



MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

CNG491 Computer Engineering Design I
Final Report

Project Name: Git Assessor - Automated Assessment of Git Usage of
Students

Project Participants:

Ziya Taner Keçeci 2152049

Yusuf Mete Kabakçı 2151983

Doğukan Çatal 2257996

Mahmut Ali Şahin 2243673

Table of Content

1.Introduction	4
1.1 Problem Formulation & Background	4
1.2 Motivation	4
1.3 Aims and Objectives	5
Aim	5
Objective	5
1.4 Stakeholders	6
2.Requirements Specification	6
2.1 Functional Requirements	6
2.2 Non-Functional Requirements	7
2.3 Assumptions and justifications.....	7
2.4 Structured use case diagrams	7
2.5 High level sequence diagrams	8
2.6 Interfaces definitions with external systems	9
2.7 Graphical user interface	9
2.7.1 Home Page	9
2.7.2 Assessment Report Page.....	9
3.System Architecture and Models.....	10
3.1 Architectural model.....	10
3.2Process model.....	11
3.2.1 Application Perspective.....	11
3.2.2 Service Perspective	12
3.3 Data flow models.....	13
3.4 List of required functions realising the subsystems and their definitions	14
3.4 List of special data structures and required data types.....	15
3.6 List of special algorithms needed for the system.....	16
3.7 List of interfaces definitions	16
4 Project Management Section.....	17
4.1 Software estimations.....	17
4.1.1Project Estimation with COCOMO Model.....	17

4.1.2 Project Estimation with Jones's Model	18
4.2 tasks and their durations	19
4.3 List of the milestones for this semester	19
4.4 Activity diagram	20
4.5 Gantt chart.....	20
5. Conclusion	21
5.1 Critical evaluation of the project and conclusions reached	21
5.2 Retrospective of the first semester	21
5.3 Future work	21

1.Introduction

1.1 Problem Formulation & Background

With this project, we will try to help instructors and students to evaluate GitHub projects efficiently. Our Project is based on an academic paper, “Errors and Poor Practices of Software Engineering Students in Using Git” written by our advisor Şükrü Eraslan and his colleagues from Manchester University. As they have pointed out in detail in that paper, they have created a table which consists of “poor practices of students in their git usage.” Those poor practices drive students to develop their coding skills poorly. That’s why we are developing a web service to evaluate the GitHub project of the students based on those poor practices.

1.2 Motivation

Git knows how to process the contents of a snapshot, one for each commit, and apply or rollback a change set between the two snapshots. During development, your project has several save points called commits. The commit history contains all the commits, that is, the changes made in the project during development. Commits allow you to roll back or fast forward your code to any commit in the commit history. A Git branch is a simple move pointer to one of these commits. The default branch name for Git is master. When you start a commit, you get a master branch that points to the last commit you made. The master branch pointer automatically advances each time you commit.

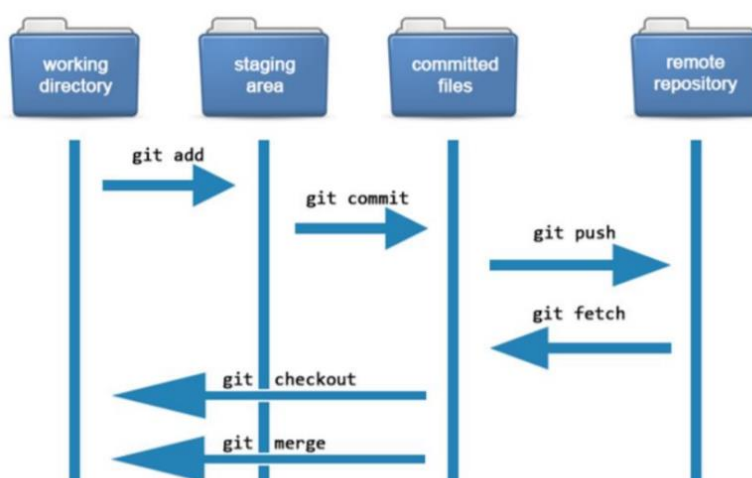


Figure 1 Basic Git Flow (Marjian 2021)

As cited in [Lawrance et al., 2013]. “Students often made multiple commits off-line between pushes, and students on teams who pushed to a single remote repository (adopting a centralized workflow) encountered more difficulty than those who pushed to their private forks and merged later (adopting a distributed workflow).” Let’s give some examples for poor practices made by students. “Students commit their changes with uninformative messages.”, “Students commit code that does not compile or does not pass the tests”, “Students do not follow the branch/tag naming convention.”, etc. [Eraslan et al., 2020].

Our project will let instructors to choose some rules which will be added by our team to the app so that instructors will be able to get an evaluation/assessment report of their students’ projects. Creating a Web Application to help instructors to assess their students’ performances is our motivation. As a result of that improvement of students will be faster and efficient.

1.3 Aims and Objectives

Aim

We aim to improve the usability and desirability of GitHub(a web-based control version system using Git) for students by identifying the poor practices and misuse of Git.

