Анализ качества ведения репозиториев GitHub для языка C++

Томилин Илья Сергеевич
Высшая школа программной инженерии
Санкт-Петербургский политехнический
университет Петра Великого
Санкт-Петербург, Россия
tomilin.is@edu.spbstu.ru

Никифоров Игорь Валерьевич
Высшая школа программной инженерии
Санкт-Петербургский политехнический
университет Петра Великого
Санкт-Петербург, Россия
nikiforov_iv@spbstu.ru

Фёдоров Иван Алексеевич
Высшая школа программной инженерии
Санкт-Петербургский политехнический
университет Петра Великого
Санкт-Петербург, Россия
fedorov.ia@edu.spbstu.ru

***Аннотация*—В данной статье представлен один из возможных подходов, позволяющий создавать программное решение для сбора набора данных с github, а также производить анализ на основе полученного набора данных и визуализировать результаты. В качестве объекта исследование был выбран анализ репозиториев C++ на качество их ведения. Данная статья может помочь упростить принятие решения в выборе средств и подхода для достижения цели анализа репозиториев.**

Ключевые слова—data, analysis, github, bigdata

I. Введение

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. На 2021 год GitHub используют больше 73 миллионов разработчиков, больше 4 миллионов организаций и содержится более 200 миллионов различных репозиториев. Одни репозитории ведутся с хорошим стилем (наличие README, описания, манифеста системы сборки и.т.п.), а другие наоборот не следуют этому правилу. Особенно актуально соблюдение этих правил для репозиториев языка C++, поскольку они требуют повышенного качества ведения. Анализ качества репозитория необходим, чтобы выявить насколько качественно разработчики подходят к ведению своих репозиториев, стоит ли использовать структуру случайно выбранного репозитория или более тщательно искать пример качественного репозитория с точки зрения его ведения при создании своего проекта. В статье решаются задачи связанные с анализом:

- лицензий репозиториев, наличием README файла;
- наличием Dockerfile, для определения того имеет ли проект CI;
- используемого стиля отступов в файлах с исходным кодом;
- наиболее популярных языков, которые используют в одном репозитории совместно с C++;
- систем сборки;
- названий веток, наличием gitflow.

II. Обзор аналогичных решений

Схожие задачи анализа репозиториев на различные метрики рассматривались в статьях, краткое содержание которых будет изложено ниже.

A. Анализ качества свободно разрабатываемого программного обеспечения

В статье [2] рассматривались какие-либо существенные взаимосвязи между качеством проекта и характеристиками членов команды, которая работает над проектом. За основу показателя качества проекта были выбраны два таких показателя, первый из них отображает популярность проекта, второй качество поддержки, которое предоставляют члены команды. Первый показатель определялся на основе количества “звезд” репозитория, второй на основе анализа “выживаемости” пользователей проекта.

B. Анализ настроений комментариев к репозиториям

В статье [3] рассматривались эмоции комментариев, которые по мнению авторов могут оказывать большое влияние на производительность, качество выполненения задач. Авторы использовали лексический анализ настроений для изучения эмоций выраженных в комментариях к репозиторию, анализировали взаимосвязь с различными факторами такими как: язык программирования, время, дата.

C. Анализ важности и влияния репозиториев

В следующей статье [4] исследовалась важность репозиториев, поскольку тысячи разработчиков, которые размещают свои проекты на github превозносят новые нерешенные проблемы тем самым мотивируя других людей на участие в решение этих проблем. В статье описываются достаточно сложные алгоритмы исследования этой проблемы на основе обращений пользователей к репозиториям.

D. Анализ наличия системы непрерывной интеграции

В статье [5] авторы затронули такую тему как наличие систем непрерывной интеграции в репозиториях. Производилось исследование наличие системы Travis CI в репозиториях с различными языками программирования.

E. *Анализ репозиториев связанных с биоинформатикой*

В статье [6] проводилось исследование репозиториев связанных с биоинформатикой. Анализировался исходный код этих репозиториев для изучения взаимосвязей между свойствами кода, деятельностью разработчиков, программным обеспечением. Было проанализировано 1720 репозиториев и получено большое количество необходимой информации. Данное исследование стало первым в данном направлении.

F. *Анализ качества ведения Dockerfile в репозиториях*

В статье [7] проводилось исследование содержимого Dockerfile с целью определения его качества. Для авторов качество Dockerfile определялось в написание конкретной версии образов, наличие строки с описанием разработчика и соблюдение хорошего стиля написания Dockerfile согласно рекомендациям описанным на сайте с документацией.

