Analytics - Product Quality

Date: 2020/02

SUMMARY:

- This notebook represents the project quality analysis of the date exposed right above.

TEAM:

Semester: 2023/01

Professor: Hilmer Neri

Members:

- Aline Lermen
- Caio Martins
- Dafne Moretti
- João Pedro
- João Vitor
- Lucas Gabriel
- Lucas Lima
- Eric Camargo
- Ester Flores
- Leonardo Ferreira
- Luana Torres
- Matheus Ferreira
- Pablo Guilherme
- Pedro Izarias
- Pedro Sena
- Suzane Duarte

Data de análise: [R1] - 26/05/2023 (Sprint 04)

LIBRARIES

In [14]:

```
# Deal with data
import pandas as pd
import json
from glob import glob
import os

# Deal with visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Deal with type hints
from typing import List

# Deal with time
import datetime
```

GRAPH SETTINGS

In [15]:

```
%config InlineBackend.figure_format ='retina'
sns.set(font_scale=1.5)
sns.set_style('darkgrid',
              {'xtick.bottom' : True,
               'ytick.left': True,
               'grid.linestyle':'--',
               'font.monospace': ['Computer Modern Typewriter'],
               'axes.edgecolor' : 'white'})
```

DATAFRAME SETTINGS

In [16]:

```
pd.set_option("display.max_rows", None, "display.max_columns", None)
```

Replace your semester, project name, repository name, and the programming language extension

In [17]:

```
# Set your repo major name here
# Example: fga-eps-mds-2022-1-MeasureSoftGram-
repo_name = 'fga-eps-mds-2023-1-Alectrion-'

# Add your repos here
# Example: 'Front': 'py',
repos_language = {
    'EquipamentApi': 'ts',
    'Gateway': 'ts',
    'UserAPI': 'ts',
    'FrontEnd': 'ts',
}
```

SonarCloud

Path to the folder with all your jsons

In [18]:

```
# Maybe you should change this path to your own path

sonar_files = glob('../analytics-raw-data/*.json')
```

Create DataFrame

Unmarshall json

In [19]:

```
def unmarshall(json_path: str) -> dict:
    with open(json_path) as json_file:
        json_obj = json.load(json_file)
    return json_obj
```

Create a list with all valid columns

In [20]:

```
metric_list = ['files',
               'functions',
               'complexity',
               'comment_lines_density',
               'duplicated_lines_density',
               'coverage',
               'ncloc',
               'tests',
               'test_errors',
               'test_failures',
               'test_execution_time',
               'security_rating']
```

Extract files dataframe out of component dataframe

In [21]:

```
def get_files_df(df: pd.DataFrame) -> pd.DataFrame:

    files_df = df[df['qualifier'] == 'FIL']

    files_df = files_df.dropna(subset=['functions', 'complexity','comment_lines_den

    return files_df
```

Extract directories dataframe out of component dataframe

In [22]:

```
def get_dir_df(df: pd.DataFrame) -> pd.DataFrame:
    dirs = df[df["qualifier"] == "DIR"]

    newdf = pd.to_numeric(dirs["tests"])

    max_value_index = newdf.idxmax()

    return dirs.loc[max_value_index]
```

Extract uts dataframe out of component dataframe

In [23]:

```
def get_uts_df(df: pd.DataFrame) -> pd.DataFrame:
    uts_df = df[df['qualifier'] == 'UTS']

    uts_df = uts_df.fillna(0)

    uts_df = uts_df.dropna(subset=['test_execution_time'])

    return uts_df
```

