

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

Oprogramowanie analizujące dane zawarte w pliku csv z zastosowaniem GitHub

Autor:
Mateusz Stanek
Dawid Szoldra

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2025

Spis treści

1. Ogólne określenie wymagań	3
2. Analiza problemu	4
2.1. Wczytanie i manipulacja plikiem	4
2.2. Git	4
2.3. Doxygen	6
3. Projektowanie	7
3.1. Implementacja programu	7
3.2. Git	7
3.3. Doxygen	8
4. Implementacja	9
4.1. Ogólne informacje o implementacji klas	9
4.1.1. MeasurementsImporter.hpp oraz MeasurementsImporter.cpp . . .	9
5. Wnioski	14
Literatura	15
Spis rysunków	16
Spis tabel	17
Spis listingów	18

1. Ogólne określenie wymagań

Celem projektu jest stworzenie programu który będzie analizował dane zawarte w pliku csv- oraz kontrolowanie jego wersji za pomocą narzędzia git. Do zadań programu będzie należało: wczytanie danych z pliku, przechowanie ich w strukturze drzewa, wykonanie operacji takich jak obliczanie sum i średnich w przedziale czasowym, porównywanie wyników w dwóch przedziałach czasowych, wyszukiwanie rekordów spełniających warunki, wypisywanie danych z przedziału czasowego, zapis zmienionych danych do pliku binarnego oraz ich odczyt.

Program będzie podzielony na plik main oraz pliki z funkcjami(każda funkcja będzie w osobnym pliku).

Wynikiem projektu powinno być działające oprogramowanie wczytujące dane z pliku, wykonujące na nim operacje a następnie zapisujące dane.

2. Analiza problemu

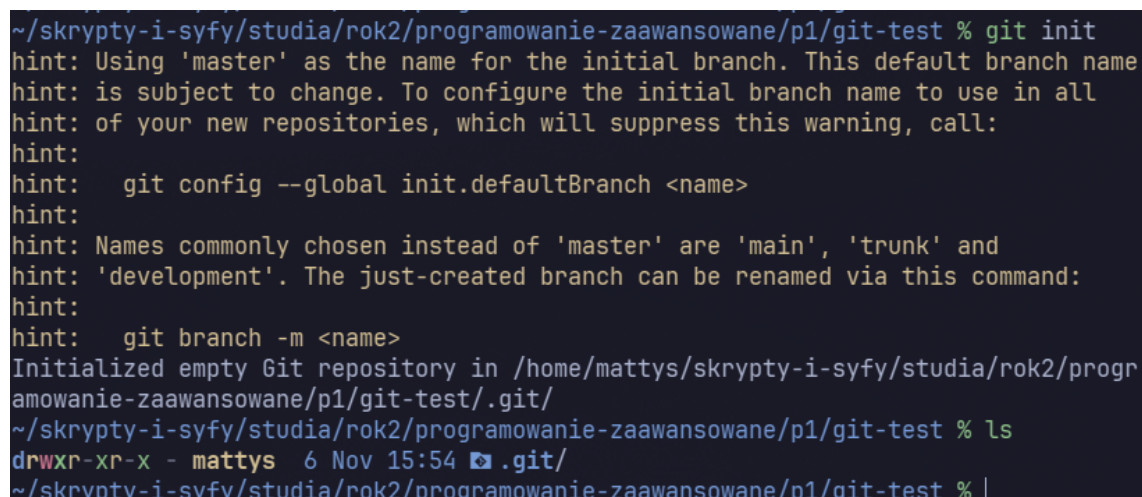
2.1. Wczytanie i manipulacja plikiem

Celem projektu jest stworzenie programu który wczyta, sprawdzi oraz przetworzy dane pomiarowe znajdujące się w pliku CSV. Bardzo ważnym jest uwzględnienie potencjalnych błędów takich jak np. puste linie lub zduplikowane wpisy, takie błędy należy odpowiednio zapisać do logów. Poprawne dane natomiast powinny być zorganizowane w strukturze typu rok -i miesiąc -i dzień -i ćwiartka doby. Taką strukturę można zaimplementować przy pomocy np wektorów. Konieczne jest również stworzenie metody umożliwiającej zapis i odczyt pliku w formacie binarnym oraz implementacja menu która pozwoli na wybór zadań do realizacji.