III. Описание модели данных и характеристики набора данных

A. *Описание модели*

Данные для анализа формируются в json документы со структурой показанной на рис. 1. В структуре находятся поля, которые заполняются на основе API github и разбора директории загруженного репозитория.


```
{
    "all_lines": 0,
    "another_files": 0,
    "another_languages": [
    ],
    "branches": [
    ],
    "build_system": [
    ],
    "contributors_female": 0,
    "contributors_male": 0,
    "contributors_total_count": 0,
    "contributors_unrecognized": 0,
    "count_spaces": 0,
    "count_tabs": 0,
    "cpp_src_files": 0,
    "cpp_src_lines": 0,
    "created_at": "",
    "description": "",
    "dockerfile": false,
    "forks": 0,
    "full_name": "",
    "indention_method": "",
    "indention_method_dominate": "",
    "issue_titles": [],
    "issues_close": 0,
    "issues_open": 0,
    "license": "Other",
    "readme": false,
    "stars": 0,
    "total_issues": 0,
    "updated_at": ""
}
```


Рисунок 1 – Структура json документа

B. *Характеристики набора данных*

За все время сбора данных, были проанализированы репозитории за следующие года: 2010, 2011, 2012, 2013, 2014 и частично 2015. Общее число проанализированых github репозиториев составило – 178469 штук, fork не рассматривались. Также следует отметить, что это приблизительно 6% от общего числа всех созданных репозиториев для языка C++ в период с 2010-01-01 до 2021-12-01 годы. Общий объем загруженных репозиториев составил 4.4 ТБ. Количество часов, которое потребовалось для загрузки репозиториев составило 387 часов. Общий объем json документов с данными составил 758 МБ. Для загрузки документов с данными в индекс elasticsearch, потребовалось 40 минут.

IV. Предлагаемое программное решение и описание его особенностей

Для упрощения взаимодействия с программным решением и автоматизацией работы для конечного пользователя был создан программнный дистрибутив содержащий 9 директорий и 30 файлов в которые входят: автоматизационные сценарии, приложение для разбора API github, приложение для разбора загруженных репозиториев, Dockerfile и docker-compose файлы для возможности сбора и анализа набора данных в специальных, изолированных, от основной операционной системы окружениях, дополнительные файлы и сценарии необходимые для корректной работы приложения. Архитектура приложения показана на рис. 2.

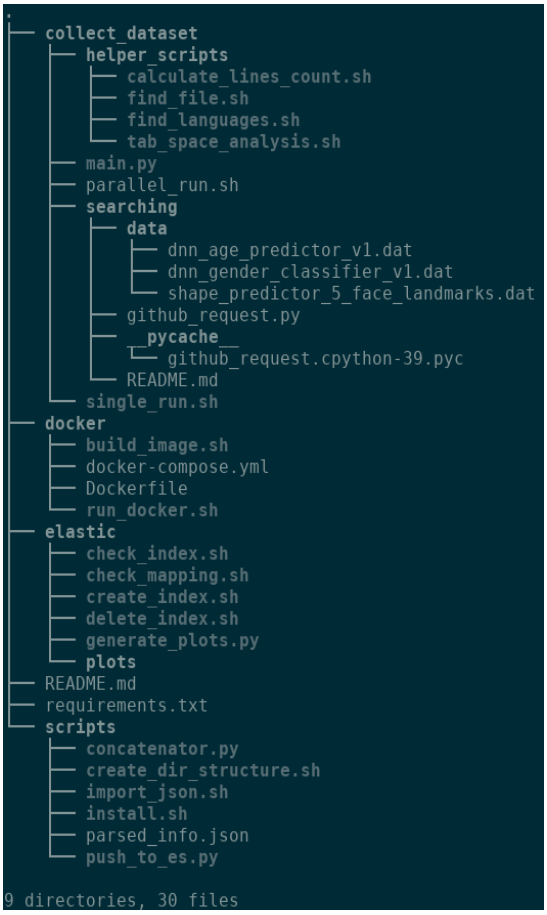

Рисунок 2 – Структура поставляемого дистрибутива

Дистрибутив был отлажен и протестирован в ОС Linux, дистрибутив Debian 11 «bullseye».

V. Описание реализации и технологии

A. *Архитектура программного решения*

Программное решение было реализовано с использованием скриптового языка программирования Python, в качестве автоматизационных скриптов используются shell сценарии написанные на bash. Архитектура приложения показана на рис. 3.