Objective

Our first objective is to design a user-friendly GUI. In other words, a GUI that is easy to understand and easy to use. Another object of ours is fetching the data from GitHub in an efficient way. Meaning, we will avoid creating busy network traffics between our web service and GitHub API by establishing efficient network connections. Our third object is manipulating and processing the fetched data to detect poor practices in a short amount of time. Our fourth goal is to evaluate the students' performances in a logical and meaningful way to generate an assessment grade for each student. Our last objective is to create an easy-to-understand report with that generated data.

1.4 Stakeholders

1. Teacher/Teaching Assistant
2. University Student

2.Requirements Specification

2.1 Functional Requirements

1- An instructor shall be able to provide GitHub repository links to system.

- A text box will be provided as an input for instructors to enter their desired repository links.

2- An instructor shall be able to delete any repository link from the list before getting the report.

- The instructor can use the check boxes to select his/her desired repository links to remove/delete those links by pressing to the remove button.

3- An instructor shall be able to choose which rules will be used for assessment.

- Check boxes will be provided with each rule and the instructor can choose any rule that he/she wants to check whether students obeyed or not.

4- An instructor shall be able to get an assessment report for repositories and participants.

- Assess button will be provided for the instructor to get an assessment report when the repository links and the desired rules are selected.

5- For the rule set, the system shall provide the list of rules and instructors shall be able to configure them.

6- Assessment report will provide Visual representation of the data. Such as Bar charts of student performances.

- A new page will be provided to let the instructor to observe the students' performances, mistakes, and grades. For easier reading some of those data will be provided as charts (eg. Bar chart, ...).

2.2 Non-Functional Requirements

- 1- The System should be available 7/24 for the users.
1. 2-The System should make analysis quickly and prepare reports in a short time to ensure user satisfaction.
- 2- The System should be easy to use which is desirable for the users.
- 3- The system should have a simple GUI to ensure users can have done his/her job without spending unnecessary time

2.3 Assumptions and justifications

- 1- We assume repositories should be public, because GIT doesn't allow receiving data from private repositories.
- 2- We assume project repositories will not be huge so that GitHub doesn't cut off the connection.
- 3- We assume students' consents have been taken to be assessed by their performance. Otherwise, it will be violation of ethics.
- 4- To calculate Jones's First-Order Effort Estimation and Schedule Estimation, we assume as we are working as shrink-wrap software organization. 5- According to [Prechelt, 2000] Python and Pearl has strong similarities. Thus, we accepted LOC/FP value of Python as 20.

2.4 Structured use case diagrams

In figure 2, we provide Structured Use Case Diagram for our Git Assessor Web Application while User, GitHub (Git API), Git Assessor Web Service are actors. User can analyze either a single or multiple repositories. To analyze repositories some rules must be selected by user and then "Rule Set Check" checks those selected rules to visualize a result which is an assessment report. "Git Assessor Web Service" handles the "Rule Set Check" by communicating with "GitHub" using GitHub API.

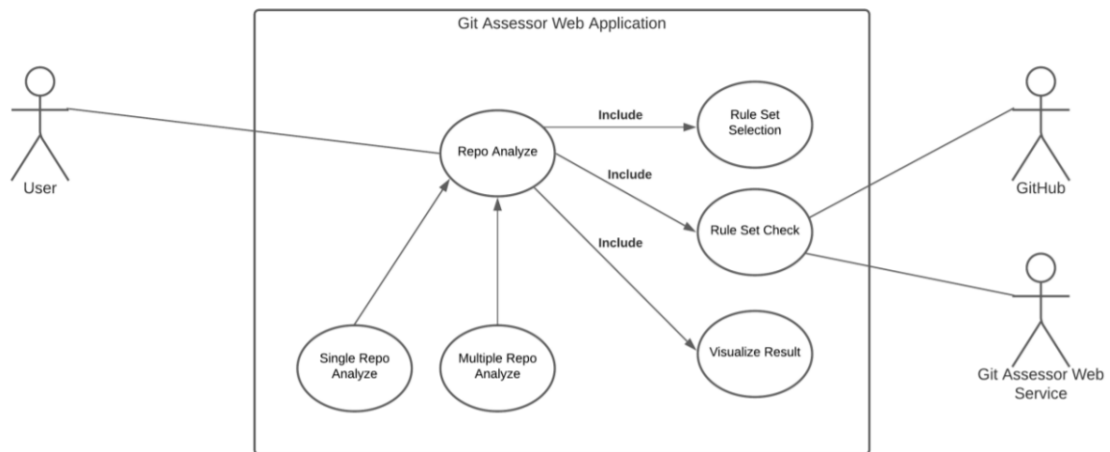


Figure 2 Structured Use Case Diagram

2.5 High level sequence diagrams

In Figure 3, we provide Sequence Diagram for our major use case. Git Assessor Web Application is getting all the repos and rules from user in one time but asking Git Assessor Web Service about repositories once at a time.