Generate component dataframe

In [24]:

```
def metric_per_file(json_dict: dict) -> List[dict]:
    file_json = []

    for component in json_dict['components']:
        ncloc_value = 0
        for measure in component['measures']:
            if measure['metric'] == 'ncloc':
                ncloc_value = float(measure['value'])
                break

        if (component['qualifier'] == 'FIL' and ncloc_value > 0) \
                or component['qualifier'] == 'DIR' \
                or component['qualifier'] == 'UTS':
            file_json.append(component)

    return file_json


def generate_component_dataframe_data(
        metrics_list: List[str],
        file_component_data: List[dict],
        language_extension: str) -> pd.DataFrame:

    df_columns = metrics_list

    files_df = pd.DataFrame(columns = df_columns)
    dirs_df = pd.DataFrame(columns = df_columns)
    uts_df = pd.DataFrame(columns = df_columns)

    for file in file_component_data:
        try:
                if file['qualifier'] == 'FIL' and file['language'] == language_exte
                    for measure in file['measures']:
                        files_df.at[file['path'], measure['metric']] = measure['valu

                    files_df['qualifier'] = file['qualifier']

                elif file['qualifier'] == 'DIR':
                    for measure in file['measures']:
                        dirs_df.at[file['path'], measure['metric']] = measure['valu

                    dirs_df['qualifier'] = file['qualifier']

                elif file['qualifier'] == 'UTS':
                    for measure in file['measures']:
                        uts_df.at[file['path'], measure['metric']] = measure['value

                    uts_df['qualifier'] = file['qualifier']

        except:
            pass

    files_df.reset_index(inplace = True)
    dirs_df.reset_index(inplace = True)
    uts_df.reset_index(inplace = True)

    files_df = files_df.rename({'index': 'path'}, axis=1).drop(['files'], axis=1)
    dirs_df = dirs_df.rename({'index': 'path'}, axis=1).drop(['files'], axis=1)
    uts_df = uts_df.rename({'index': 'path'}, axis=1).drop(['files'], axis=1)
    df = pd.concat([files_df, dirs_df, uts_df], axis=0)

    return df


def create_component_df(json_list):
    df = pd.DataFrame()

    for json_path in json_list:
        file_component = unmarshall(json_path)
        file_component_data = metric_per_file(file_component)

        base_name = os.path.basename(json_path)

        file_component_dataframe = generate_component_dataframe_data(
            metric_list,
            file_component_data,
            language_extension = repos_language[base_name.split("-")[6]])


        file_component_dataframe['filename'] = base_name

        df = pd.concat([df, file_component_dataframe], ignore_index=True)

    aux_df = df['filename'].str.split(r"-(\d+-\d+-\d+-\d+-\d+-\d+)-(.*?).json", exp

    df['repository'] = aux_df[0]
    df['datetime'] = aux_df[1]
    df['version'] = aux_df[2]

    df = df.sort_values(by=['repository', 'datetime'])

    return df
```

In [25]:

```
file_component_df = create_component_df(sonar_files)
file_component_df.repository.unique()
```

Out[25]:

```
array(['fga-eps-mds-2023-1-Alectrion-EquipamentApi',
       'fga-eps-mds-2023-1-Alectrion-FrontEnd',
       'fga-eps-mds-2023-1-Alectrion-Gateway',
       'fga-eps-mds-2023-1-Alectrion-UserAPI'], dtype=object)
```

Create dataframe per repository

In [26]:

```
repos_dataframes = []

for repo in repos_language.keys():
    dataframe = file_component_df[file_component_df['repository'] == repo_name+repo
    repos_dataframes.append({'name': repo, 'df': dataframe})
```

Measure calculations according to Q-Rapids quality model

In [27]:

```
def _ncloc(df: pd.DataFrame) -> int:
    ncloc = 0
    for each in df['ncloc']:
        # try to cast the current ncloc value to int, if the value is NaN/Null, con
        try:
            n = int(each)
        except ValueError:
            n = 0
        ncloc += n

    return ncloc
```

Quality Aspect - Maintainability

Factor - Code Quality

Complexity

In [28]:

```
def m1(df: pd.DataFrame):

    files_df = get_files_df(df)

    density_non_complex_files = len(files_df[(files_df['complexity'].astype(float)
                                          files_df['functions'].astype(float))

    return density_non_complex_files
```