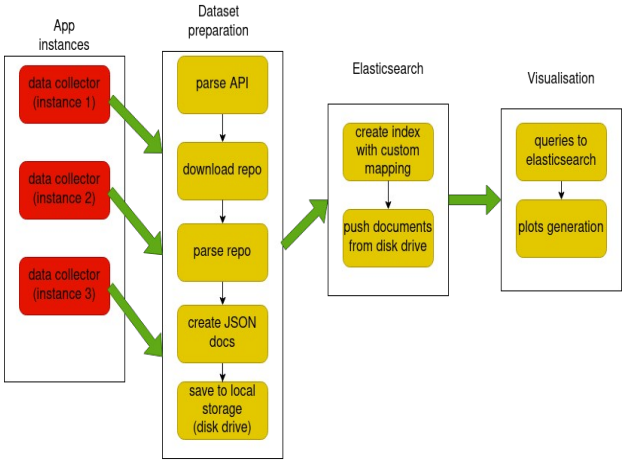

Рисунок 3 – Архитектура приложения

Архитектура разбита на три основные части:

- создания набора данных на основе информации API github и разбора репозиториев;
- запуск экземпляра elasticsearch, создание нового индекса и загрузка полученных данных из первого пункта в индекс;
- создание запросов к индексу для получения необходимых метрик на основе набора данных и визуализация полученных метрик.

B. Описание библиотек

Для запросов к API github использовались библиотеки requests, curl. Для разбора содержимого репозитория:

- анализ репозитория для поиска различных расширений файлов и их количества;
- частота вхождение того или иного файла по его типа.

использовалась библиотека cloc. Пример выходного результата работы библиотеки cloc показан на рис. 4.

```
      33 text files.
      33 unique files.
       4 files ignored.

github.com/AlDanial/cloc v 1.86  T=0.02 s (1215.5 files/s, 90916.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                           5            220            320            932
Bourne Shell                    15             69             15            348
XML                              5              0              0            207
Dockerfile                       1              7              8             31
Markdown                         2              3              0             30
YAML                             1              4             24             20
JSON                             1              0              0              6
-------------------------------------------------------------------------------
SUM:                            30            303            367           1574
-------------------------------------------------------------------------------
```

Рисунок 4 – Результат работы cloc

Для поиска Dockerfile, файла с README использовались самописные сценарии на bash. Также все сценарии отвечающие за автоматизацию различных действий, написаны на bash.

За модуль распознавания пола по изображению отвечает библиотека dlib-models. Библиотека представляет сеть ResNet с 29 слоями, для создания этой сети была взята сеть из статьи [1], которая подверглась небольшим изменениям по слоям. Данная нейронная сеть тренировалась на наборе данных из 3 миллионов лиц. Пример распознавания пола моделью показана на рис. 5.

Рисунок 5 – Распознавание пола dlib-models с выходной визуализацией

Для визуализации данных использована библиотека matplotlib. Данная библиотека позволяет создавать различные типы графиков. API, которое предоставляет библиотека является достаточно низкоуровневым, что позволяет очень гибко настраивать различные параметры графиков, однако усложняет написание кода. Пример графика полученный с помощью matplotlib показан на рис. 6.

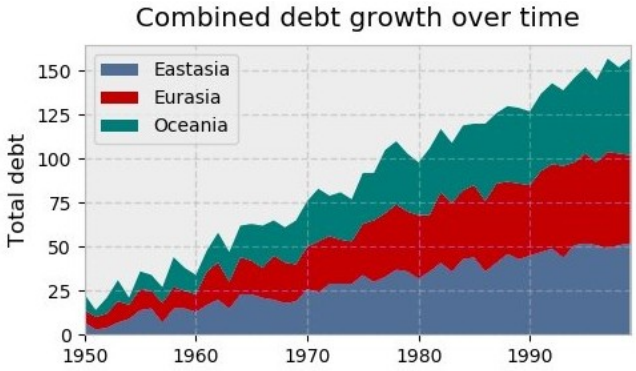

Рисунок 6 – Пример графика созданный с помощью matplotlib

C. Потребление ресурсов

Расход ресурсов процессорного времени и оперативной памяти не является узким местом для приложения, потребление ресурсов с тремя запущенными экземплярами приложения показано на рис. 7.

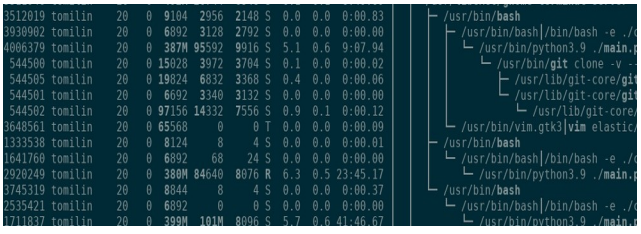

Рисунок 7 – Потребление ресурсов приложения запущенного в трех экземплярах

Как было выяснено, основная проблема связана со временем анализа репозиториев, которое тратится на загрузку репозитория, анализ изображений разработчиков, разбор содержимого репозитория.