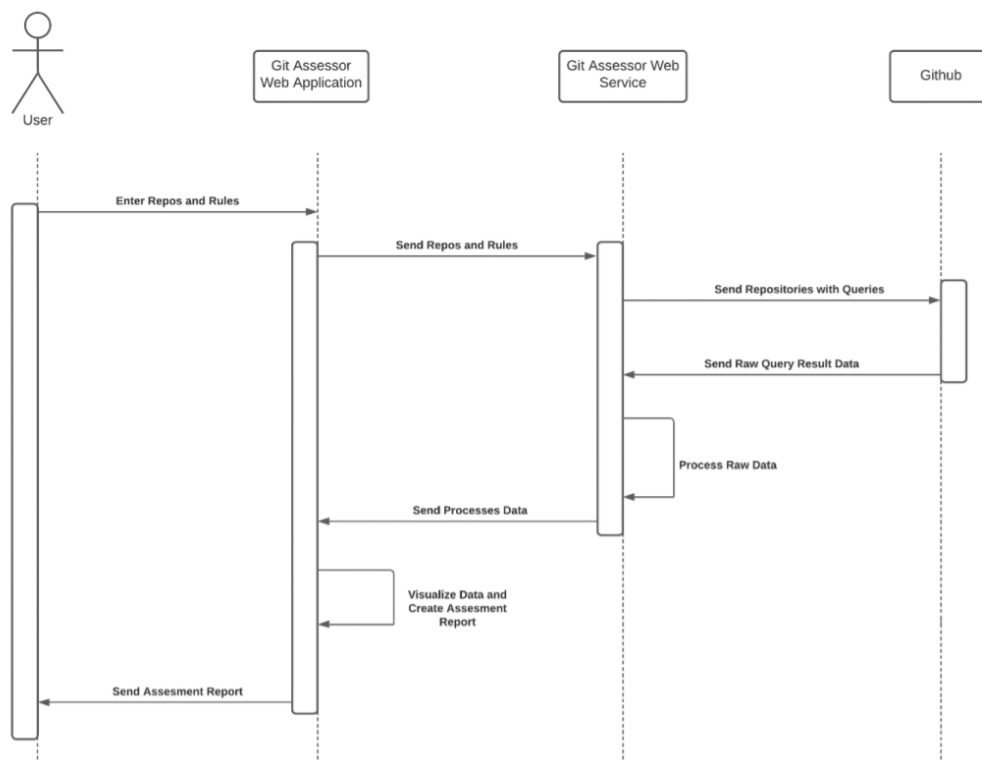


Figure 3 Sequence Diagram

2.6 Interfaces definitions with external systems

- FastAPI
- PyGitHub

Both interfaces will be explained in detail in “List of interfaces definitions” section.

2.7 Graphical user interface

2.7.1 Home Page

Figure 4 shows the “Home page” when the user enters the site. In this page user can enter the repository links to the URL box (where it says “Repo Link”). When the user pastes the desired repository links and clicks to “Add” button those links will be added to the “list of added links”. If user has a change of mind and wants to remove some links, it can be done by clicking to the checkbox next to the links and then pressing to the “Remove” button. There is a list of rules under the “Rules” title, desired rules to be checked by the web service for each link in the list can be selected by clicking to the checkbox next to each rule. After everything is done, user can press to the “Assess” button to open “Assessment Report Page” (Figure 5).