Comments

In [29]:

```
def m2(df: pd.DataFrame):

    files_df = get_files_df(df)

    density_comment_files = len(files_df[(files_df['comment_lines_density'].astype(
                                      (files_df['comment_lines_density'].astype(

    return density_comment_files
```

Duplications

In [30]:

```
def m3(df: pd.DataFrame):

    files_df = get_files_df(df)

    duplication = len(files_df[(files_df['duplicated_lines_density'].astype(float) 

    return duplication
```

Quality Aspect - Reliability

Factor - Testing Status

Passed tests

In [31]:

```
def m4(df: pd.DataFrame):

    dir_df = get_dir_df(df)

    passed_tests = (float(dir_df['tests']) - (float(dir_df['test_errors']) + float(
               float(dir_df['tests'])

    return passed_tests
```

Fast test builds

In [32]:

```
def m5(df: pd.DataFrame):
    dir_df = get_uts_df(df)

    density_fast_test_builds = len(dir_df[(dir_df['test_execution_time'].astype(flo
                               len(dir_df['test_execution_time'].astype(float))
    return density_fast_test_builds
```

Test coverage

In [33]:

```
def m6(df: pd.DataFrame):

    files_df = get_files_df(df)

    density_test_coverage = len(files_df[(files_df['coverage'].astype(float) > 60)]

    return density_test_coverage
```

Calculate measures for each repository

In [34]:

```
def create_metrics_df(df: pd.DataFrame) -> pd.DataFrame:

    date_time_vec = df['datetime'].unique()

    m1_list = []
    m2_list = []
    m3_list = []
    m4_list = []
    m5_list = []
    m6_list = []

    ncloc_list = []
    repository_list = []
    version_list = []

    for version in date_time_vec:

        version_df = df[df['datetime'] == version]

        try:
            m1_list.append(m1(version_df))
        except Exception:
            m1_list.append(0)

        try:
            m2_list.append(m2(version_df))
        except Exception:
            m2_list.append(0)

        try:
            m3_list.append(m3(version_df))
        except Exception:
            m3_list.append(0)

        try:
            m4_list.append(m4(version_df))
        except Exception:
            m4_list.append(0)

        try:
            m5_list.append(m5(version_df))
        except Exception:
            m5_list.append(0)

        try:
            m6_list.append(m6(version_df))
        except Exception:
            m6_list.append(0)

        ncloc_list.append(_ncloc(version_df))
        repository_list.append(version_df['repository'].iloc[0])
        version_list.append(version)

    final_dict = {
        'm1': m1_list,
        'm2': m2_list,
        'm3': m3_list,
        'm4': m4_list,
        'm5': m5_list,
        'm6': m6_list,
        'repository': repository_list,
        'version': version_list,
        'ncloc': ncloc_list
    }

    metrics_df = pd.DataFrame(final_dict)

    return metrics_df
```

In [35]:

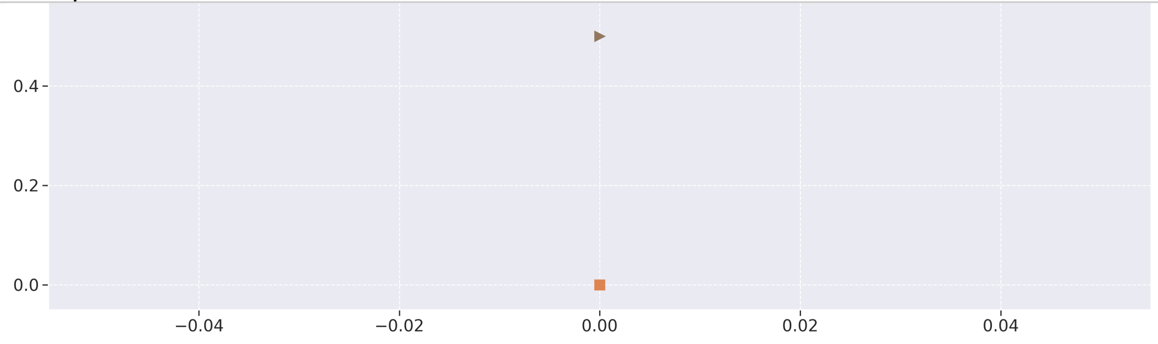
```
# Here we will create a dictionary with the metrics for each repository
metrics = {}

for repo_df in repos_dataframes:
    metrics[repo_df['name']] = create_metrics_df(repo_df['df'])
```