2.2. Git

Kolejnym konceptem, którym zajmuje się projekt jest narzędzie git[1]. Pozwala ono zarządzać poszczególnymi wersjami projektów. Głównym korzeniem gita jest system commitów, czyli zapisania zmian w pliku w stosunku do commita starszego. To, w połączeniu z jego innymi możliwościami pozwala na tworzenie długich i skomplikowanych osi czasu danych projektów.

Użycie gita można zademonstrować na prostym przykładzie. Tworzymy katalog a w nim repozytorium, używając komendy `git init`, jak widać na rys. 2.1.



```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/mattys/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test/.git/
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % ls
drwxr-xr-x - mattys  6 Nov 15:54 .git/
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % |
```

Rys. 2.1. Puste repozytorium git

Stwórzmy jakiś plik i dodajmy go do repozytorium. Plik można dodać do repozytorium komendą `git add`

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo "commit1" > plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git add plik.txt
```

Rys. 2.2. Stworzenie pliku w repozytorium

Następnie należy scommitować zmiany.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git commit -am "utworzenie pl
ik.txt"
[master (root-commit) bb3b898] utworzenie plik.txt
1 file changed, 1 insertion(+)
create mode 100644 plik.txt
```

Rys. 2.3. Commit nr. 1

Na rysunku 2.3 użyta komenda `git commit` commituje wszystkie dodane pliki (-a) z jakimś komunikatem (-m).

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo "commit2" >> plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git commit -am "uzupelnienie
plik.txt"
[master 40694c2] uzupelnienie plik.txt
1 file changed, 1 insertion(+)
```

Rys. 2.4. Commit nr. 2

Na rys. 2.4, został utworzony kolejny commit, dodający zmiany do `plik.txt`.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git log
commit 40694c2e73758368445cb817f4524ee5c5afbece (HEAD -> master)
Author: mattys1 <mattys0082@gmail.com>
Date:   Wed Nov 6 16:26:07 2024 +0100

    uzupelnienie plik.txt

commit bb3b898df760395b5763575e204c3c43b513d2f6
Author: mattys1 <mattys0082@gmail.com>
Date:   Wed Nov 6 16:12:31 2024 +0100

    utworzenie plik.txt
```

Rys. 2.5. Log gita

Jak na rys. 2.5 jest pokazane, używając komendy `git log`, można wyświetlić log commitów w repozytorium.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git checkout bb3b898df760395b5763575e204c3c43b513d2f6
Note: switching to 'bb3b898df760395b5763575e204c3c43b513d2f6'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at bb3b898 utworzenie plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo plik.txt
plik.txt
```

Rys. 2.6. Demonstracja checkout

Jak widać na rys. 2.6, komenda `git checkout`, pozwala na przejście repozytorium w inny stan, w tym przypadku przechodzi się do commita o danym ID, pokazanym na rys. 2.5. Jako, że jest to pierwszy commit, nie ma w nim zmian z drugiego.

2.3. Doxygen

Doxygen[2] jest narzędziem automatycznie generującym dokumentację programu z komentarzy w kodzie źródłowym. Potrafi on generować strony HTML, gdzie można dynamicznie nawigować się między różnymi częściami kodu oraz pliki \LaTeX , które można konwertować na różne, statyczne formaty.

3. Projektowanie

3.1. Implementacja programu

Do zaimplementowania programu zostanie użyty Język C++ z kompilatorem `gcc` oraz `MVSC`. Wersja standardu C++ to C++23. Wersja ta została użyta, ze względu na zawartą w niej funkcję `std::print()`. Jako, że projekt ma być rozdzielony na dwa pliki, zostanie zastosowany `CMake` w celu automatyzacji procesu budowania. `CMake` pozwala na generowanie plików budujących dany projekt, zgodnie z określoną konfiguracją. Oszczędza to programiście, szczególnie przy większych projektach, manualne pisanie `Makefile`ów. Plik konfiguracyjny `CMakeLists.txt` może wyglądać jak na rysunku