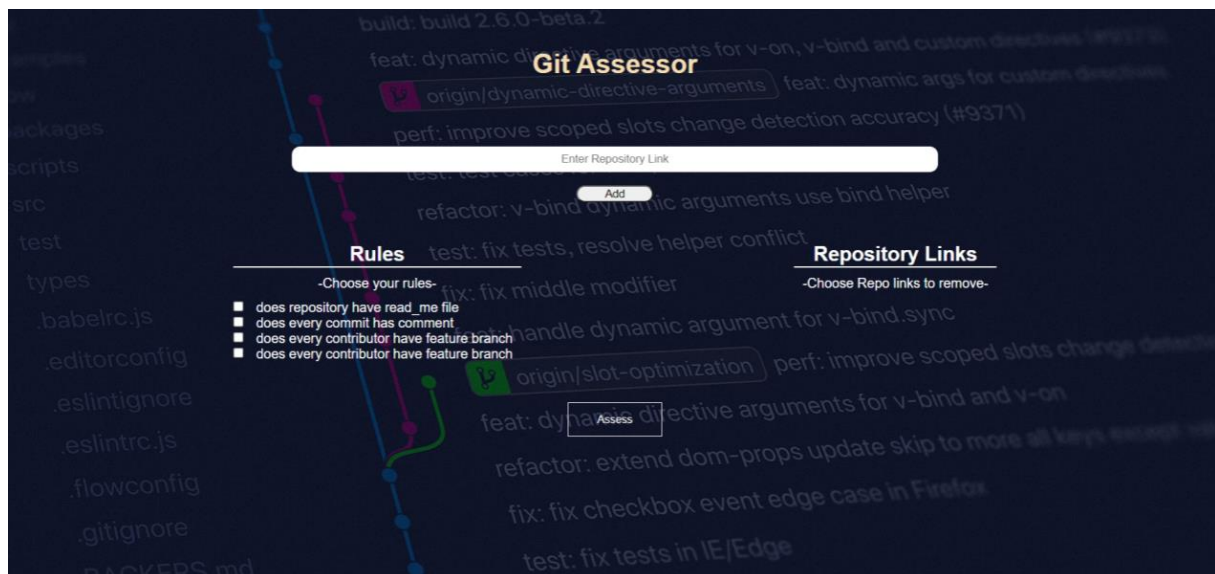
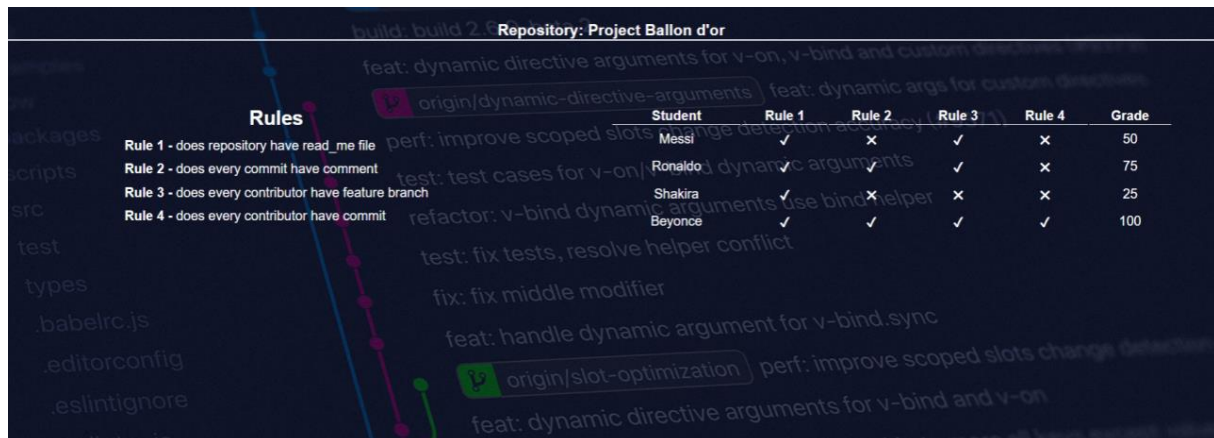


Figure 4 GUI Home Page

2.7.2 Assessment Report Page

Figure 5 shows the “Assessment Report Page” which is created when the “Assess” button is clicked in “Home Page” (Figure 4). This page shows the results of checking rules for each repo links by the web service for user to analyze. In the report, selected rules by the user can be seen and each participant student for that repository is graded by whether they have obeyed to that specific rule or not. As the report goes on, mistakes that have been made can be seen. In the future we are also planning the visualize the results for each student (Using pie chart etc.).



Rules	Student	Rule 1	Rule 2	Rule 3	Rule 4	Grade
Rule 1 - does repository have read_me file	Messi	✓	✗	✓	✗	50
Rule 2 - does every commit have comment	Ronaldo	✓	✓	✓	✗	75
Rule 3 - does every contributor have feature branch	Shakira	✓	✗	✗	✗	25
Rule 4 - does every contributor have commit	Beyonce	✓	✓	✓	✓	100

Figure 5 Assessment Report Page

3.System Architecture and Models

3.1 Architectural model

In Figure 6, we provide Architectural Model. Git Assessor Web Service contains GitHub API and Rule Checker. Git Assessor Application contains Rule Selector (lets user to select desired rules) and Report Generator. Git Assessor Application sends rules and links to the Git Assessor Web Service and receives result (data) from Git Assessor Web Service to create a report. Git Assessor Web Service sends/receives data to/from GitHub and Git Assessor Application (Web Application). GitHub Lets Git Assessor Web Service to retrieve desired data via GitHub API.