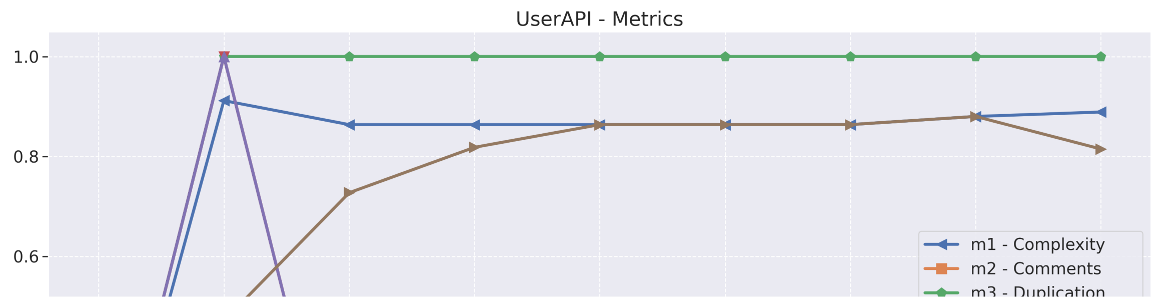
Data visualization

In this area you will need to plot the metrics of each repository.

In [36]:

```
for name, data in metrics.items():
    fig = plt.figure(figsize=(20, 10))

    plt.plot(data['m1'], linewidth=3, marker='<', markersize=10, label="m1 - Comple
    plt.plot(data['m2'], linewidth=3, marker='s', markersize=10, label="m2 - Commen
    plt.plot(data['m3'], linewidth=3, marker='p', markersize=10, label="m3 - Duplic
    plt.plot(data['m4'], linewidth=3, marker='v', markersize=10, label="m4 - Passed
    plt.plot(data['m5'], linewidth=3, marker='^', markersize=10, label="m5 - Fast t
    plt.plot(data['m6'], linewidth=3, marker='>', markersize=10, label="m6 - Covera

    plt.title(f"{name} - Metrics", fontsize=20)
    plt.legend(loc='best')
    plt.show()
```





