AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Fizyki i Informatyki Stosowanej

Praca inżynierska

Jarosław Cierpich
Arkadiusz Kasprzak

kierunek studiów: **informatyka stosowana**

Rozbudowa i uaktualnienie oprogramowania systemu GGSS detektora ATLAS TRT

Opiekun: **dr hab. inż. Bartosz Mindur**

Kraków, styczeń 2020

Oświadczenie studenta

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. — Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

.....

(czytelny podpis)

Spis treści

1. Wstęp

1.1. Wprowadzenie do systemu GGSS

Detektor ATLAS (*A Toroidal LHC ApparatuS*), znajdujący się w Europejskim Ośrodku Badań Jądrowych *CERN*, jest jednym z detektorów pracujących przy Wielkim Zderzaczu Hadronów (*LHC - Large Hadron Collider*). Pełni on kluczową rolę w rozwoju fizyki cząstek elementarnych, w szczególności badania przy nim prowadzone doprowadziły do potwierdzenia istnienia tzw. *bozonu Higgsa* w roku 2012.

Detektor ATLAS charakteryzuje się budową warstwową - składa się z kilku subdetektorów[1]. Jednym z nich jest Detektor Wewnętrzny (*Inner Detector*) składający się z trzech głównych elementów zbudowanych za pomocą różnych technologii. Elementy te, w kolejności od położonego najbliżej punktu zderzeń cząstek, to: detektor pikselowy (*Pixel Detector*), krzemowy detektor śladów (*SCT - Semiconductor Tracker*) oraz detektor promieniowania przejścia (*TRT - Transition Radiation Tracker*). Dokładny opis zasad działania całego dektektora oraz poszczególnych jego komponentów wykracza poza zakres niniejszego manuskryptu.

W kontekście niniejszej pracy kluczowym jest System Stabilizacji Wzmocnienia Gazowego (*GGSS - Gas Gain Stabilisation System*) dla detektora TRT. Jego oprogramowanie jest zintegrowane z systemem kontroli detektora ATLAS (*DCS - Detector Control System*). W skład systemu GGSS wchodzą zarówno urządzenia takie jak multiplekser i zasilacz wysokiego napięcia, jak i rozbudowana warstwa oprogramowania. Autorzy pracy zaprezentują opis zmian, jakie do tej pory wprowadzili w projekcie GGSS. Zmiany te obejmują m.in. sposób budowania aplikacji wchodzących w skład systemu, ale również automatyzacja prac związanych z jego utrzymaniem i użytkowaniem.

1.2. Cel pracy

Przed autorami postawiony został szereg celów do zrealizowania, związanych zarówno ze zdobyciem wymaganej wiedzy domenowej, jak i przeprowadzeniem modyfikacji oprogramowania systemu GGSS.

Jednym z nich było zapoznanie się z infrastrukturą informatyczną CERN-u. Praca z oprogramowaniem oparta jest tam o unikalny ekosystem, mający zapewnić bezpieczeństwo i stabilność całej infrastruktury, co wiąże się z wieloma ograniczeniami dotyczącymi m.in. dostępu do komputerów produkcyjnych. Konieczne było więc uzyskanie odpowiednich uprawnień i zdobycie doświadczenia w

pracy z tą infrastrukturą. Ze względu na domenę działania systemu GGSS celem było również zdobycie wiedzy na temat sposobu pracy przy dużych eksperymentach, na przykładzie eksperymentu ATLAS. Ponadto uczestnictwo w rozwoju projektu tego typu miało na celu nabycie przez autorów doświadczenia w pracy w międzynarodowym środowisku, jakim jest CERN. Kluczowym dla poprawnego przeprowadzenia prac było również zapoznanie się autorów z zastosowaniem i podstawami sposobu działania systemu GGSS.