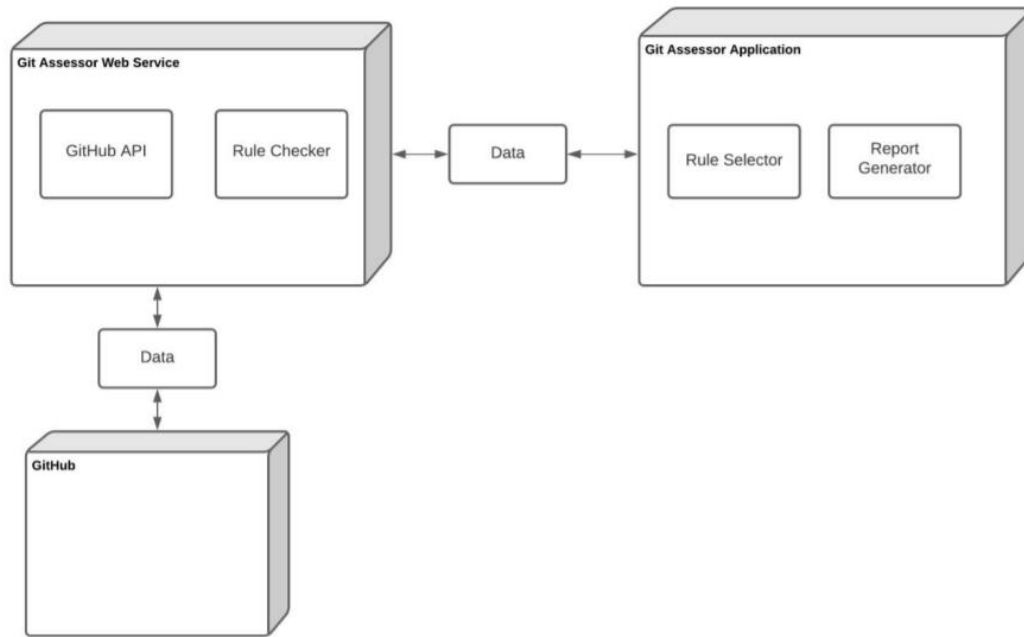


Figure 6 Architectural model

3.2 Process model

3.2.1 Application Perspective

Application process will start with asking for a Repo Link to be entered. If the user wants to add more links, he/she will be able to do that. After adding all desired links, the user will need to select rule(s). When all links and rules are selected, links and rules will be sent to the web service. When the result is retrieved from the web service, process will end by creating a report with those retrieved results.

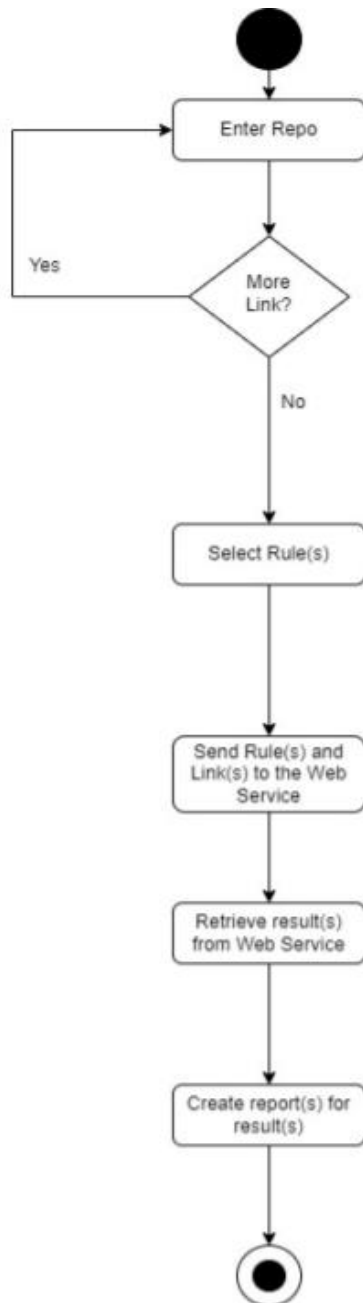


Figure 7 Process Model Application Perspective

3.2.2 Service Perspective

Service process will start with getting repository links and selected rules from the application. Process will continue by retrieving data of repository links from GitHub via GitHub API. Then, selected rules will be checked to create a result and afterwards, When the results are created for each link, process will end by sending those results to the application.

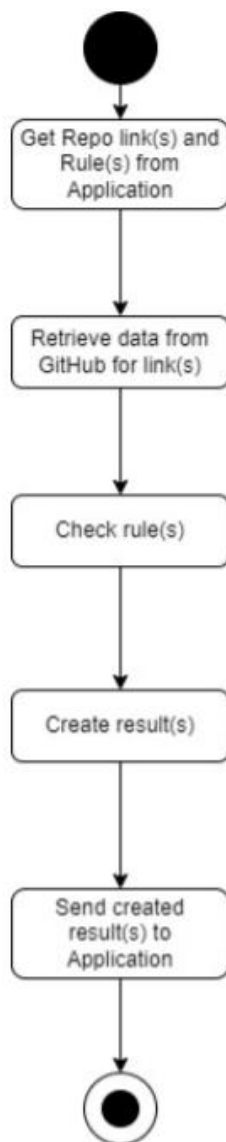


Figure 8 Process Model Service Perspective

3.3 Data flow models

Our project consists of 2 parts Git Assessor Web application and Git Assessor Web Service. The user enters the rules and repository links to the web application. The web application uses that data, communicating with GitHub and fetched the necessary repository information. Then, using this repository data and rules, Web services process the data for contributors and repositories. Then this processed data is used to generate an assessment report. In this process, we also need to assess a grade for each student. After that, the web service sends the assessment report and grades to the web application.

