Mining GitHub Software Repositories to Look for Programming Language Cocktails

João Loureiro
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal

Alvaro Costa Neto
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal
Research Centre in Digitalization and Intelligent Robotics (CeDRI),
Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC),
Instituto Politécnico de Bragança, Portugal
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Barretos, Brazil

Maria João Varanda Pereira
Research Centre in Digitalization and Intelligent Robotics (CeDRI),
Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC),
Instituto Politécnico de Bragança, Portugal

Pedro Rangel Henriques
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal

Abstract

In light of specific development needs, it is common to concurrently apply different technologies to build complex applications. Given that lowering risks, costs, and other negative factors, while improving their positive counterparts is paramount to a better development environment, it becomes relevant to find out what technologies work best for each intended purpose in a project. In order to reach these findings, it is necessary to analyse and study the technologies applied in these projects and how they interconnect and relate to each other. The theory behind *Programming Cocktails* (meaning the set of programming technologies – *Ingredients* – that are used to develop complex systems) can support these analysis. However, due to the sheer amount of data that is required to construct and analyse these Cocktails, it becomes unsustainable to manually obtain them. From the desire to accelerate this process comes the need for a tool that automates the data collection and its conversion into an appropriate format for analysis. As such, the project proposed in this paper revolves around the development of a web-scraping application that can generate Cocktail Identity Cards (CIC) from source code repositories hosted on GitHub. Said CICs contain the Ingredients (programming languages, libraries and frameworks) used in the corresponding GitHub repository and follow the ontology previously established in a larger research project to model each Programming Cocktail. This paper presents a survey of current Source Version Control Systems (SVCSs) and web-scrapping technologies, an overview of Programming Cocktails and its current foundations, and the design of a tool that can automate the gathering of CICs from GitHub repositories.

2012 ACM Subject Classification Software and its engineering → Software libraries and repositories

Keywords and phrases Software Repository Mining, Source Version Control, GitHub Scraping, Programming Cocktails

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.13

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UID/00319/2023. The work of Maria João and Alvaro was supported by national funds: UID/05757 - Research Centre in Digitalization and Intelligent Robotics (CeDRI); and SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020).

1 Introduction

As technology rapidly advances, more and more ways of creating and developing applications surface. Each project uses its own blend of programming languages, libraries and frameworks (that is, its Programming Cocktail [6]). As the development of a project begins with their own unique goals in mind, it becomes important to choose the right combination of these technologies in order to ensure its success. For example, it has been considered that *JavaScript*[1] is a better choice when it comes to developing a web application than *Python*, due to faster response times [4]. By analysing Programming Cocktails and their characteristics, many useful statistics can be extracted, such as which combinations of technologies work best with each other, and what particular technology is best suited for a specific task [26]. In summary, the more data obtained about these Programming Cocktail, the stronger and more reliable structural conclusions become. One strong caveat with this approach is the difficulty in obtaining said Cocktails for analysis, as currently there is not a tool that could gather them automatically, and the process to so by hand is inefficient and time consuming.

A possible solution to this problem would be the development of a web-scraping tool to mine software repositories, like the ones hosted on GitHub[2]. By extracting the relevant data needed to generate a Cocktail Identity Card (further explained in Section 2), one could structurally define the project's Programming Cocktail. This solution would allow for many Cocktails to be gathered automatically, accelerating further development into their theories and uses. This paper describes the main techniques and results already obtained in this endeavour, while establishing the architectural and functional components of a Programming Cocktail Identity Card extractor, currently under development.

This article is divided in six sections: after the Introduction, Section 2 introduces the main concepts behind Programming Cocktails and Cocktail Identity Cards (CIC). Section 3 provides an overview of Source Version Control Systems (SVCS), with emphasis on Git and GitHub, while Section 4 presents some web scrapping techniques and tools and discusses related work. Following that, Section 5 proposes the architecture and discusses the development of the automatic CIC extractor for GitHub, and finally, Section 6 summarizes the main concepts and ideas proposed, while providing remarks for the next phases of the project.

2 Programming Cocktails

The development process of an application requires the use of many different interacting components, each with its own purpose. As such, it can be useful to identify and categorize them for organizational and statistical purposes. The concept of a *Programming Cocktail* was created to aid in these tasks. It can be defined as an agglomerate of computer programming technologies (the *Ingredients*) that are used to develop a specific software application [6]. The four types of Ingredients are:

Language: any text or graphic based language used for programming, specifying, scripting, and so on (examples: Python, JavaScript, HTML, SQL, etc.);

Library: chunks of code that augment programming languages' standard instructions and commands (the language native statements) with new or additional functionality that increase their expressiveness (examples: BS4, GLib, etc.);

[1] Mozilla page about JavaScript: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`
[2] Official GitHub website: `https://github.com`

