0517011249c3c7d38b79d27ad7fa77694bebd97f329301a7caaef8158e1342ba
1a3910ecf48034ea37b760de279bfc87ed5d3251edabd1c9a1a204e77a7e85be
Collaborations between Authors and Submitters
Top Submitters
Submitter 1
Submitter 2
Submitter 3
Submitter 4
Submitter 5

FIGURE B 1: Iterations of the NetworkX graph in the framework, illustrating the complexity of finding the balance between readability and analytically meaningful outputs.