```
1  cmake_minimum_required(VERSION 3.5)
2
3  set(PROJECT_NAME proj1)
4
5  project(${PROJECT_NAME} VERSION 0.1 LANGUAGES CXX)
6
7  set(CMAKE_CXX_STANDARD 23)
8  set(CMAKE_CXX_STANDARD_REQUIRED ON)
9  set(CMAKE_EXPORT_COMPILE_COMMANDS True)
10 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_CURRENT_LIST_DIR}/out)
11
12 add_subdirectory(src)
13
```

Listing 1. Plik konfiguracyjny `CMake`

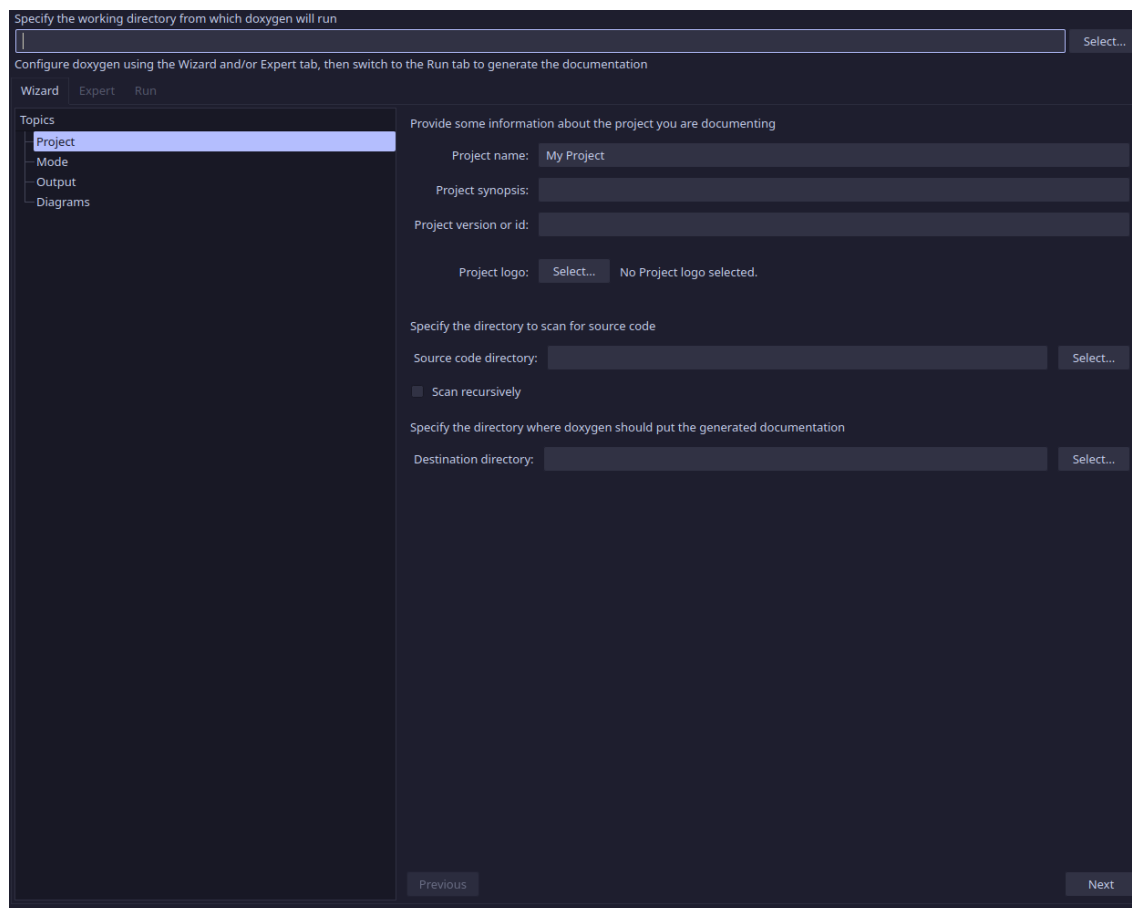
Edytorem będzie program `Neovim` oraz `Visual Studio 2022`. Jest to terminalowy edytor tekstu z możliwością poszerzenia funkcjonalności przy użyciu wszelkiego rodzaju pluginów. Wybrany został, dlatego że jest on już skonfigurowany na moim komputerze zgodnie z moimi preferencjami.

3.2. Git

Dla ułatwienia pracy, zastosowany został front-end dla gita o nazwie `lazygit`. Jest to terminalowy program, którego główną zaletą jest łatwa nawigacja przy użyciu klawiatury. Ponadto, jest on lekki i szybki.

3.3. Doxygen

Konfiguracja dla Doxygena jest wygenerowana przy użyciu programu doxywizard, pokazany na rys. 3.1, pozwalającego na graficzne zmienianie ustawień. Po wygenerowaniu konfiguracji, Doxygen wywoływany jest przy użyciu komendy.



Rys. 3.1. Interfejs programu doxywizard

4. Implementacja

4.1. Ogólne informacje o implementacji klas

Program podzielony jest na wiele plików, w tym rozdziale zostaną opisane funkcjonalności każdego z nich.

4.1.1. MeasurementsImporter.hpp oraz MeasurementsImporter.cpp

Klasa została podzielona na dwa pliki: .hpp oraz .cpp. Na listingu nr 2 przedstawiona została zawartość pliku MeasurementsImporter.hpp.