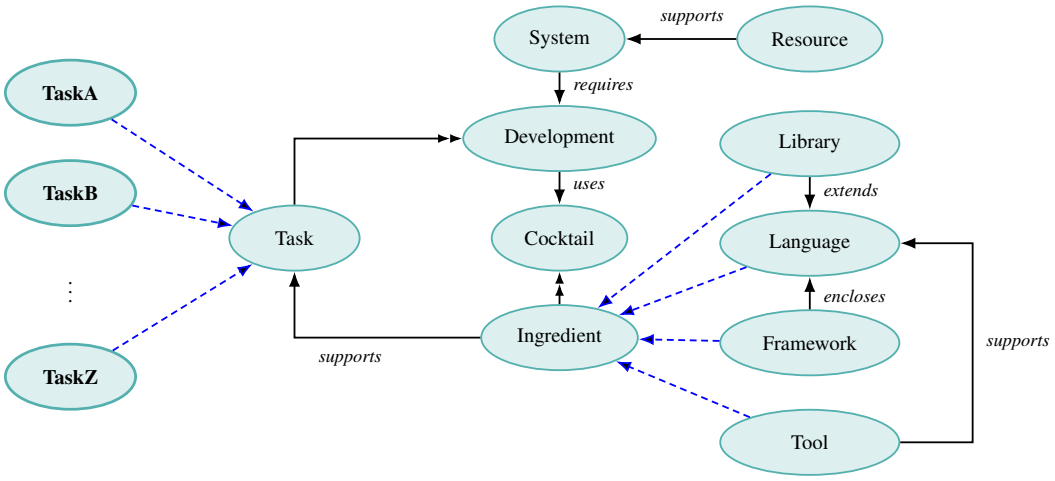


Figure 1 OntoCoq, the conceptual model for Programming Cocktails [6].

Framework: language extensions that enhance functionality and provide additional capabilities, while enforcing organizational patterns (examples: Scrapy, .NET, etc.);

Tool: any software package directly used in programming tasks during the software development process, like editors, debuggers, etc.

The ideas behind Programming Cocktails have been modelled through an ontology that describes their main concepts (see Figure 1, where the dashed blue arrows represent the "subclass-of (*isa*)" relationship, and the double-headed arrows represent the "is-part-of (*iof*)" relationship). In the ontology, each *Cocktail* is associated to a *System* that is or was under *Development*. *Systems* can be supported by *Resources*, that represent external systems and services used by the application that do not participate directly on its development (such as operating systems, database server runtimes, etc.). Finally, *Ingredients* support *Tasks* in the System's *Development*, and can represent simple, general tasks, such as frontend programming and full-stack development, or more specialized and distinct ones, like specific database communication, and memory management [6].

In order to store, identify and construct knowledge on Programming Cocktails, the concept of a Cocktail Identity Card (CIC) was created – an example of a CIC can be observed in Figure 2, modelled after a Q&A web application. The graph depicted in Figure 2 models "App1"s Programming Cocktail to formally define its Identity. Moreover, extra data can be added to the ontology to enable more types of analysis, such as risk, costs, cognitive load, etc. [5]. In short, a CIC is a resumed version of the conceptual model's instantiation, that can provide insights into the components used in the development of an application, and how they relate to each other. It also allows for the evaluation of different aspects of the application's development, such as the dependency on external resources, possible redundancies, tasks depending on many *Ingredients*, to name a few. Its continuous use can also support the documentation of a project's evolution, as long as the CIC is updated and versioned to form a Programming Cocktail history log.

3 Software Repositories

Version control is an important part of software development that consists in the management of changes to various files: programs, documents, models or other artefacts. It is a common aspect of managing a team in charge of large scale projects, offering control over any changes

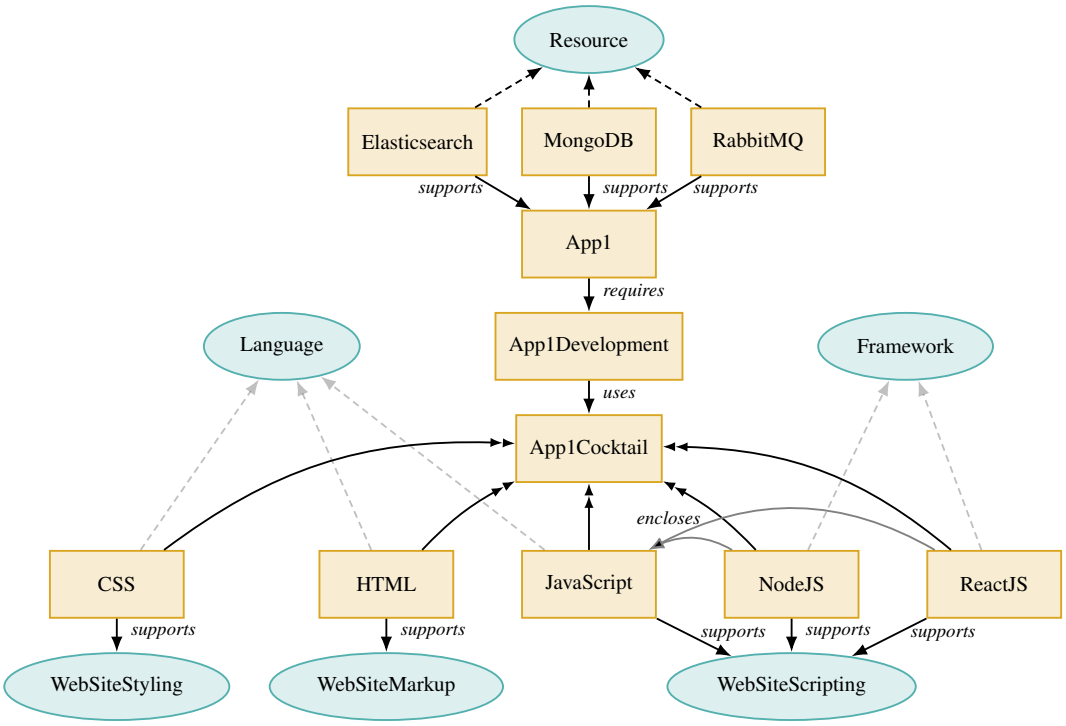


Figure 2 A fragment of OntoCoq to illustrate a Cocktail Identity Card for App1 [6].

made to the source code through various operations such as edit tracking, corrections and vandalism/spamming protection [8]. As such, Source Version Control Systems (SVCS) were devised to automate, synchronize, better safeguard, and improve efficiency of the entire version control process. By copying, merging and saving all the changes made to files, it is possible to overcome typical inefficiencies and errors in manual source control [16].

