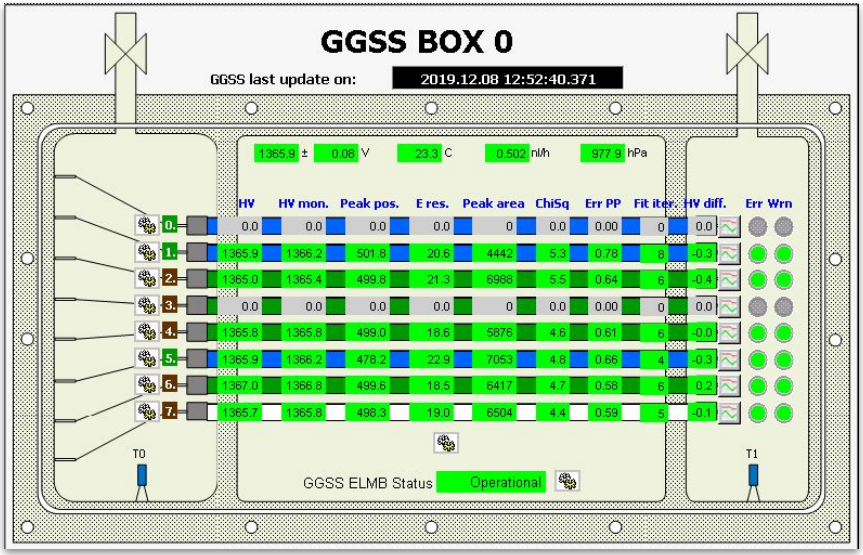
Rozbudowa i uaktualnienie systemu GGSS detektora ATLAS TRT

Opiekun: dr hab. inż. Bartosz Mindur, prof. AGH
Arkadiusz Kasprzak, Jarosław Cierpich

Wprowadzenie do systemu GGSS

- projekt działający przy detektorze ATLAS TRT w CERN
- GGSS składa się z warstwy oprogramowania i sprzętu
- w uproszczeniu: warstwa oprogramowania koordynuje działanie urządzeń (m.in. zasilacze, multipleksery analogowe) oraz umożliwia sterowanie nimi
- cykliczny pomiar i aktualizacja



Panel WinCC OA przedstawiający liczniki słomkowe
(źródło: materiały własne)

Cele pracy magisterskiej

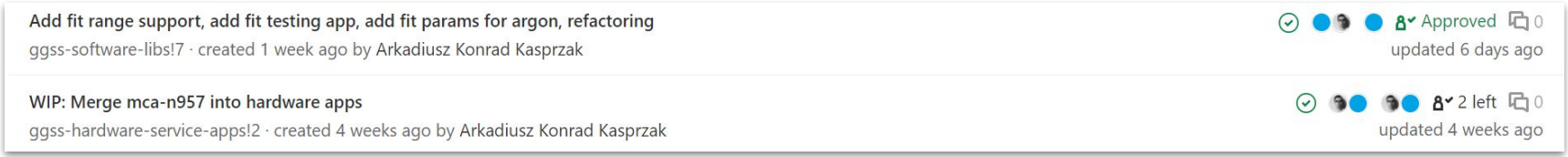
- kontynuacja pracy inżynierskiej dotyczącej tego samego zagadnienia
- zwiększenie jakości kodu źródłowego (m. in. migracja do standardu C++11)
- rozszerzenie możliwości aplikacji (m. in. dodanie dodatkowych komend)
- rozbudowanie systemów budowania i zarządzania zależnościami aplikacji
- przygotowanie narzędzi ułatwiających testy warstwy sprzętowej
- przygotowanie przyjaznego systemu wdrożenia w środowisko produkcyjne w oparciu o pakiety RPM
- przeprowadzenie testów działania nowej wersji systemu
- migracja systemu na nową infrastrukturę sprzętową

Praktyki stosowane w projekcie

W celu organizacji i usprawnienia pracy stosowane są:

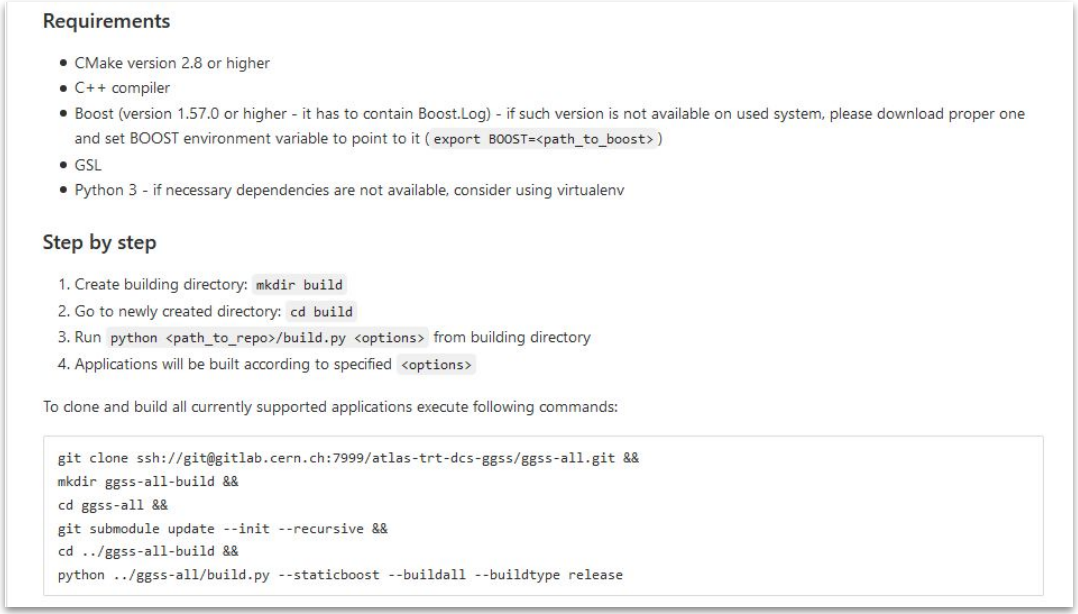
- issues - jako organizacja pojedynczego zadania
- code review oraz merge request
- kanban board - w celu organizacji pracy nad całym projektem
- coding convention
- dokumentacja dla każdego projektu w README oraz kodzie

Zrezygnowano natomiast z wykorzystywania kamieni milowych.



Przykładowe *merge request'y*
(źródło: materiały własne)

Praktyki stosowane w projekcie



Requirements

- CMake version 2.8 or higher
- C++ compiler
- Boost (version 1.57.0 or higher - it has to contain Boost.Log) - if such version is not available on used system, please download proper one and set BOOST environment variable to point to it (`export BOOST=<path_to_boost>`)
- GSL
- Python 3 - if necessary dependencies are not available, consider using virtualenv