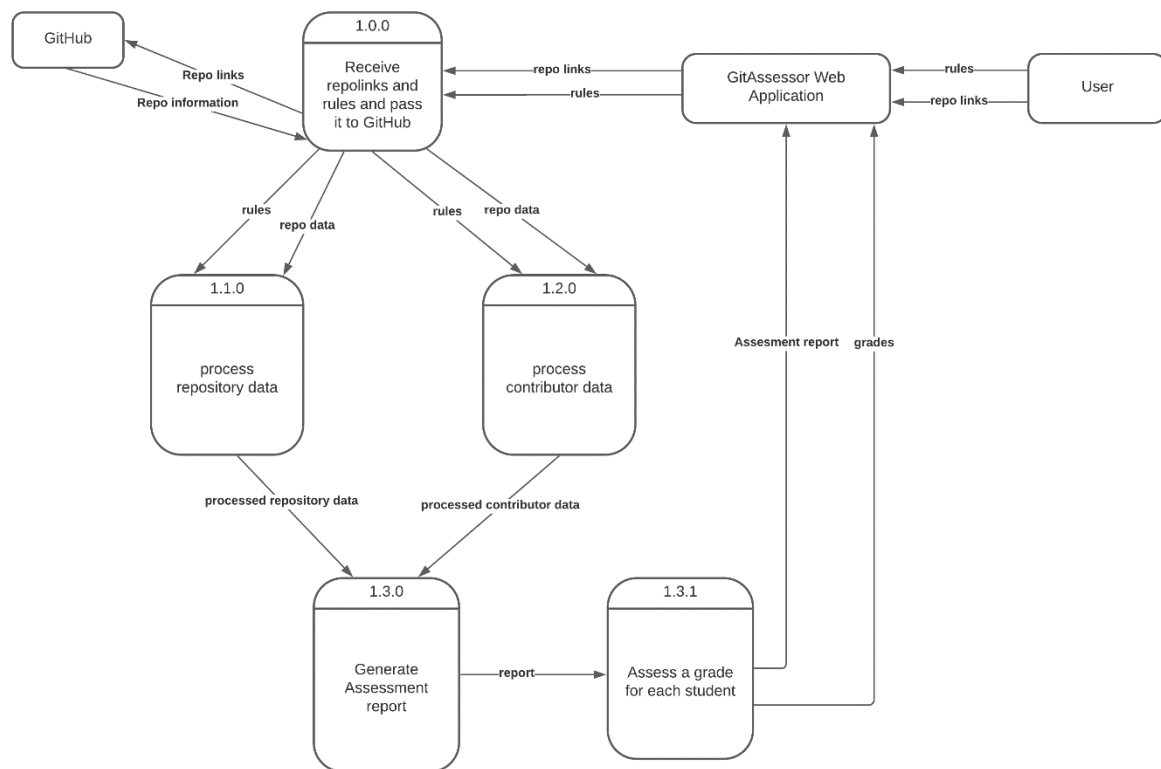


Figure 9 Data Flow Model

3.4 List of required functions realising the subsystems and their definitions

1- "check_links_service"

Creating an array of repository links. Using that array to create GitHub access token to reach statistics of repositories. Getting contributors, commits, branches, repository names and sending that information to rule checking algorithms.

2- Functions to check "poor practices"

2-1 "all_commit" This function checks if every contributor has at least 1 commit or not. If a contributor has 0 commits this function will return false for that contributor otherwise it will return true for that contributor.

2-2 "commit_comments" This function checks if every commit has commented with it. When a user does a commit without comment we accept it as poor practice. Thus this function returns false for those contributors who don't comment on their commits.

2-3 "feature_branch" This function returns true if every contributor has a branch for himself/herself to avoid working on the main branch. If a contributor didn't create a branch for his/her own developing process this function will return false for those contributors

2-4 "readme_rule" This function checks the documents of the repository to detect a readme file. The accepted format of a readme file is ".md" or ".txt". If a repository doesn't have a readme file in one of these extensions the function returns false, otherwise it returns true

3.4 List of special data structures and required data types

1- Data Structures:

1.1- Array(s)

1.1.1- We are getting 5 repository links from front-end and placing those links to an array.

1.1.2- We are getting a rule set from front-end and placing those rules in an array

1.2- Dictionary

1.2.1- We are creating a dictionary for every repository. In this dictionary we have repository name, rules and repository names.

1.2.2- We are creating another dictionary which consists of repository dictionaries. In the end this dictionary will be a nested dictionary.

1.3- PaginatedList

1.3.1- All contributors, commits and branches are created as PaginatedList by PyGithub API.

1.4- JSON

1.4.1- We are sending the nested dictionary from back-end to front-end in JSON format.

2- Data Types:

2.1- Strings

2.1.1- We are using repository links as string to manipulate and send it to PyGithub.

2.1.2- We are using access_token as string to send it to PyGithub

2.1.3- We are using contributor names as string to send it to front-end.

2.2- Boolean

2.2.1- For "poor practice" checking algorithms we are using boolean results as return value.

Figure 10 Data structures and Data Types

3.6 List of special algorithms needed for the system.

Our programs don't use any special algorithm.