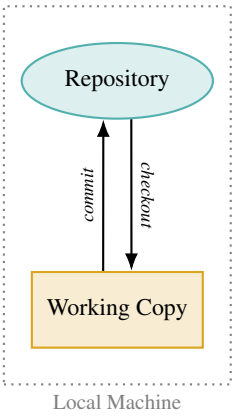
Serving as the foundation of any SVCS, a *repository* is the component responsible to store the *data and metadata* of each version of a file in a directory, as well as the history of changes made to its contents [14]. This archive is what allows one to switch between different versions, in case the need arises. The location and functionality of the repository greatly differs on what type of SVCS is used:

Local: in this type (exemplified by Figure 3a), as the name suggests, the repository containing all versions is maintained locally in the user's file system [16];

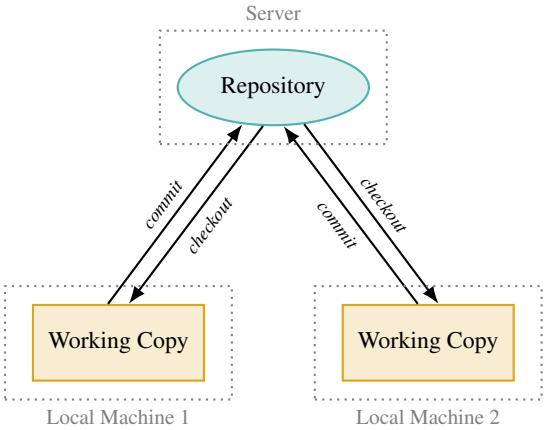
Centralized: a single repository is maintained centrally in this type of SVCS (see Figure 3b), working in a client-server fashion and storing a master copy of all files and their change history [28];

Distributed: this type of SVCS is a combination of the previous two types. It locally stores the entire history of the files in the repository, while also being capable of synchronizing the local changes back to a shared repository (Figure 3c) [28].

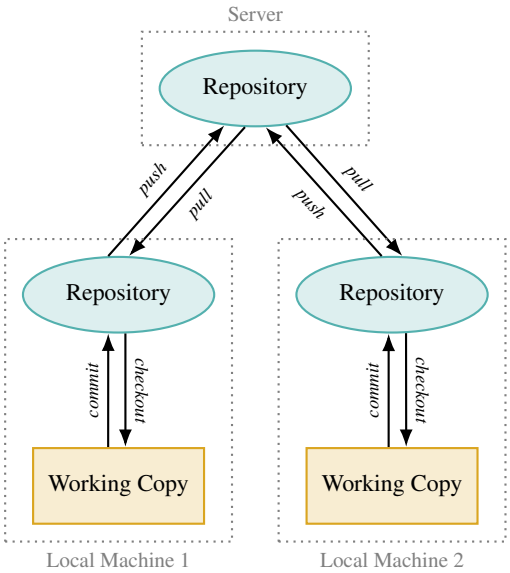
The *metadata* of the project stored in the repository is another key element for the version control. It refers to additional information about the content of the files and the repository itself, such as keywords, tags, formats, *etc.* providing an insight of the development context and history, without the need to look directly into the source code [13].



(a) Local version control system architecture.



(b) Centralized version control system architecture.



(c) Distributed version control system architecture.

Figure 3 Different repository architectures.

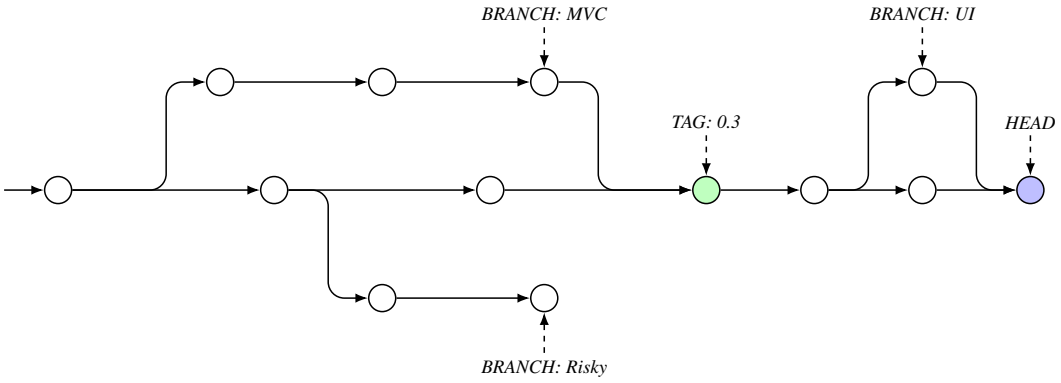


Figure 4 Representation of a Git commit graph.

3.1 Version Control Systems

There have been numerous SCVS implementations, the first notable one being Source Code Control System (SCCS)[3], a local version control system responsible for laying the base foundations of versioning files and how to store and retrieve them [12]. Revision Control System (RCS)[4] came after, allowing collaborative work between users while building upon the functionalities offered by SCCS as well as modifying how changes are stored. It resulted in a faster and more accessible alternative [25]. Finally, Concurrent Versions System (CVS)[5] is a centralized version control system that also applies the tree branch based system [2] employed by RCS to keep track of changes, while using a different method to handle changes made to the main repository: while RCS uses a lock system that prevents any changes being made simultaneously – a safer but more restrictive method – CVS employs an automatic merge system, attempting to merge any parallel changes that don't cause conflicts [2].