```
1  #pragma once
2  #include "MeasurementRecord.hpp"
3  #include <string_view>
4  #include <vector>
5  #include <fstream>
6
7  class MeasurementsImporter {
8      private:
9          std::vector<MeasurementRecord> records;
10     public:
11         void read_measurements(const std::string_view fileName);
12         std::vector<MeasurementRecord> get_records() const;
13     };
```

Listing 2. Zawartość pliku MeasurementsImporter.hpp

Na listingu nr 3 przedstawiona została zawartość pliku MeasurementsImporter.cpp.

```
1  #include "MeasurementsImporter.hpp"
2  #include <algorithm>
3  #include <cassert>
4  #include <chrono>
5  #include <ranges>
6  #include <variant>
7
8  void MeasurementsImporter::read_measurements(const std::
9      string_view fileName) {
10     const auto check_for_double { [this](const MeasurementRecord&
11         record) -> bool {
12         return std::ranges::find(records, record) != records.end();
13     }};
14
15     const auto log { [](std::string_view line, std::fstream& file)
16         {
```

```
14     const auto time { std::chrono::system_clock::now() };
15
16     file << std::format("{}: Loading record: {}\n", time, line)
17 ;
18     });
19
20     const auto log_err { [](std::string_view line, const std::
21 string_view message, std::fstream& file) {
22         const auto time { std::chrono::system_clock::now() };
23
24         file << std::format("{}: {}: {}\n", time, message, line);
25     }
26 };
27
28     std::fstream measurementsFile;
29     measurementsFile.open(fileName.data(), std::fstream::in);
30
31     if(!measurementsFile.is_open()) {
32         throw std::runtime_error("Could not open input file");
33     }
34
35     std::fstream errorLogs; errorLogs.open("log_error_data_godzina.
36 txt", std::fstream::app);
37     if(!errorLogs.is_open()) {
38         throw std::runtime_error("Could not open error log file");
39     }
40
41     std::fstream allLogs; allLogs.open("log_data_godzina.txt", std
42 ::fstream::app);
43     if(!allLogs.is_open()) {
44         throw std::runtime_error("Could not open log file");
45     }
46
47     std::vector<std::string> lines;
48     std::string line;
49     while(std::getline(measurementsFile, line)) {
50         if(line.empty() || std::find_if(line.begin(), line.end(), [](
51 const auto ch) { return !isspace(ch); }) == line.end()) {
52             continue;
53         }
54
55         log(line, allLogs);
56
57         if(const auto invalid = std::find_if(line.begin(), line.end()
58 , [](const auto x) {
59             return !((x >= '0' && x <= '9') || x == '.' || x == ',' ||
```