Quality factor and aspect aggregation

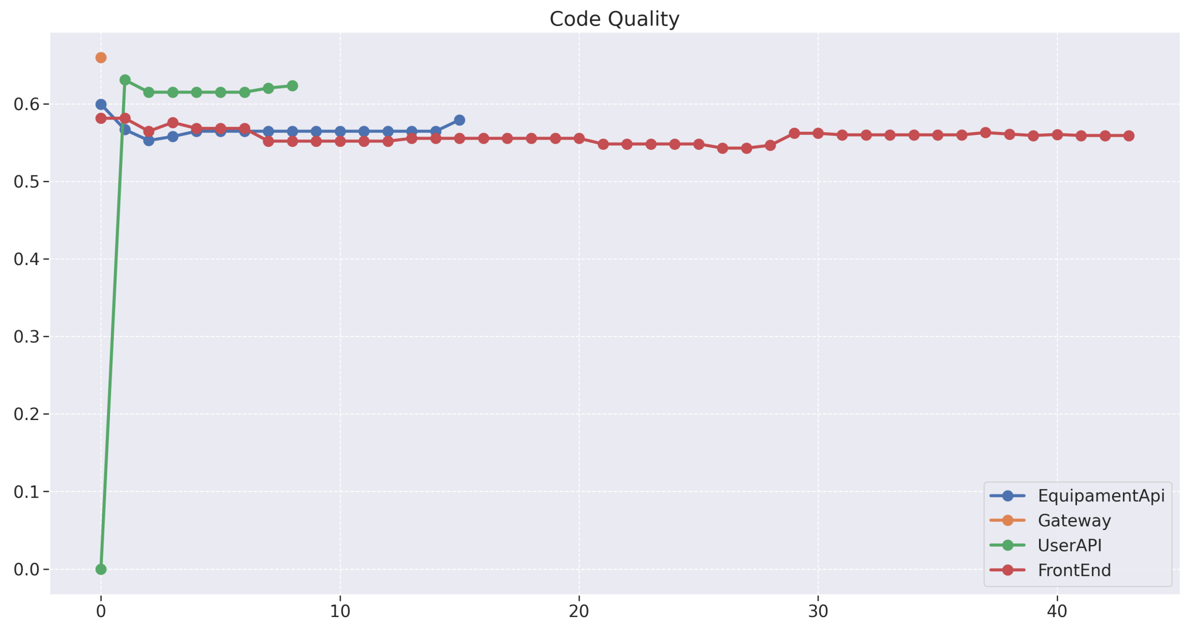
In [37]:

```
psc1 = 1
psc2 = 1
pc1 = 0.5
pc2 = 0.5
pm1 = 0.33
pm2 = 0.33
pm3 = 0.33
pm4 = 0.25
pm5 = 0.25
pm6 = 0.5


# Here you will need to create the code_quality and testing_status metrics for each

for name, data in metrics.items():
    data['code_quality'] = ((data['m1']*pm1) + (data['m2']*pm2) + (data['m3']*pm3))
    data['testing_status'] = ((data['m4']*pm4) + (data['m5']*pm5) + (data['m6']*pm6
```

Code Quality visualization

In [38]:

```
fig = plt.figure(figsize=(20, 10))

for name, data in metrics.items():
    plt.plot(data['code_quality'], linewidth=3, marker='o', markersize=10, label=nar

plt.title("Code Quality", fontsize=20)
plt.legend(loc='best')
plt.show()
```