Oprócz wyżej wymienionych czynności związanych ze zdobyciem podstawowej wiedzy domenowej, celem niniejszej pracy było przeprowadzenie modyfikacji w warstwie oprogramowania projektu GGSS. Do postawionych przed autorami zadań należało zaplanowanie prac i utworzenie wygodnego, nowoczesnego środowiska do zarządzania projektem informatycznym oraz utworzenie prostego w rozwoju, intuicyjnego systemu budowania oprogramowania opartego o narzędzie CMake. Miało to na celu umożliwienie modularyzacji projektu tak, by każdy z komponentów mógł być niezależnie budowany. Ponadto zadaniem autorów była migracja projektu do systemu kontroli wersji *Git*, stanowiącego ogólnoprzyjęty standard we współczesnych projektach informatycznych. W celu uproszczenia procedury wdrażania projektu w środowisku produkcyjnym celem autorów było również zautomatyzowanie procesu budowania i dystrybucji projektu. Na koniec, by umożliwić innym uczestnikom projektu sprawne korzystanie z nowych rozwiązań, przygotowana miała zostać dokumentacja projektu w formie krótkich instrukcji czy zestawów komend. Dokumentacja, z uwagi na międzynarodowy charakter środowiska w CERN, miała zostać napisana w języku angielskim.

Niniejszy manuskrypt opisuje przede wszystkim prace związane z rozwojem oprogramowania przeprowadzone przez autorów. Praca opisuje stan początkowy projektu, założenia dotyczące stanu docelowego oraz wybrane, zdaniem autorów najważniejsze, zadania zrealizowane w ramach pracy z oprogramowaniem systemu GGSS.

2. Zastosowane technologie

Niniejszy rozdział zawiera krótki opis najważniejszych technologii i narzędzi używanych przez autorów podczas pracy z oprogramowaniem systemu GGSS. Przedstawione tu opisy zawierają podstawową wiedzę o sposobie działania i użytkowania tych technologii - szczegółowe przykłady przedstawione zostały w dalszej części pracy, w kontekście konkretnych rozwiązań zrealizowanych przez autorów w projekcie.

2.1. Język C++

C++ jest kompilowanym językiem programowania ogólnego przeznaczenia **BJARNE** opartym o statyczne typowanie. Został stworzony jako obiektowe rozszerzenie języka C (z którym jest w dużej mierze wstecznie kompatybilny), lecz wraz z rozwojem pojawiło się w nim wsparcie dla innych paradygmatów, w tym generycznego i funkcyjnego. Sprawiło to, że język ten stał się bardzo wszechstronny - pozwala zarówno na szybkie wykonywanie operacji niskopoziomowych **BJARNE strona 41**, jak i na tworzenie wysokopoziomowych abstrakcji**BJARNE, PRACA GOSCIA**. Dodatkową cechą wyróżniającą C++ wśród innych języków umożliwiających programowanie obiektowe jest jego wysoka wydajność.

Standardy języka
W ciągu ostatnich kilku lat C++ przechodzi proces intensywnego rozwoju - od 2011 roku pojawiły się trzy nowe standardy tego języka, a kolejny przewidziany jest na rok 2020. Wspomniane nowe standardy to:

- C++11 - wprowadza funkcjonalności takie jak: wsparcie dla wielowątkowości, wyrażenia lambda, referencje do *r-wartości*, biblioteka do obsługi wyrażeń regularnych, dedukcja typów za pomocą słowa kluczowego *auto* czy pętla zakresowa. Standard ten uważany jest za przełom w rozwoju języka.
- C++14 - rozszerza zmiany wprowadzone w C++11. Nie zawiera tak wielu przełomowych zmian jak poprzedni standard - twórcy skupili się na poprawie istniejących błędów oraz rozwoju istniejących rozwiązań **wiki: https://en.wikipedia.org/wiki/C%2B%2B14**, np. dedukcji typu zwracanego z funkcji za pomocą słowa kluczowego *auto*.

– C++17 - wprowadza m.in. nowe typy danych (np. *std::variant*, *std::byte* i *std::optional*), algorytmy współbieżne, biblioteka *filesystem* przeznaczona do obsługi systemu plików oraz rozszerzenie mechanizmu dedukcji typów w szablonach na szablony klas. **STRONA https://infotraining.pl/szkolenie/c-plus-plus/cpp17**. Standard ten usuwa również pewne elementy uznane za przestarzałe, np. inteligentny wskaźnik *std::auto_ptr*, zastąpiony w standardzie C++11 przez inne rozwiązanie.