```

x == ':' || x == '\\' || isspace(x));
53     }); invalid != line.end()) {
54         log_err(line, "Line with invalid characters", errorLogs);
55         continue;
56     }
57
58     lines.emplace_back(line);
59 }
60
61 if(lines.size() == 0) {
62     return;
63 }
64
65 records.reserve(lines.size());
66 for(const auto& line : lines) {
67     auto entries { std::views::split(line, ',') | std::ranges::to
<std::vector<std::string>>() };
68
69     if(entries.size() != 6) {
70         log_err(line, "Invalid entry size", errorLogs);
71         continue;
72     }
73
74     const auto DateAndHourTime { entries[0] |
75         std::views::split(',') |
76         std::ranges::to<std::vector<std::string>>() };
77
78     const auto& date { DateAndHourTime[0] }, hourTime {
DateAndHourTime[1] };
79
80     const auto splitDate { date |
81         std::views::split('.') |
82         std::ranges::to<std::vector<std::string>>() };
83
84     const auto& day { splitDate[0] }, month { splitDate[1] },
year { splitDate[2] };
85
86     for(auto& x : entries) {
87         x.erase(remove(x.begin(), x.end(), '\\'), x.end());
88     }
89
90     const auto minutesPerQuarter { 24 * 60 / 4 };
91
92     const auto hoursMinutes { hourTime |
93         std::views::split(':') |

```

```
94     std::ranges::to<std::vector<std::string>>()
95     /* std::ranges::to<std::vector<std::string>>() */; */
96 };
97
98     const auto timeInMinutes { std::stoi(hoursMinutes[0]) * 60 +
std::stoi(hoursMinutes[1]) };
99
100     /* return timeInMinutes / minutesPerQuarter + 1; */
101
102     const auto record { [&]() -> std::variant<MeasurementRecord,
std::exception> const {
103         try {
104             return MeasurementRecord {
105                 .time = {
106                     .year = std::stoi(year),
107                     .month = std::stoi(month),
108                     .day = std::stoi(day),
109                     .inMinutes = timeInMinutes,
110                     .quarter = timeInMinutes / minutesPerQuarter + 1
111                 },
112
113                 .autoconsumption = std::stod(entries[1]),
114                 .gridExport = std::stod(entries[2]),
115                 .gridImport = std::stod(entries[3]),
116                 .consumption = std::stod(entries[4]),
117                 .production = std::stod(entries[5])
118             };
119         } catch(std::exception& e) {
120             return e;
121         }
122     }()};
123
124     if(std::holds_alternative<std::exception>(record)) {
125         log_err(line, "Error converting line to values", errorLogs);
126         continue;
127     }
128
129     if(check_for_double(std::get<MeasurementRecord>(record))) {
130         log_err(line, "Line already exists", errorLogs);
131         continue;
132     }
133
134     records.push_back(std::get<MeasurementRecord>(record));
135 }
```