Out of all existing SCVS, *Git*[6] stands out as the most used SVCS according to Stack Overflow's 2022 Annual Developer Survey [19]. Beyond the previously discussed attributes of a *distributed* version control system, Git utilizes *commits* as the main way of discerning the various versions of a project. A commit is a snapshot of the current version, generally accompanied by a descriptive message. The repository is then comprised by a collection of commits organised into a graph that represents not only the current version, but also all versions that contributed to its state (see Figure 4 in which each circle represents a commit). Users can also create branches that diverge from the main one, allowing for parallel and simultaneous development of features without interference, that can be later merged into each other [17].

In terms of storage structures, Git implements different types of data objects for both structural info and files:

Blobs: store the content of any file put through version control;

Trees: store folders, meaning that they can also store other blobs and trees;

Commits: store metadata relevant to a revision, references to parent commits and the root tree;

Branches and Tags: files within Git that point to a commit [3];

[3] Open Group's Source Code Control System page: `https://pubs.opengroup.org/onlinepubs/9699919799/utilities/sccs.html`
[4] GNU OS' Revision Control System page: `https://www.gnu.org/software/rcs/`
[5] Concurrent Versions System info page: `https://www.nongnu.org/cvs/`
[6] Official Git site: `https://git-scm.com`

As for metadata, Git stores a collection of content taken from each commit, including its unique ID, author, attached message, creation date, and information pertaining to the files that were modified [13]. This means that the metadata is the main description for the versioning and subsequent identification of the repository's files, while being the identifying factor and differentiator of all versions.

3.2 GitHub

GitHub is an on-line development platform that allows users to store, manage and share code with one another through its repository hosting services. As the name implies, GitHub employs Git for version control, and offers additional services such as issue and bug tracking, access control, hosting of documentation, and more. It is also the site that houses the largest amount of source code repositories [27]. All projects hosted on GitHub are rooted on their Git code repositories, and, according to [1], one can identify five main elements in each one:

Commits: the change unit in a repository that represents a modification in the code, while also functioning as a snapshot of that version;

Issues: used to report bugs, questions pertaining to the project or announcements from the developers;

Pull requests: requests to perform changes to the code, such as merging branches;

Code reviews: contain suggestions and changes to be incorporated in the associated pull request;

Comments: the main form of communication between users, being them collaborators or not. Can be attached to issues, pull requests and code reviews.

A screenshot of a standard GitHub repository[7] can be seen in Figure 5.

As for users, they can be categorized in three types: *individual* which represents an account that belongs to a single user of the platform; *organization*, which entails a company, project or initiative; and *bot*, that provides services for repositories such as automated validation tasks [1].

4 Web Scrapping in Repositories

Data mining from the Web means to extract and compile relevant information from Webpages. One of the most used methods to achieve this is *Web Scraping*, a method that is used to extract data from websites either by directly accessing them through HTTP requests, or via a web browser. While it can be done manually, Web Scraping typically refers to the automated process that uses a *Web Crawler* to copy data found in the Web and store it locally to be later analysed and processed [22].

Since webpages are written using markup languages, such as Hypertext Markup Language (HTML), much of the desired information is in text format, allowing for the mining process to be relatively fast and flexible. As such, Web Scrapping is considered a solid approach for applications that collect several kinds of metrics, such as price trackers, product review collectors, weather data monitoring, and more [24]. It can be done through different techniques, such as HTML parsing or Document Object Model (DOM) Parsing. The former consists in fetching the raw HTML source code of a desired web page and generating its syntactic token parse tree. Then, a specialized extractor tool or system utilizes the generated

[7] KutoSchemaTools repository's page: `https://github.com/github/KustoSchemaTools`

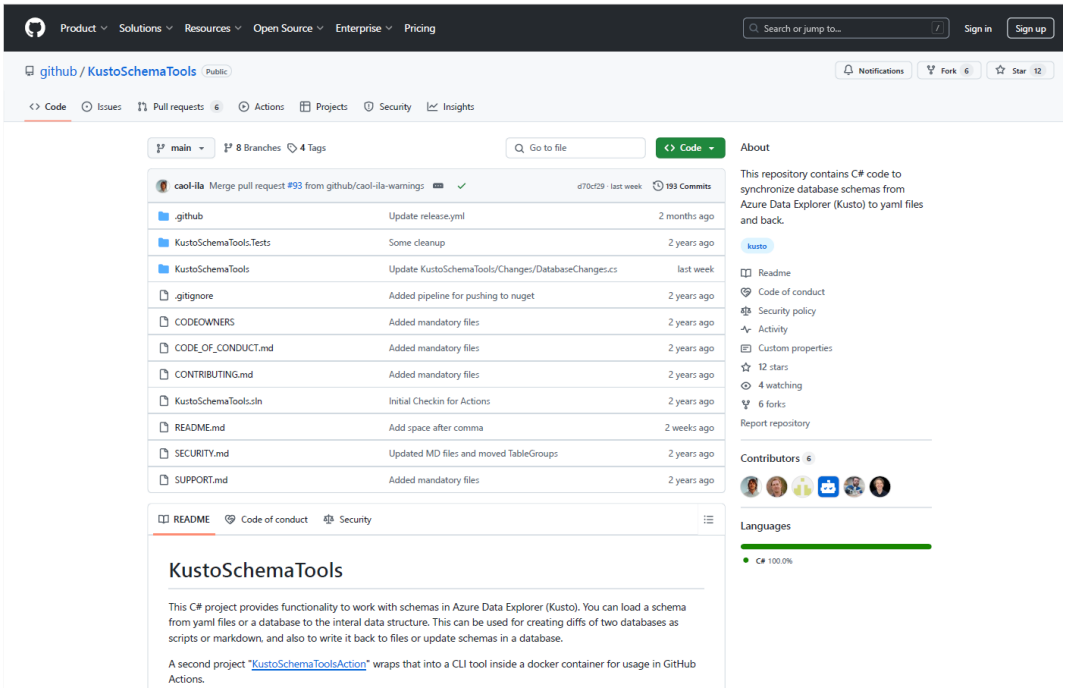


Figure 5 A GitHub repository's page as viewed in a web browser.

tree in order to extract the relevant information contained in the tags that make it up [21]. The latter utilizes the DOM[8] to create a structural tree out of the desired document. Said tree is comprised of various nodes, each representing an element/individual component of the source document, as well as itself. Within each element, there can be an identifying attribute, sub-elements, themselves elements, or text, where the context is located [23].