Zmiany wprowadzane w nowych standardach pozwalają na tworzenie czytelniejszego kodu, który łatwiej utrzymywać i rozwijać. Ma to znaczenie zarówno na poziomie pojedynczych instrukcji czy typów danych, jak i na poziomie architektury projektu. Listingi 2.1 oraz 2.2 przedstawiają przykład zmiany, jaka zaszła między starym standardem C++03, a C++11. Zaprezentowany kod realizuje w obu przypadkach iterację po zawartości kontenera typu *std::vector<int>* mającą na celu wypisanie na standardowe wyjście jego zawartości. Przykład ten, pomimo że bardzo prosty, dobrze obrazuje wzrost jakości i czytelności kodu w nowym standardzie.

Listing 2.1. Przykład kodu w języku C++ napisany z wykorzystaniem standardu C++03

```
1  // kontener zawierający 6 elementów typu int - inicjalizacja
2  // za pomocą tymczasowej tablicy, możliwa w statym standardzie
3  // języka
4  int tmp_arr[] = {1, 2, 3, 4, 5, 6};
5  std::vector<int> a (tmp_arr, tmp_arr + 6);
6
7  // iteracja po zawartości kontenera w standardzie C++03
8  for (std::vector<int>::const_iterator it = a.begin(); it != a.end(); ++it) {
9      std::cout << *it << " ";
10 }
```

Listing 2.2. Przykład kodu w języku C++ napisany z wykorzystaniem funkcjonalności ze standardu C++11 (zakresowa pętla for)

```
1  // kontener zawierający 6 elementów typu int - nowy
2  // sposób inicjalizacji
3  std::vector<int> a{1, 2, 3, 4, 5, 6};
4
5  // iteracja po zawartości kontenera w standardzie C++11 -
6  // przykład zastosowania zakresowej pętli for
7  for (const auto& elem: a) {
8          std::cout << elem << " ";
9  }
```

Boost

Boost jest zestawem bibliotek dla języka C++, poszerzających w znacznym stopniu wachlarz narzędzi programistycznych dostarczanych przez język. Biblioteki wchodzące w skład Boost dostarczają funk-

cjonalności takich, jak: wygodne przetwarzanie tekstu, zapewnienie interfejsu między C++ a językiem Python czy programowanie sieciowe. Boost to projekt aktywnie rozwijany, bardzo popularny. Niektóre z bibliotek wchodzących w jego skład zostały przeniesione (nie zawsze w postaci identycznej względem oryginału) do standardu C++. **Doxy BOOST**

2.2. Biblioteki statyczne i dynamiczne

2.3. Narzędzie CMake

TODO: dodac referencje do zrodel

CMake (Cross-platform Make) to narzędzie pozwalające na konfigurację procesu budowania oprogramowania (aplikacji oraz bibliotek) w sposób niezależny od platformy. Jego działanie opiera się na generowaniu pliku budującego natywnego dla określonej platformy (dla systemów z rodziny UNIX jest nim *Makefile*) na podstawie przygotowanego przez użytkownika pliku *CMakeLists.txt*. Takie podejście w znacznym stopniu ułatwia tworzenie aplikacji multiplatformowych oraz pozwala na intuicyjne zarządzanie zależnościami w projekcie. Domyślnie CMake pracuje z językami C i C++, natomiast nowe wersje narzędzia wpierają ponadto m.in. język C# czy technologię CUDA. Narzędzie to jest rozwijane i wspierane przez firmę *Kitware*.

Plik CMakeLists.txt

Jak wspomniano wyżej działanie narzędzia CMake opiera się na przygotowanym przez użytkownika pliku (lub zestawie plików rozmieszczonych w strukturze katalogów projektu) *CMakeLists.txt*. Plik ten zawiera polecenia napisane w specjalnie do tego celu przygotowanym języku skryptowym. Użytkownik może za jego pomocą m.in. określać jakie pliki wykonywalne mają zostać wygenerowane podczas procesu budowania, wskazać lokalizację plików źródłowych czy określić zależności między komponentami projektu oraz bibliotekami zewnętrznymi.