```
136     }  
137  
138     std::vector<MeasurementRecord> MeasurementsImporter::get_records  
139     () const {  
140         return records;  
    }
```

Listing 3. Zawartość pliku `MeasurementsImporter.cpp`

Głównym zadaniem klasy jest odczytywanie zawartości pliku oraz zapisywanie ich we własnym kontenerze. Za odczyt danych z pliku odpowiedzialna jest metoda `read_measurements` znajdująca się w wierszach od 8 do 136 na listingu nr 2. Instrukcja w wierszach od 8 do 11 odpowiedzialna jest za sprawdzanie czy nie ma duplikatów. Instrukcja w wierszach od 13 do 17 jest odpowiedzialna za logowanie poprawnych danych. Instrukcja w wierszach od 19 do 23 jest odpowiedzialna za logowanie błędów. Instrukcje w wierszach od 25 do 26 są odpowiedzialne za otwarcie pliku wejściowego. Instrukcja `if` w wierszach od 28 do 30 jest odpowiedzialna za sprawdzanie czy plik został otwarty. Instrukcje w wierszach od 32 do 35 są odpowiedzialne za otwarcie pliku z logami błędów. Instrukcje w wierszach od 37 do 40 są odpowiedzialne za otwarcie pliku z logami. Instrukcje w wierszach od 42 do 43 są odpowiedzialne za stworzenie kontenera na wszystkie linie. Instrukcje w wierszach od 44 do 59 są odpowiedzialne za utworzenie pętli wczytującej dane z pliku. Na początku, w wierszach od 45 do 47 są instrukcje odpowiedzialne za pomijanie pustych linii lub linii z samymi białymi znakami. Następnie w wierszu 49 jest instrukcja która loguje każdą linię do pliku z logami zwykłymi. Następne instrukcje w wierszach od 61 do 57 sprawdzają czy linia zawiera dodatkowe znaki, jeżeli wystąpił błąd to logujemy do loga z błędami i przechodzimy dalej. Ostatnia instrukcja pętli w wierszu 58 dodaje linię do wektora `lines`. Instrukcja `in` w wierszach od 61 do 63 sprawdza czy wczytaliśmy linie, jeżeli nie to kończy się działanie funkcji. Instrukcje w wierszu 65 jest odpowiedzialna za rezerwację pamięci dla `records`. Pętla `for` w wierszach od 66 do 135 jest odpowiedzialna za przetwarzanie każdej linii oraz konwersję do `MeasurementRecord`. Pętla zaczyna działanie od rozbicia linii po przecinku na 6 fragmentów w wierszu 67. Następnie w wierszach od 69 do 72 sprawdzane jest czy są 6 fragmentów, jeżeli nie to logujemy do logów z błędami i linia jest pomijana. Instrukcja w wierszach 74 do 78 wyciągają datę. Instrukcje znajdujące się w wierszach od 80 do 82 rozbijają datę po kropkach na dzień, miesiąc i rok. Pętla `for` w wierszach od 86 do 88 jest odpowiedzialna za usuwanie cudzośćłówów. Instrukcja w wierszu 90 ustala liczbę minut przypadających na ćwiartkę doby. Instrukcje w wierszach od 92 do 96 rozdzielają godziny po dwukropku na godziny i minuty. Instrukcja w wierszu 98 jest odpowie-

działa za konwersję godziny i minuty na jedną wartość w minutach od północy. Instrukcje w wierszach od 102 do 122 są odpowiedzialne za tworzenie rekordu w bloku lambda. Instrukcja if w wierszach od 124 do 127 jest odpowiedzialna za czy rekord jest poprawny rekord czy wyjątek. Instrukcja if w wierszach od 129 do 132 jest odpowiedzialna za sprawdzenie czy wczytany rekord już istnieje. Instrukcja w wierszu 134 dodaje nowy rekord do kontenera **records**

5. Wnioski

- Konstrukcja drzewa binarnego przydaje się wtedy, kiedy mamy zamiar kilkakrotnie wyszukiwać dane z nieposortowanego zbioru
- Branchowanie w git bardzo pomaga w pracy kolaboracyjnej
- Rekurencja w pewnych algorytmach jest prostszym rozwiązaniem do zaimplementowania niż metody iteracyjne

Bibliografia

- [1] *Strona Gita*. URL: <https://git-scm.com/>.
- [2] *Strona Doxygena*. URL: <https://www.doxygen.nl/>.

Spis rysunków

2.1. Puste repozytorium git	4
2.2. Stworzenie pliku w repozytorium	5
2.3. Commit nr. 1	5
2.4. Commit nr. 2	5
2.5. Log gita	5
2.6. Demonstracja checkout	6
3.1. Interfejs programu doxywizard	8

Spis tabel

Spis listingów

1.	Plik konfiguracyjny CMake	7
2.	Zawartość pliku <code>MeasurementsImporter.hpp</code>	9
3.	Zawartość pliku <code>MeasurementsImporter.cpp</code>	9