4.1 Technologies and Techniques to Implement Web Crawlers

There is a wide range of technologies that allow users to create a specialized Web Crawler. These can range from:

- Libraries, such as BeautifulSoup4 (BS4)[9] in Python or Jsoup[10] in Java, that implement their respective languages idioms on top of a HTML parser, allowing for the search, navigation and modification of the generated parse tree [20, 9];
- Frameworks, like Scrapy[11], that allows users to create Web Crawlers (*spiders*, in the framework's lingo) that extract data from websites or even perform automated testing [10];
- Or even tools, like Selenium[12], an open-source automation tool supported by various programming languages to test web applications across many different platforms and browsers.

[8] A platform-independent interface that allows the creation, navigation and modification of elements and content of HTML and XML documents.
[9] Official BS4 website: `https://www.crummy.com/software/BeautifulSoup/`
[10] Official Jsoup website: `https://jsoup.org`
[11] Official Scrapy website: `https://scrapy.org`
[12] Official Selenium website: `https://www.selenium.dev`

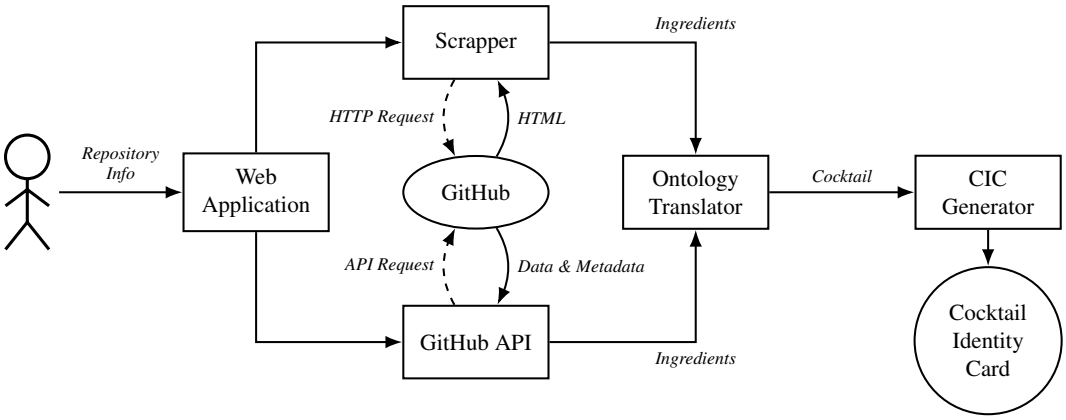


Figure 6 Architecture for the Cocktail Identity Card extractor.

Libraries like BS4 and Jsoup are easier to use and faster than more complex frameworks like Scrapy, that are more suited to scrape complex websites, and thus, larger projects [20]. These technologies can also be combined in order to cover each others limitations. As explored in [18], Selenium is used to take care of browser interactions that others cannot, in this case, to dynamically send data to input fields and press buttons on a website, while BS4 is used to grab the data from the tags.

In the specific case of mining GitHub repositories, [7] details a data extractor that was implemented to obtain characteristics frequently used as project selection criteria. It consists of a hybrid application that makes requests to GitHub's official Representational State Transfer (REST) Application Programming Interface (API) and also utilizes a specialized Web Crawler to scrape the repository's webpage. Said API allows users to extract information about the repositories it hosts, from general information like the description, owner and languages, to its contents or even analytics and collaboration data. The Crawler is used as a complement to obtain information that cannot be retrieved from the API alone. It was built primarily using Jsoup, while applying Selenium to handle the dynamically generated content. As the use of Selenium results in a large hit to the application's performance, it is only utilized as a backup strategy when Jsoup returns an error. The works detailed in [11] and [15], while similarly related in regards to the subjects of scrapping GitHub data by foraging repositories and using it to achieve a certain goal, have no direct relevancy to this project.

5 Mining GitHub to Extract Programming Cocktails

The focus of this paper is the proposal of a system able to extract, process, and analyse data from GitHub repositories in order to create Cocktail Identity Cards. Figure 6 shows its architecture.

The process of obtaining a CIC begins with the user inserting the target repository's information into the application, which in turn passes it into the *Scraper*. The *Scraper* uses this info to make requests to the *GitHub* servers in order to retrieve the information of the repository page and begin discovering its Ingredients. The application also utilizes the information to make requests to the *GitHub API* and obtain additional information on the repository. The relevant data that was scraped is then gathered and passed onto the *Ontology Translator*. This component will rearrange and transform the data into the ontology's syntax

and format. By following the conceptual model (Figure 1), it generates the concepts, the individuals and their relationships that can be identified in the repository. With the full ontology instantiated, the *CIC Generator* trims it down into its *Cocktail Identity Card*, which is then presented to the user.