Prosty przykład

Listing 2.3 zawiera przykład prostego pliku *CMakeLists.txt*, pozwalającego na zbudowanie napisanej w języku C++ obiektowej wersji klasycznego programu *Hello world*. Przykład ilustruje zastosowanie podstawowych poleceń CMake do określenia minimalnej wersji narzędzia, standardu języka C++, wynikowego pliku wykonywalnego oraz potrzebnych plików nagłówkowych.

Listing 2.3. Przykład prostego pliku CMakeLists.txt przeznaczonego do budowania programu napisanego w C++

```
1 # Określenie minimalnej wersji CMake
2 cmake_minimum_required(VERSION 3.0 FATAL_ERROR)
3
4 # Określenie standardu języka C++
5 set(CMAKE_CXX_STANDARD 14)
```

```
6  set(CMAKE_CXX_STANDARD_REQUIRED True)
7
8  # Nazwa oraz wersja projektu
9  project(Hello VERSION 1.0)
10
11 # Dodanie pliku wykonywalnego, który powinien powstać
12 # wskutek procesu budowania
13 add_executable(Hello Main.cpp)
14
15 # Dodanie do projektu katalogu include wraz ze znajdującym się
16 # wewnątrz niego plikiem nagłówkowym
17 target_include_directories(Hello PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include")
```

Wersje CMake

CMake jest narzedziem, który w ciągu ostatnich kilku lat przechodzi gruntowne zmiany. Starsze wersje (np. 2.8) oparte są o prosty system zmiennych**STRONA Z MODERN**, co wprowadza szereg trudności w zarządzaniu dużymi projektami z wielopoziomowymi drzewami zależności. Dodatkowym problemem tych wersji jest również brak dobrze zdefiniowanych tzw. *dobrych praktyk* oraz nieprzystępna dla początkujących dokumentacja. Współczesne wersje narzędzie CMake (zwykle za takie uznaje się nowsze od wersji 3.0) opierają się na innym, bardziej ustrukturyzowanym**STRONA Z MODERN** podejściu, co było przyczyną pojawienia się dla nich wyżej wspomnianych *dobrych praktyk*. Zalecane jest więc, by nowe projekty prowadzone były właśnie z użyciem nowszych wersji narzędzia.

Narzędzia CTest i CPack

CMake oferuje również możliwość konfiguracji sposobu testowania projektu. Służy do tego narzędzie *CTest*, dystrybuowane razem z podstawowym narzędziem CMake. Innym przydatnym modułem jest *CPack* - narzędzie to służy przygotowywaniu pakietów instalacyjnych z oprogramowaniem. Użycie obu wymienionych narzędzi polega na umieszczeniu w pliku *CMakeLists.txt* kilku przeznaczonych do tego komend.

2.4. Język Python

Python jest nowoczesnym, wysokopoziomowym językiem programowania, wspierającym takie paradygmaty jak programowanie obiektowe czy imperatywne. Działanie Pythona opiera się na dynamicznym systemie typów. Z założenia Python jest językiem przyjemnym w użytkowaniu, co przyczyniło się do jego dużej popularności. Python jest szeroko stosowany jako język skryptowy - takie też zastosowanie znalazł w projekcie GGSS. **WIKI albo cos**

Prosty przykład

Listing 2.4 przedstawia prosty przykład skryptu napisanego w języku Python w wersji 3. Kod ten

stanowi uproszczoną wersję skryptu zaprezentowanego w dalszej części pracy, którego zadaniem jest zbudowanie aplikacji w zależności od przekazanych przez użytkownika argumentów. Przykład prezentuje prosty skrypt przyjmujący jeden z dwóch możliwych argumentów i wypisujący informację na temat otrzymanego argumentu na standardowe wyjście.