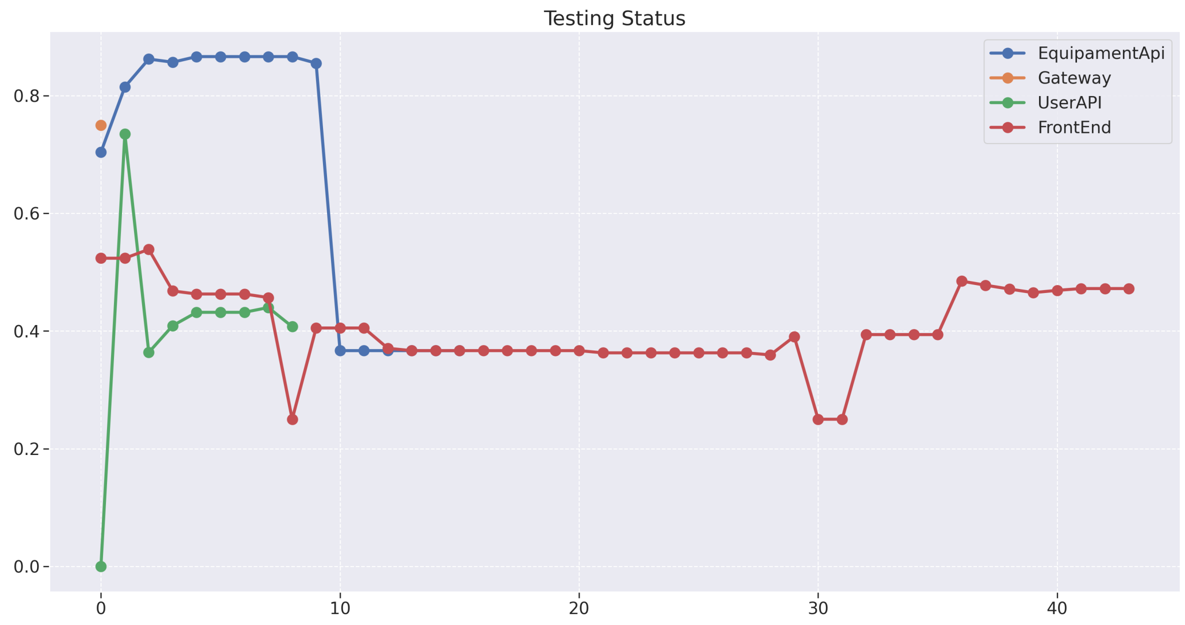


Testing Status visualization

In [39]:

```
fig = plt.figure(figsize=(20, 10))

for name, data in metrics.items():
    plt.plot(data['testing_status'], linewidth=3, marker='o', markersize=10, label=
```

```
plt.title("Testing Status", fontsize=20)
plt.legend(loc='best')
plt.show()
```



Aggregations

In [40]:

```
for name, data in metrics.items():
    data['Maintainability'] = data['code_quality'] * pc1
    data['Reliability'] = data['testing_status'] * pc2
    data['total'] = data['Maintainability'] + data['Reliability']
```

Repositories analysis

In [41]:

```
def get_characteristc_stats(repo_series):
    return {
        'mean': repo_series.mean(),
        'mode': repo_series.mode(),
        'median': repo_series.median(),
        'std': repo_series.std(),
        'var': repo_series.var(),
        'min': repo_series.min(),
        'max': repo_series.max()
    }
```

In [42]:

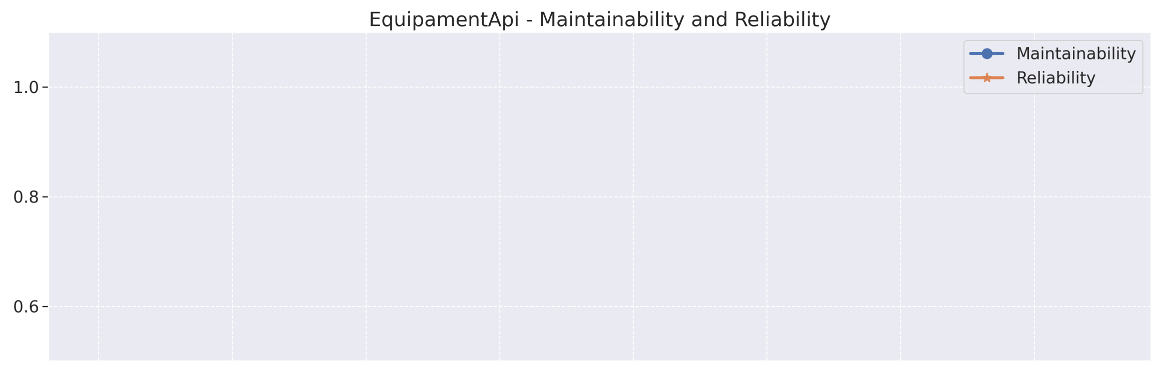
```
def analysis(metrics, name):
    maintainability_stats = pd.DataFrame(get_characteristc_stats(metrics["Maintainal
                                         columns=['mean', 'mode', 'median', 'std', 'var

    reliability_stats = pd.DataFrame(get_characteristc_stats(metrics["Reliability"]
                                     columns=['mean', 'mode', 'median', 'std', 'var', 'ı


    print("Maintainability Stats")
    print(maintainability_stats.to_string(index=False))

    print("Reliability Stats")
    print(reliability_stats.to_string(index=False))

    fig = plt.figure(figsize=(20, 10))

    plt.plot(metrics['Maintainability'], linewidth=3, marker='o', markersize=10, lal
    plt.plot(metrics['Reliability'], linewidth=3, marker='*', markersize=10, label=

    plt.ylim(0.1,1.1)
    plt.title(f'{name} - Maintainability and Reliability', fontsize=20)
    plt.legend(loc='best')
    plt.show()

    fig = plt.figure(figsize=(20, 10))

    plt.plot(metrics['total'], linewidth=3, marker='X', markersize=5)

    plt.ylim(0.1,1.1)
    plt.title(f'{name} - Total', fontsize=20)
    plt.show()
```