5.1 Identifying Ingredients

In regards to identifying the Ingredients, possible strategies have already been devised, including:

- Scraping every webpage in the repository, including each individual file, as GitHub displays a specific webpage that contains the contents of each file in a repository.
- Searching specific contents in different file types, as each language can directly import modules or libraries via specific commands.
- Applying machine learning techniques to the repository's structure and the content of the *README* file (if present), which usually includes information about the project.
- Analysing the contents of common dependency files (such as `pyproject.toml` for Python, and `package.json` for JavaScript), which contain key components used in the project.

The final method for gathering Ingredients will include a combination of these strategies. When possible, the tool will prioritize API calls over Web Scraping to improve the speed and efficiency of the fetching and extracting processes.

As of the writing of this paper, the prototype of the application makes calls to GitHub API to retrieve information about the repository in JSON format. After inserting a valid personal access token, and the URL of the desired public repository, the application makes an request to the repository's endpoint[13] in order to retrieve general information about the repository such as its name, detailed information about the owner, its approximated size in kilobytes, the name of the default/main branch, other endpoint URLs that can be used to retrieve other information (tags, branches, collaborators...) and many others.

Next, it makes a request to the `languages` endpoint[14] that returns a JSON dictionary where the keys represent the languages that GitHub detected in the project, and the values associated to each key are the number of bytes of code that were written in that language, as seen in Listing 2. The information retrieved from this endpoint will be used to instantiate the *Language* concept of the ontology and all of the ingredients of this type. An example of a resulting Cocktail Identity Card containing only the languages obtained from the API is presented in Figure 7.

The `readme` endpoint[15] is then used to retrieve the raw content of the project's main `README.md` file. As specified in the API documentation, this content is encoded in Base64 and, as such, it is appropriately decoded back to UTF-8 using Python's `base64` module[16].

Then the application recursively uses the `branch` endpoints[17] to retrieve the name and path of all files present in the repository. This endpoint requires the specification of the name of the branch from which to pull the files from, with the desired target being the default branch – most commonly referred to as `main`. However, since this naming is not a rule and more of a convention, the application uses the `default_branch` parameter retrieved from

[13] `https://api.github.com/repos/{owner_name}/{repository_name}`
[14] `https://api.github.com/repos/{owner_name}/{repository_name}/languages`
[15] `https://api.github.com/repos/{owner_name}/{repository_name}/readme`
[16] Base64 module documentation: `https://docs.python.org/3/library/base64.html`
[17] `https://api.github.com/repos/{owner_name}/{repository_name}/git/trees/{branch_name}?recursive=1`

Listing 1 Example of a response from a repository endpoint. Repository used: https://github.com/cosmochefe/Strawberry.

```
{
  "id": 14406041,
  "node_id": "MDEwOlJlcG9zaXRvcnkxNDQwNjA0MQ==",
  "name": "Strawberry",
  "full_name": "cosmochefe/Strawberry",
  ...
  "owner": {
    "login": "cosmochefe",
    "id": 988484,
    ...
  }
  "html_url": "https://github.com/cosmochefe/Strawberry",
  "description": "Compilador did\u00e1tico para Portugol.",
  ...
  "collaborators_url": "https://api.github.com/repos/cosmochefe/
    ↪ Strawberry/collaborators{/collaborator}",
  "branches_url": "https://api.github.com/repos/cosmochefe/Strawberry/
    ↪ branches{/branch}",
  "tags_url": "https://api.github.com/repos/cosmochefe/Strawberry/tags",
  ...
  "size": 324,
  "default_branch": "master",
  ...
}
```

Listing 2 Example of a response from a repository's languages endpoint. Repository used: https://github.com/cosmochefe/Strawberry.

```
{
  "languages": {
    "C": 73481,
    "C++": 291
  }
}
```

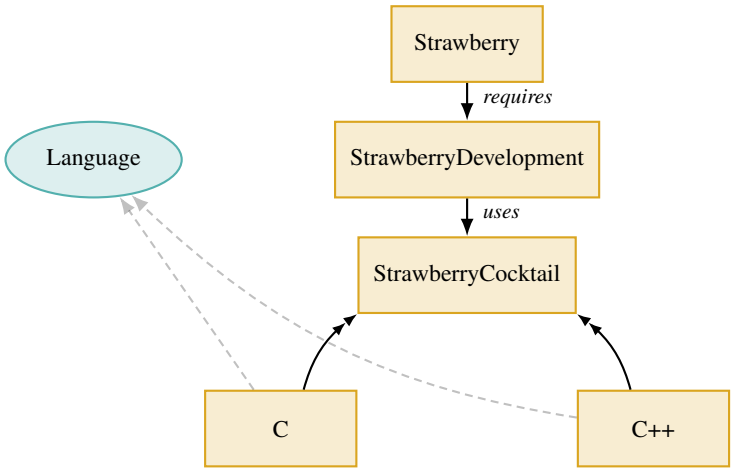


Figure 7 Cocktail Identity Card for the Strawberry repository (see Listing 2).

Listing 3 Example of a response from a repository's branch endpoint. Repository used: https://github.com/rust-lang/rust.

```
{
  ... ,
  {
    "path": "compiler",
    "type": "tree",
    "url": "https://api.github.com/repos/rust-lang/rust/git/trees/
    ↪ a0b0dae8da54f929eae78a96ce14fe526ea91e70"
  },
  {
    "path": "compiler/rustc",
    "type": "tree",
    "url": "https://api.github.com/repos/rust-lang/rust/git/trees/6
    ↪ f54be4ccb00e6baa2c949de41120a831645e8d0"
  },
  {
    "path": "compiler/rustc/Cargo.toml",
    "type": "blob",
    "url": "https://api.github.com/repos/rust-lang/rust/git/blobs/
    ↪ f4caa3ef769d5f9d9e6c960af53b7d3f0cca9cb8"
  },
  ...
}
```

the repository's endpoint to ensure that it is always the one targeted. The `recursive=1` flag is used to ensure that the file paths are retrieved recursively. The response is a JSON dictionary that contains a list of objects, one for each file and their attributes. Three main attributes, the most relevant to the task at hand, are chosen to be processed: `path`, the name of the path where one can locate the file, `type`, the Git type of the file as previously explained in Section 3.1 and `url`, the API endpoint of the file. This information is later used to reassemble the complete file structure of the repository.