Listing 2.4. Przykład prostego skryptu napisanego w języku Python 3 - przetwarzanie argumentów podanych przez użytkownika do skryptu

```
1  import argparse
2
3  ## Prosty skrypt przetwarzający argumenty podane
4  ## przy jego uruchomieniu przez użytkownika
5
6  ## Definicja funkcji w języku Python
7  def parse_command_line_arguments():
8
9      ## Obiekt przetwarzający argumenty (parser)
10     parser = argparse.ArgumentParser()
11
12     ## Argumenty wzajemnie się wykluczające
13     group = parser.add_mutually_exclusive_group(required=True)
14     group.add_argument("-s", "--staticboost",
15         help="Use static Boost linking.", action="store_true")
16     group.add_argument("-d", "--dynamicboost",
17         help="Use dynamic Boost linking.", action="store_true")
18
19     return parser.parse_args()
20
21 if __name__=="__main__":
22     ## Wywołanie funkcji
23     arguments = parse_command_line_arguments()
24     print(arguments)
```

Wersje języka Python

Python funkcjonuje w dwóch wersjach: Python 2 oraz 3. Wersje te nie są ze sobą w pełni kompatybilne, tzn. pewne funkcjonalności Pythona 2 nie są dostępne w Pythonie 3 i odwrotnie. Różnice znaleźć można również np. w domyślnym sposobie kodowania łańcuchów znakowych (ASCII w Pythonie 2, Unicode 2 Pythonie 3) oraz w wyniku dzielenia (za pomocą operatora /) dwóch liczb całkowitych (w Pythonie 2 wynikiem jest liczba całkowita, w Pythonie 3 liczba zmiennoprzecinkowa typu *float*)**REFERECJA DO https://learntocodewith.me/programming/python/python-2-vs-python-3/**. Ponadto zakończenie oficjalnego wsparcia Pythona w wersji 2 przewidziane jest na styczeń 2020 roku**WSTAWIC LINKA DO NJUSA** - co w momencie pisania niniejszej pracy (grudzień 2019) jest terminem niedalekim i miało kluczowe znaczenie w czasie podejmowania pewnych decyzji projektowych.

Zewnętrzne biblioteki

Jedną z największych zalet Pythona jest bardzo duża liczba bibliotek zewnętrznych tworzonych przez społeczność Pythona. Rozbudowują one język o wiele nowych funkcjonalności, np. przetwarzanie plików HTML czy wykonywanie obliczeń numerycznych. W niniejszej pracy zastosowanych zostało kilka tego typu bibliotek, m.in. *Beautiful Soup* do wspomnianego wyżej przetwarzania dokumentów w formacie HTML. Omówienie ich działania na przykładach znaleźć można w dalszej części pracy - przy opisie konkretnego ich zastosowania.

2.5. Język powłoki bash

2.6. System kontroli wersji Git i portal Gitlab

2.7. Manager pakietów - RPM

2.8. Technologie wirtualizacji i konteneryzacji

3. Stan początkowy projektu

3.1. Architektura

3.2. Budowanie

3.3. Dostarczanie i uruchamianie

3.4. Kontrola wersji

4. Stan docelowy projektu

5. Ograniczenia dostępnej infrastruktury

Z uwagi na silny związek oprogramowania GGSS z infrastrukturą CERN oraz wymóg zapewnienia możliwości budowania projektu na należących do niej maszynach, przed autorami postawiony został szereg ograniczeń związanych z możliwymi do użycia technologiami oraz sposobem wykonywania pewnych operacji. Niniejszy rozdział stanowi opis najważniejszych z tych ograniczeń z uwzględnieniem ich wpływu na obraną przez autorów pracy ścieżkę rozwoju projektu.

5.1. Ograniczone uprawnienia w środowisku docelowym

5.2. Wersje kompilatorów i interpreterów

Dostępne wersje kompilatorów i interpreterów stanowią jeden z kluczowych czynników, który należy uwzględnić podczas wprowadzania zmian w istniejącym systemie, ponieważ definiują one możliwy do wykorzystania podzbiór technologii. W kontekście systemu GGSS ograniczenia te dotyczą przede wszystkim kompilatora języka C++ oraz interpretera języka Python.