3.7 List of interfaces definitions

In this project we used FastAPI. FastAPI is used to create communication between front-end users and back-end users; on the other hand, PyGithub library of Python is used for communication with Github. Let's take a look at FastAPI first. The key feature of the FastAPI is completing a lot of operations faster than others. "FastAPI is a modern, fast (high-performance), a web framework for building APIs with Python 3.6+ based on standard Python type hints." (<https://fastapi.tiangolo.com/>). In general, we used FastAPI to add end-points to our web service. We also used the PyGithub library to communicate with Github. There are special functions in PyGithub that help us to reach data in Github; for instance, the name of the contributors or the name of the branches in the project.

4 Project Management Section

4.1 Software estimations

4.1.1 Project Estimation with COCOMO Model

Program Characteristics	Low Complexity	Medium Complexity	High Complexity
Number of inputs	$14 \times 3 = 42$	$1 \times 4 = 4$	$0 \times 6 = 0$
Number of outputs	$2 \times 4 = 8$	$0 \times 5 = 0$	$5 \times 7 = 35$
Inquiries	$5 \times 3 = 15$	$0 \times 4 = 0$	$0 \times 6 = 0$
Logical Internal files	$1 \times 7 = 7$	$0 \times 10 = 0$	$5 \times 15 = 75$
Unadjusted function-point total			236
Influence Multiplier/VAF			1.13
Adjusted Function-point Total			289.28

Figure 11 COCOMO Model

General System Characteristics

No	GSC	DI
1	Data Communications	5
2	Distributed data processing	5
3	Performance	3
4	Heavily used configuration	1
5	Transaction Rate	5
6	On-Line data entry	2
7	End-user efficiency	5
8	On-Line update	0
9	Complex Processing	4
10	Re-usability	5
11	Installation ease	0
12	Operational ease	5
13	Multiple sites	5
14	Facilitate change	3

Figure 12 General system characteristics

Brandon Simplified Term	Total Number	Multiplier	Total Functional Points
Number of Places we input values in	15	Times 3	45
Number of places read only	5	Times 4	12
Database queries used to produce simple information	0	Times 3	0
Tables in AX for main query	0	Times 7	0
Outside data sources for information	1	Times 5	5
Business Rules	XXXXXXXX	XXXXX	XXXXXXXXXXXXXXXX
Total functional points			70

Total Degree of Influence = 48
 Value Adjust Factor (VAF) = 1.13
 $(TDI \times 0.01) + 0.65 = 48 \times 0.01 + 0.65$
 Function Point Count = $70 \times 1.13 = 79.1$

$$KDSI = \frac{289.28 \times 20}{1000} = 5.7856$$

Development Mode is Organic:

a = 2.4, b = 1.05, c = 0.38

$$Man - Month = 2.4 \times 5.7856^{1.05} = 15.16$$

$$TDEV = 2.5 \times 15.16^{0.38} = 7.02$$

Figure 13 : Calculation of TDEV and other parameters

4.1.2 Project Estimation with Jones's Model

Kind of Software	Best in Class	Average	Worst in Glass
Systems	0.43	0.45	0.48
Business	0.41	0.43	0.46
Shrink-wrap	0.39	0.42	0.45

Our ATFP is 289.28

$$Estimated\ Effort\ (man - month) = ((289.28)^{3 \times 0.42}) / 27 = 46.76$$

$$Rough\ Schedule\ Estimate = (289.28)^{0.42} = 10\ months$$

Figure 14 Jones's Model

4.2 tasks and their durations

- Complete the advanced level GUI
 - 01/12/2021 - 25/01/2022
- Calculate user-base & repo-base grade of repositories to be shown in tables.
 - 01/12/2021 - 25/01/2022

Estimated duration in terms of weeks :

- Complete the advanced level GUI
 - 01/12/2021 - 25/01/2022
 - Task Dependency :
 - Basic GUI should be completed first.
 - Basic web-service operations should be working correctly.
- Calculate user-base & repo-base grade of repositories to be shown in tables.
 - 01/12/2021 - 25/01/2022
 - Task Dependency :
 - Basic web-service functions should be working correctly.
 - Calculation methods of grades should be decided.

Who are responsible for each task :

- Complete the advanced level GUI
 - Yusuf Mete Kabakçı
 - Doğukan Çatal
- Calculate user-base & repo-base grade of repositories to be shown in tables.
 - Ziya Taner Keçeci
 - Mahmut Ali Şahin

4.3 List of the milestones for this semester

List of milestones for this semester :

- Complete the advanced level GUI
 - User interface should be tested and improved.
 - Tables that show result of operations should be added
- Calculate user-base & repo-base grade of repositories to be shown in tables.
 - Rule functions should be improved with detailed calculations.
 - New type of JSON values that has both repository name, grade and rules should be created

4.4 Activity diagram

Our milestones are divided for back-end and front-end, so that they work in parallel. On the other hand, our tasks are mainly focused on understanding the topic and preparing for the milestones.