Lastly, the application utilizes the GraphQL endpoint[18] in order to fetch the content of certain common dependency files. This endpoint is particularly useful for more complex and specific requests as it allows the use of GraphQL[19] queries. In this case, the query requests the dependency files seen in Table 1 by name, that when processed will return the raw content of those that are present. The information gathered from this endpoint, along with the one from the `readme` and the `branch` one, will be used to instantiate the *Library*, *Framework* and possibly *Tool* concept of the ontology and to later represent the ingredients of these types.

One big limitation encountered so far is related to the GitHub API rate limits. In order to obtain the information of a repository, multiple calls to the API must be made. Without user authentication, only 60 requests per hour can be made, while when authenticated it is possible to perform up to 5000 request or 15000 if the user is related to the GitHub Enterprise Cloud[20]. This means that the application will have to enforce GitHub authentication of some kind in order to increase the instance limit, try to limit as much as possible the number of requests made to the API and block usage once the limit has been reached for that hour.

[18] `https://api.github.com/graphql`
[19] Official GraphQL website: `https://graphql.org`
[20] `https://docs.github.com/en/rest/using-the-rest-api/rate-limits-for-the-rest-api`

Table 1 Dependency files that the application currently requests, sorted by their language.

Language	Dependency Files
Python	requirements.txt, pyproject.toml, setup.py
JavaScript	package.json, yarn.lock, package-lock.json
Java	pom.xml, build.gradle
Ruby	Gemfile, Gemfile.lock
PHP	composer.json, composer.lock
Go	go.mod, go.sum
Rust	Cargo.toml, Cargo.lock
.NET Languages	.csproj, packages.config

Listing 4 Format of the GraphQL query used to retreive the contents of the dependency files.

```
query FetchDependencyFiles(${owner_name}: String!,
                           ${repo_name}: String!) {
  repository(owner: ${owner_name}, name: ${repo_name}) {
    # Repeat for each files
    requirements: object(expression: "HEAD:{file_name}") {
      ... on Blob { text }
    }
  }
}
```

Immediate follow-up work will involve analysing the data currently being fetched to determine what information can be extracted and the best methods for doing so. Alternate ways to obtain information that may arise will also be explored and implemented if deemed acceptable, such as the implementation of some of the web scraping techniques discussed in order to retrieve information only present in the repository's HTML page, or even potentially replace an endpoint request to reduce the number of requests made.

6 Conclusion

This paper highlighted the main concepts behind data mining of software repositories, more specifically, GitHub. It also presented the foundations of Programming Cocktails, Cocktail Identity Cards and their ontology-based models. Finally in the paper we proposed, as the main contribution, the architecture of an automated web application aimed at discovering Programming Cocktails and generate their Identity Cards based on data extracted from GitHub repositories. These results will provide a deeper understanding on the myriad of programming technologies commonly present in the development of complex applications. By automating the gathering of Programming Cocktails, more results and conclusions on this topic can be reached, extending knowledge on both software development complexity and evolution.

Currently, the extractor is capable of constructing CICs containing the languages identified in the Github API, while also gathering data from many other sources of information. Such as the `README.md` file, the overall repository folder structure and some characteristic dependency files.

The next phase of this project will continue with the implementation of the Cocktail Identity Card extractor, taking into account possible paths for discovery, such as the ones presented in Section 5, as well as the development and implementation of methods to analyse

and extract Ingredients from the retrieved information. After that, a set of public GitHub repositories will be defined to be used as a case study to test and optimize said methods, as well as identifying other possible additions and options to the process. Beyond what was discussed in this article, other alternative approaches could be explored, such as one based purely on machine learning techniques, focused on the creation and training of a robust identification model. Lastly, some other techniques for automatic discovery of *Resources* and *Tasks* are planned to be implemented in order to complete the extraction of Cocktail Identity Cards.

References

1 Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. An empirical study on the survival rate of github projects. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, pages 365–375, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3524842.3527941.

2 John Alhson. Parallel (concurrent) code development or how to develop software in a distributed workgroup or the concurrent versions system, cvs. *Nuclear Physics B - Proceedings Supplements*, 61(3):674–677, 1998. Proceedings of the Fifth International Conference on Advanced Technology and Particle Physics. doi:10.1016/S0920-5632(97)00636-1.

3 Natanael Arndt, Patrick Naumann, Norman Radtke, Michael Martin, and Edgard Marx. Decentralized collaborative knowledge management using git. *Journal of Web Semantics*, 54:29–47, 2019. Managing the Evolution and Preservation of the Data Web. doi:10.1016/j.websem.2018.08.002.

4 Sai Sri Nandan Challapalli, Prakarsh Kaushik, Shashikant Suman, Basu Dev Shivahare, Vimal Bibhu, and Amar Deep Gupta. Web development and performance comparison of web development technologies in node.js and python. In *2021 International Conference on Technological Advancements and Innovations (ICTAI)*, pages 303–307, 2021. doi:10.1109/ICTAI53825.2021.9673464.

5 Alvaro Costa Neto, Maria João Varanda Pereira, and Pedro Rangel Henriques. Application of programming cocktails identity cards to development complexity analysis. In *Proceedings of the 23nd Belgium-Netherlands Software Evolution Workshop (BENEVOL 2024)*. CEUR-SW.org, 2024. To be published.