VI. Результаты анализа репозиториев

В результате анализа набора данных были получены искомые результаты по рассматриваемым критериям.

A. *Количество используемых лицензии*

На основе анализа лицензий рис. 8, можно сделать вывод, что большинство репозиториев не используют какую-либо лицензию. Самая популярная из известных лицензий стала MIT License.

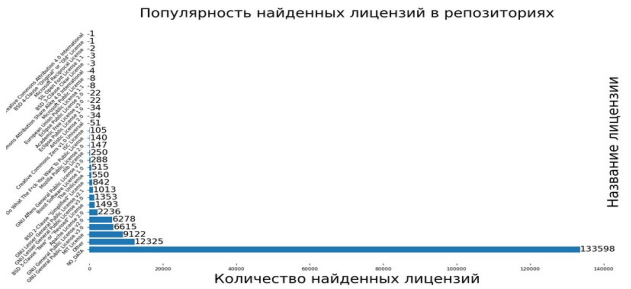

Рисунок 8 – Используемые лицензии в репозиториях

B. *Наличие README файла в репозиториях*

Выполнив анализ репозиториев на наличие README файла рис. 9, стало понятно, что примерно половина репозиториев не имеют README файла, однако с небольшим отрывом в ~1.5% лидируют репозитории в которых README присутствует.

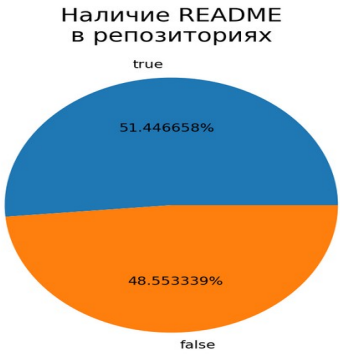

Рисунок 9 – Наличие файла README в репозиториях

C. *Наличие Dockerfile манифеста в репозиториях*

Выполнив анализ репозиториев на наличие Dockerfile манифеста рис. 10, стало понятно, что меньше 1% из общего числа репозиториев имеют Dockerfile.

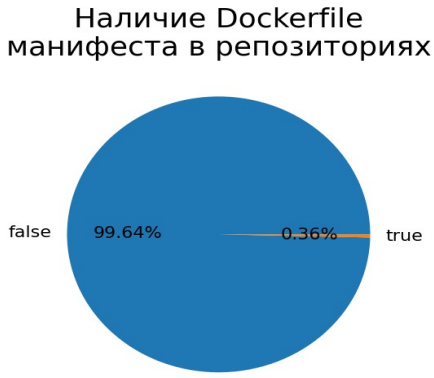

Рисунок 10 – Наличие файла README в репозиториях

D. *Тип отступов в репозиториях*

Выполнив анализ отступов в репозиториях рис. 11, удалось выяснить, что преобладает смешанный стиль, затем идут пробелы и на последнем месте табуляция.

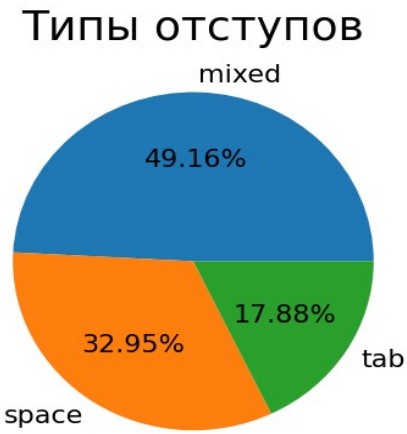

Рисунок 11 – Тип используемого отступа в репозиториях

E. *Наиболее часто встречающиеся языки в одном репозитории совместно с C++*

Выполнив анализ расширений файлов с исходным кодом рис. 12, был определен список языков, если исключить языки сборочных манифестов, то самыми популярными стали shell, python и C.