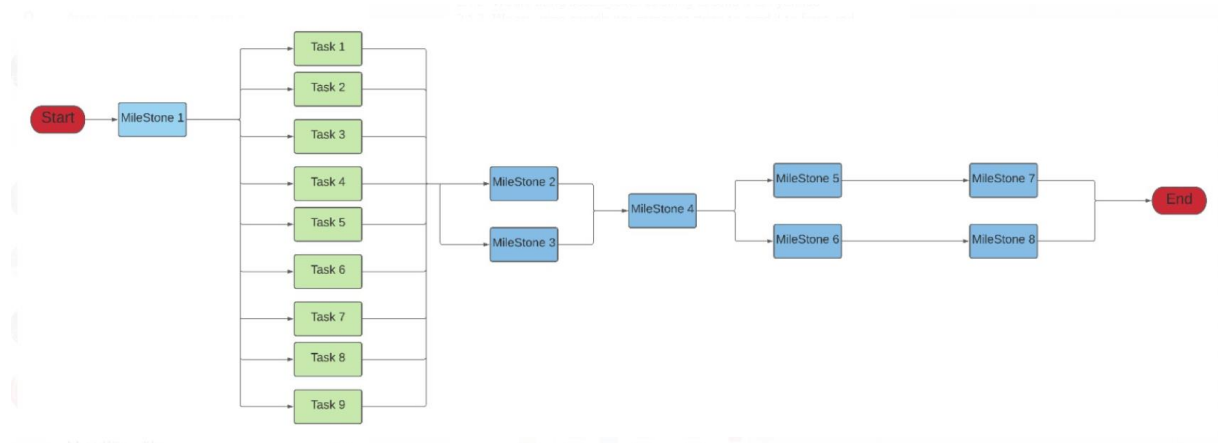
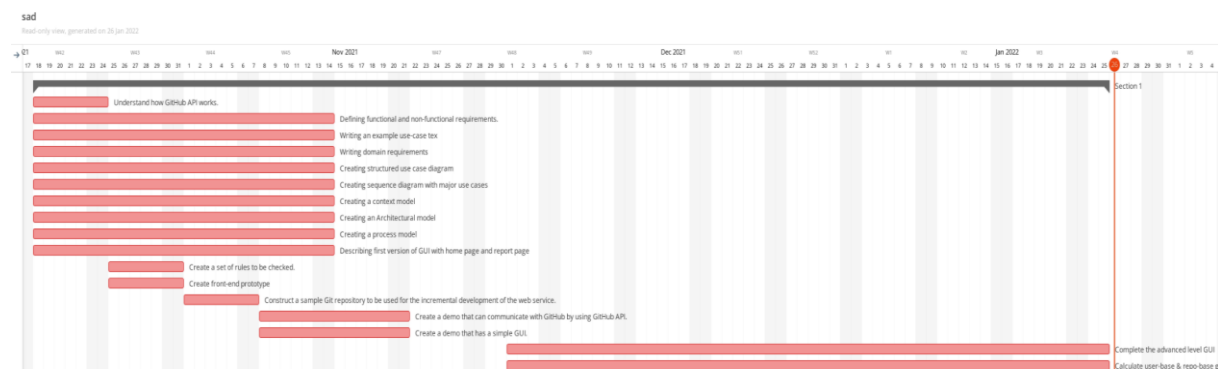


Figure 15 Activity Diagram

4.5 Gantt chart



5. Conclusion

5.1 Critical evaluation of the project and conclusions reached

For the first semester, we have completed most of our goals. As a group, we are okay with what we have done until now and we will continue to improve our program day by day. Our program has a user-friendly user interface, for now, we couldn't add detailed information of mistakes in the analysis part. The complexity of the algorithms has been sorted to make the program run as fast as possible. The main aim was to let the user select desired rules and assess/analyze more than one repository at the same time. We have reached this main aim. More improvements will be done, and we developed the program in a way that we can do these improvements pretty easily as long as there are no unexpected errors.

5.2 Retrospective of the first semester

In the first semester, we learned how to work with each other, and we completed the main parts of our project in success. At the beginning of the semester, we arranged a meeting to get to know our teammates better. These types of meetings were followed by each other throughout the semester. To start the completion process of our project, we have begun with role division. Each one of us talked about what he really wanted, and we decided on roles according to these demands. When we started the completion process, both parts of the team created their own Gantt chart, these charts were combined and served in the report as a one-piece. While completing our tasks, we both learned and taught; moreover, we understood that we chose the right topic. We submitted our reports and took feedback from Dr. Enver Ever, then we completed our corrections on our report based on his feedback.

5.3 Future work

In the near future, we are planning to add a graphical representation for the user interface to show contributors' points clearly. This will improve the user's user-interface experience. In addition to these plans, we planned to add more rules. Completing real-life tests and sharing this project with society are parts of our plan also. Moreover, we are planning to create a real-time database to keep track of mistakes. In that way, we can compare users with other users to show better understanding.