6 Alvaro Costa Neto, Maria João Varanda Pereira, and Pedro Rangel Henriques. An ontology to understand programming cocktails. In Maria Ganzha, Leszek Maciaszek, Marcin Paprzycki, and Dominik Ślezak, editors, *Proceedings of the 19th Conference on Computer Science and Intelligence Systems*, Annals of Computer Science and Information Systems. IEEE, 2024. To be published.

7 Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling projects in github for msr studies. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 560–564, 2021. doi:10.1109/MSR52588.2021.00074.

8 N. Deepa, B. Prabadevi, L.B. Krithika, and B. Deepa. An analysis on version control systems. In *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, pages 1–9, 2020. doi:10.1109/ic-ETITE47903.2020.39.

9 Yılmaz Dikilitaş, Çoşkun Çakal, Ahmet Can Okumuş, Halime Nur Yalçın, Emine Yıldırım, Ömer Faruk Ulusoy, Bilal Macit, Aslı Ece Kırkaya, Özkan Yalçın, Ekin Erdoğmuş, and Ahmet Sayar. Performance analysis for web scraping tools: Case studies on beautifulsoup, scrapy, htmlunit and jsoup. In Fausto Pedro García Márquez, Akhtar Jamil, Alaa Ali Hameed, and Isaac Segovia Ramírez, editors, *Emerging Trends and Applications in Artificial Intelligence*, pages 471–480, Cham, 2024. Springer Nature Switzerland.

10 M El Asikri, S Knit, and H Chaib. Using web scraping in a knowledge environment to build ontologies using python and scrapy. *European Journal of Molecular & Clinical Medicine*, 7(03):2020, 2020.

11 Hamzeh Eyal Salman. Ai-based clustering of similar issues in github's repositories. *Journal of Computer Languages*, 78:101257, 2024. doi:10.1016/j.cola.2023.101257.

12 Alan L. Glasser. The evolution of a source code control system. *SIGSOFT Softw. Eng. Notes*, 3(5):122–125, January 1978. doi:10.1145/953579.811111.

13 Youngtaek Kim, Jaeyoung Kim, Hyeon Jeon, Young-Ho Kim, Hyunjoo Song, Bohyoung Kim, and Jinwook Seo. Githru: Visual analytics for understanding software development history through git metadata analysis. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):656–666, 2021. doi:10.1109/TVCG.2020.3030414.

14 Ali Koc and Abdullah Uz Tansel. A survey of version control systems. *ICEME 2011*, 2011.

15 Sandeep Kaur Kuttal, Se Yeon Kim, Carlos Martos, and Alexandra Bejarano. How end-user programmers forage in online repositories? an information foraging perspective. *Journal of Computer Languages*, 62:101010, 2021. doi:10.1016/j.cola.2020.101010.

16 Rana Majumdar, Rachna Jain, Shivam Barthwal, and Chetna Choudhary. Source code management using version control system. In *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pages 278–281, 2017. doi:10.1109/ICRITO.2017.8342438.

17 Edi Muškardin, Tamim Burgstaller, Martin Tappler, and Bernhard K. Aichernig. Active model learning of git version control system. In *2024 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 78–82, 2024. doi:10.1109/ICSTW60967.2024.00024.

18 Simona Vasilica Oprea and Adela Bâra. Why is more efficient to combine beautifulsoup and selenium in scraping for data under energy crisis. *Ovidius University Annals, Economic Sciences Series*, 22(2):146–152, 2022.

19 Stack Overflow. Stack overflow developer survey 2022. https://survey.stackoverflow.co/2022/#version-control-version-control-system, 2022. Last Accessed: 12/01/2025.

20 Sakshi Pant, Er. Narinder Yadav, Milan, Monnie Sharma, Yash Bedi, and Anshuman Raturi. Web scraping using beautiful soup. In *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, volume 1, pages 1–6, 2024. doi:10.1109/ICKECS61492.2024.10617017.

21 Pranit Patil, Pramila Chawan, and Prithviraj Chauhan. Parsing of html document. *International Journal of Advanced Research in Computer Engineering & Technology*, 1:2278–1323, July 2012.

22 Ruchitaa Raj N R, Nandhakumar Raj S, and Vijayalakshmi M. Web scrapping tools and techniques: A brief survey. In *2023 4th International Conference on Innovative Trends in Information Technology (ICITIIT)*, pages 1–4, 2023. doi:10.1109/ICITIIT57246.2023.10068666.

23 Martina Radilova, Patrik Kamencay, Robert Hudec, Miroslav Benco, and Roman Radil. Tool for parsing important data from web pages. *Applied Sciences*, 12(23), 2022. doi:10.3390/app122312031.

24 Vidhi Singrodia, Anirban Mitra, and Subrata Paul. A review on web scrapping and its applications. In *2019 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–6, 2019. doi:10.1109/ICCCI.2019.8821809.

25 Walter F. Tichy. Design, implementation, and evaluation of a revision control system. In *Proceedings of the 6th International Conference on Software Engineering*, ICSE '82, pages 58–67, Washington, DC, USA, 1982. IEEE Computer Society Press. URL: http://dl.acm.org/citation.cfm?id=807748.

26 Federico Tomassetti and Marco Torchiano. An empirical assessment of polyglot-ism in github. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2601248.2601269.

27 Wikipedia. Comparison of source-code-hosting facilities. https://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities#Popularity, 2024. Last Accessed: 05/10/2024.

28 Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. Version control system: A review. *Procedia Computer Science*, 135:408–415, 2018. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life. doi:10.1016/j.procs.2018.08.191.