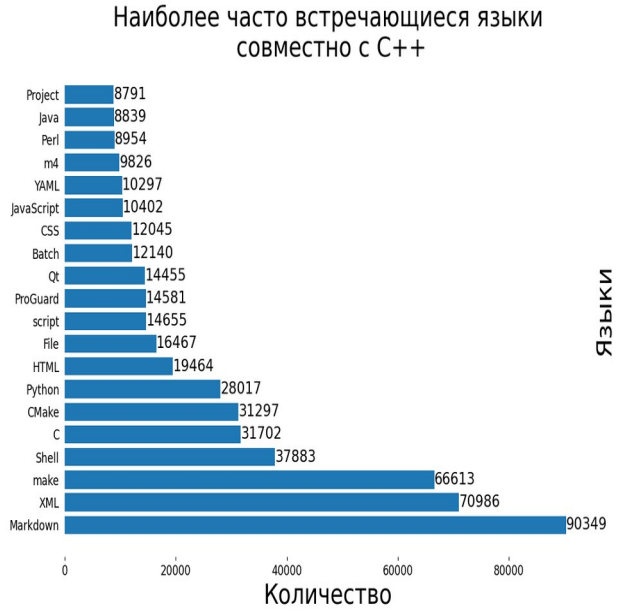

Рисунок 12 – Наиболее популярные языки совместно с C++

F. *Процентное отношение популярных систем сборки*

Проанализировав содержимое репозиторев на наличие самым популярных систем сборки cmake, make, qmake, meson, ms build system рис. 13, было получено процентное соотношение в котором на первом месте оказалась система сборки make, которая считается достаточно неудобной и сложной.

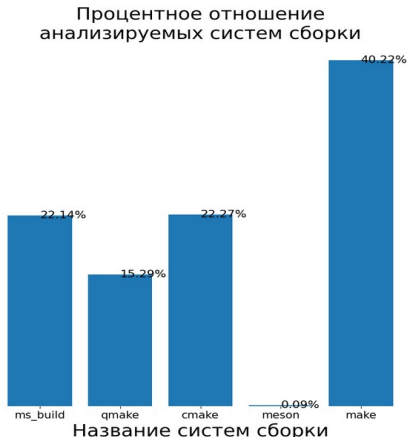

Рисунок 13 – Процентное отношение рассматриваемых систем сборок

G. Самые популярные названия веток и наличие gitflow

Выполнив анализ репозиториев на наименования веток рис. 14, название master оказалось на первом месте.

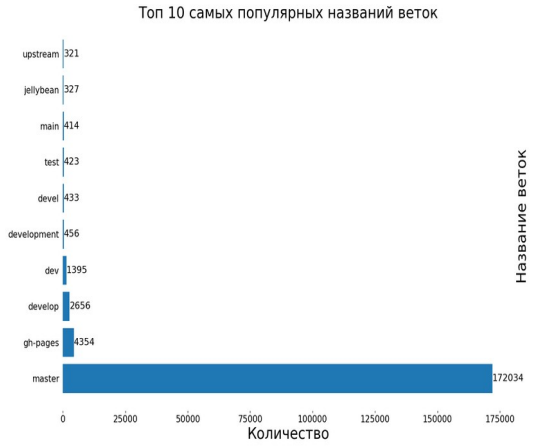

Рисунок 14 – Самые популярные названия веток

Наличие комбинации веток, которое используется в gitflow рис. 15 оказалось у относительно малого числа репозиториев.

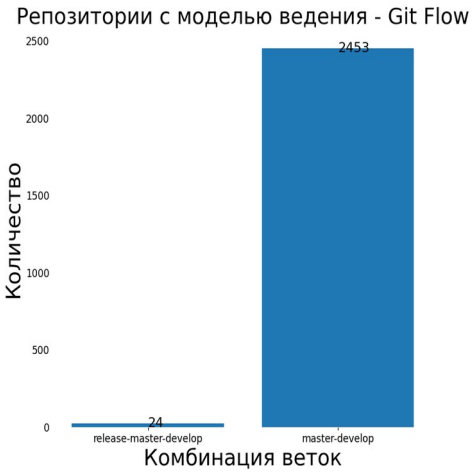

Рисунок 15 – Модель ветвления gitflow в репозиториях

Список литературы

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” In CVPR, 771–777, 2016.

[2] O. Jarczyk, B. Gruszka, S. Jaroszewicz, “Quality analysis of open-source software”, SocInfo 2014, LNCS 8851, pp. 80–94, 2014.

[3] E. Guzman, D. Azocar, Y. Li, “Sentiment Analysis of Commit Comments in GitHub: An Empirical Study,” In MSR 2014.

[4] Y. Hu, J. Zhang, X. Bai, S. Yo and Z. Yang, “Influence analysis of Github repositories,” SpringerPlus 5, 2016.

[5] M. Beller, G. Gousios, A. Zaidman, “Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub,” In MSR 2017.

[6] P. Russell, R. Johnson, S. Ananthan, B. Harnke and N. Carlson, “A large-scale analysis of bioinformatics code on GitHub,” PLoS ONE 13, 2018.

[7] J. Cito, G. Schermann, J. Wittern, P. Leitner, S. Zumberi, H. Gall, “An Empirical Analysis of the Docker Container Ecosystem on GitHub,” In MSR, 2017.