Analysis loop in each repo

In [43]:

```
for name, data in metrics.items():
    print(name)
    analysis(data, name)
```

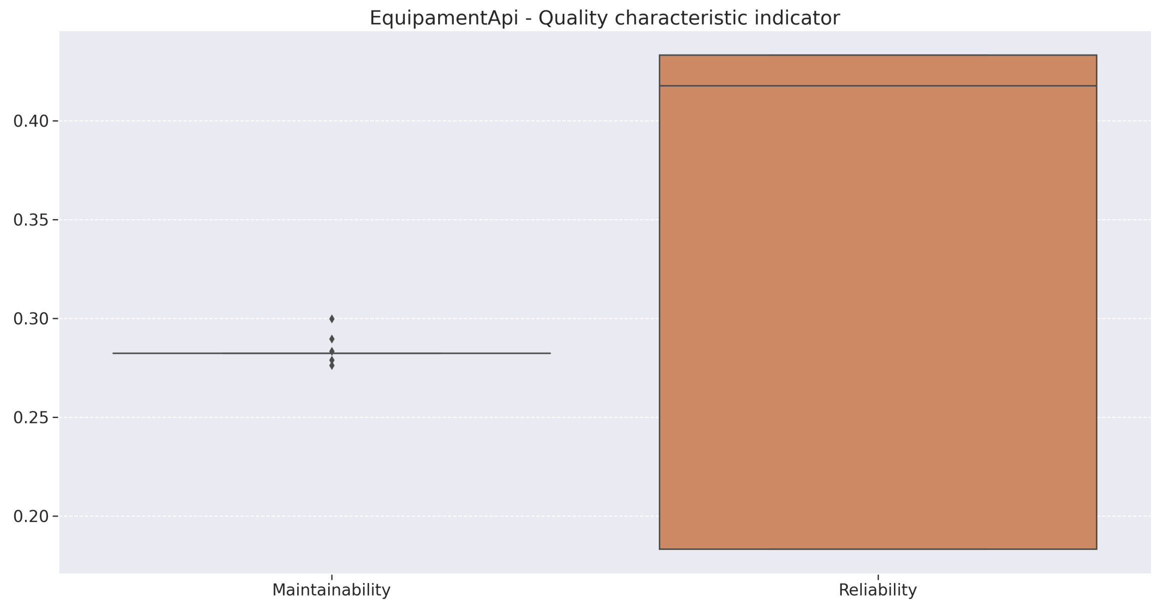
```
EquipamentApi
Maintainability Stats
    mean     mode   median      std      var      min      max
0.283362 0.282333 0.282333 0.005097 0.000026 0.276375 0.299789
Reliability Stats
    mean     mode   median      std      var      min      max
0.332124 0.183333 0.417693 0.120671 0.014562 0.183333 0.433333
```



Quality characteristic indicator

In [44]:

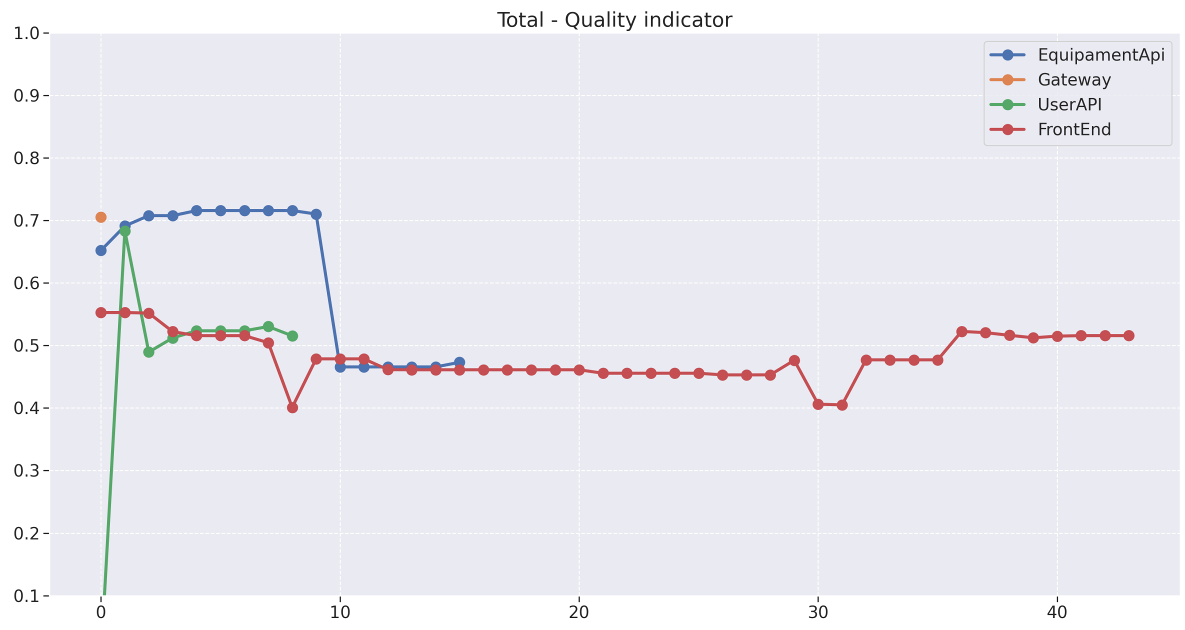
```

for name, data in metrics.items():
    fig = plt.figure(figsize=(20, 10))
    sns.boxplot(data=data[['Maintainability','Reliability']])

    plt.title(f"{name} - Quality characteristic indicator", fontsize=20)
    plt.show()
```



Quality indicator visualization

In [45]:

```
fig = plt.figure(figsize=(20, 10))

for name, data in metrics.items():
    plt.plot(data['total'], linewidth=3, marker='o', markersize=10, label=name)

plt.ylim(.1,1)
plt.title("Total - Quality indicator", fontsize=20)
plt.legend(loc='best')
plt.show()
```



Export data

```
In [46]:
metrics_list = metrics.values()

metrics_df = pd.concat(metrics_list, ignore_index=True)

display(metrics_df)

current_datetime = datetime.datetime.now().strftime("%m-%d-%Y--%H-%M-%S")

metrics_df.to_excel('./data/fga-eps-mds-2023-1-Alectrion--{}.xlsx'.format(current_da

metrics_df.to_csv('./data/fga-eps-mds-2023-1-Alectrion--{}.csv'.format(current_date
```

	m1	m2	m3	m4	m5	m6	repository	version	ncloc	code_quality	testing_sta
0	0.873239	0.000000	0.943662	1.0	1.0	0.408451	fga-eps-mds-2023-1-Alectrion-EquipamentApi	05-13-2023-17-01-24	12409	0.599577	0.704
1	0.804348	0.000000	0.913043	1.0	1.0	0.630435	fga-eps-mds-2023-1-Alectrion-EquipamentApi	05-13-2023-18-10-02	12409	0.566739	0.815
2	0.775000	0.000000	0.900000	1.0	1.0	0.725000	fga-eps-mds-2023-1-Alectrion-EquipamentApi	05-17-2023-22-22-51	12409	0.552750	0.862
3	0.785714	0.000000	0.904762	1.0	1.0	0.714286	fga-eps-mds-2023-1-Alectrion-EquipamentApi	05-25-2023-13-13-16	13213	0.557857	0.857