Wersja kompilatora języka C++

Dostępna w ramach infrastruktury projektu wersja kompilatora języka C++ to **g++ (GCC) 4.8.5**. Wspiera ona w pełni standard C++11, czyli funkcjonalności takie, jak referencje do r-wartości, wyrażenia lambda czy zakresowa pętla for. **STRONA Z NOWOSCIAMI W WERSJACH** Wersja ta nie wspiera niestety nowszych wydań języka (C++14/17).

Wersja interpretera języka Python

Domyślną wersją Pythona jest **Python 2.7.5**, jednak dostępny jest również Python 3 (w wersji **Python 3.6.8**). Z uwagi na wspomniany wcześniej koniec oficjalnego wsparcia dla Pythona 2, który ma nadejść wraz z początkiem 2020 roku, naturalnym jest więc wybór wersji 3. Infrastruktura projektu posiada jednak znaczące braki jeśli chodzi o dostępne dla wersji 3 biblioteki zewnętrzne - domyślnie nie jest np. dostępna biblioteka *Beautiful Soup*, służąca do przetwarzania dokumentów w formacie HTML. Niektóre popularne bibliteki i frameworki (np. *PyTest* - wykorzystywany do przeprowadzania testów oprogramowania) nie są dostępne dla obu wersji Pythona. Taka sytuacja wymusza więc wyko-

rzystanie narzędzia *virtualenv* w celu ich instalacji w odizolowanym środowisku, nie mającym wpływu na infrastrukturę CERN-u.

5.3. Wersja narzędzia budującego CMake

Dostępna wersja narzędzia CMake stanowiła zdaniem autorów największe ograniczenie w czasie prac nad projektem. Na maszynach docelowych dostępna jest jedynie stara wersja **2.8.12.2**. Nowsza wersja (**3.14.6**) dostępna jest na niektórych z komputerów, jednak z uwagi na konieczność zachowania kompatybilności ze wspomnianymi maszynami docelowymi, nie było możliwe jej użycie. Stosowanie wersji o numerze niższym niż **3.0** skutkuje szeregiem ograniczeń - brakuje w niej wielu funkcjonalności pozwalających na stosowanie ogólnoprzyjętych dziś praktyk, jak np. określenie zakresu wersji narzędzia CMake, w którym powinna mieścić się używana wersja, by projekt można było bez problemu zbudować, czy wsparcie dla instrukcji *target_link_directories*. **https://cliutils.gitlab.io/modern-cmake/chapters/intro/newcmake.html**

5.4. Związek projektu z wersją jądra systemu

6. Wykonane prace

6.1. Wykorzystanie funkcjonalności portalu Gitlab wspierających zarządzanie projektem

6.2. Migracja projektu do systemu kontroli wersji Git i zmiany w architekturze

6.3. Zastosowanie podejścia CI/CD

6.4. Zmiana sposobu budowania aplikacji

6.5. Budowanie i dystrybucja sterownika oraz aplikacji testującej

6.6. Maszyna wirtualna oraz konteneryzacja - Docker

6.7. Pomniejsze prace

6.7.1. Integracja bibliotek napisanych w języku C z aplikacją w C++

6.7.2. Integracja zewnętrznej biblioteki dynamicznej z użyciem narzędzia CMake

6.8. Dokumentacja projektu

7. Dalsza ścieżka rozwoju projektu

7.1. Wprowadzenie zautomatyzowanego systemu testowania projektu

7.2. Migracja do nowego standardu języka C++

7.3. Automatyzacja procesu publikowania produktu

8. Podsumowanie oraz wnioski

A. Dodatki/Appendixes

A.1. Adding new modules to the project using existing CMake templates

A.2. Preparing virtual machine to work as a runner

Bibliografia

[1] Ulrich Drepper. „Budowa aparatury detekcyjnej i przygotowanie programu fizycznego przyszłych eksperymentów fizyki cząstek (ATLAS i LHCb na akceleratorze LHC i Super LHC oraz eksperymentu na akceleratorze liniowym ILC).” W: (list. 2007), s. 13.