Step by step

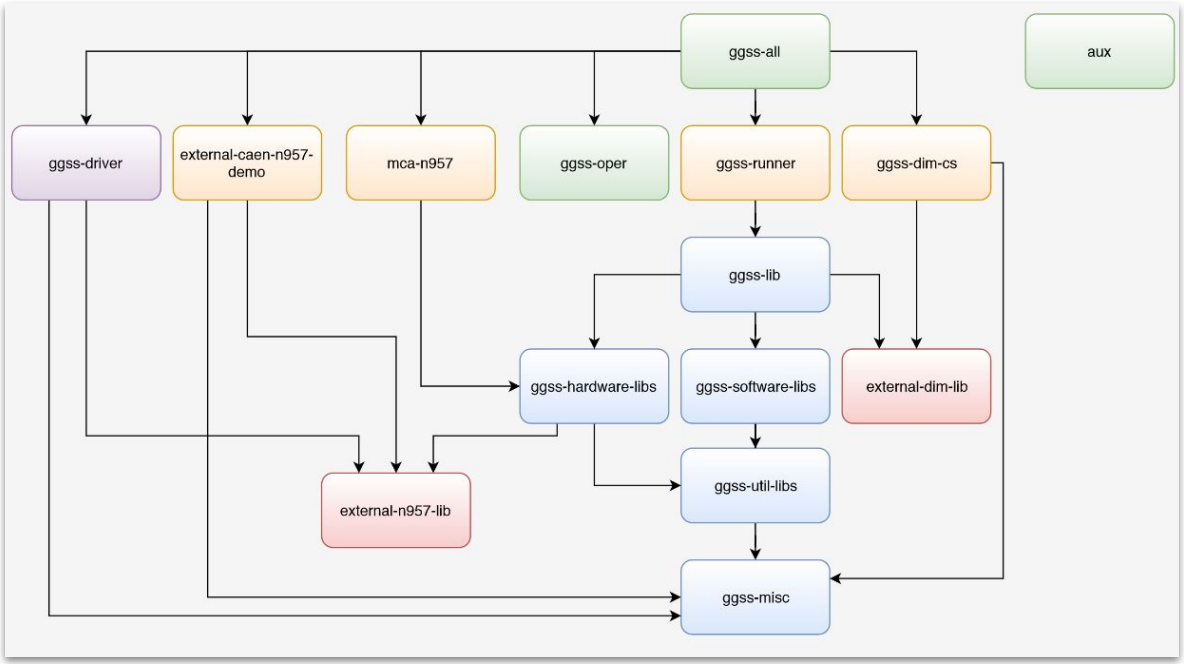
1. Create building directory: `mkdir build`
2. Go to newly created directory: `cd build`
3. Run `python <path_to_repo>/build.py <options>` from building directory
4. Applications will be built according to specified `<options>`

To clone and build all currently supported applications execute following commands:

```
git clone ssh://git@gitlab.cern.ch:7999/atlas-trt-dcs-ggss/ggss-all.git &&
mkdir ggss-all-build &&
cd ggss-all &&
git submodule update --init --recursive &&
cd ../ggss-all-build &&
python ../ggss-all/build.py --staticboost --buildall --buildtype release
```

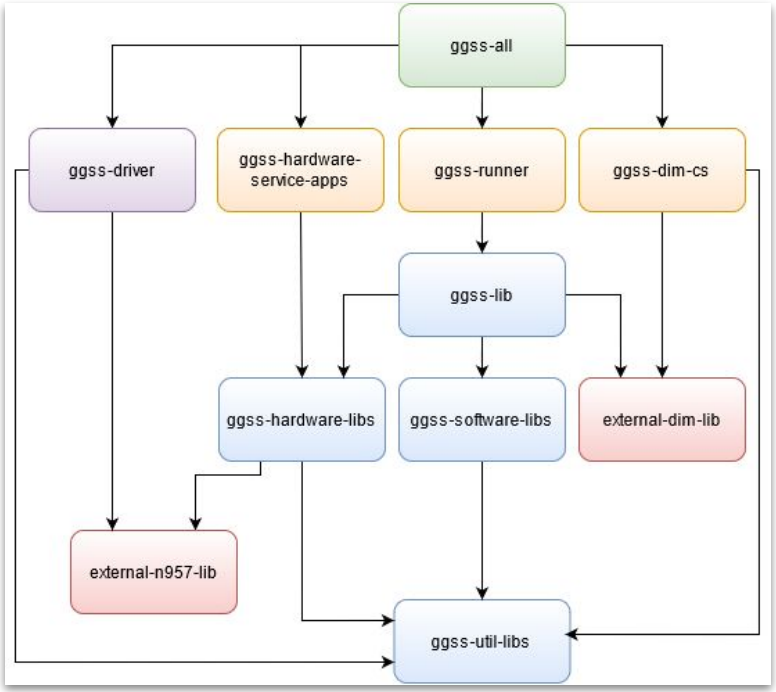
Przykładowe *README*
(źródło: materiały własne)

Aktualizacja struktury projektu



Oryginalna struktura projektu (podział na repozytoria i ich zależności)
(źródło: materiały własne)

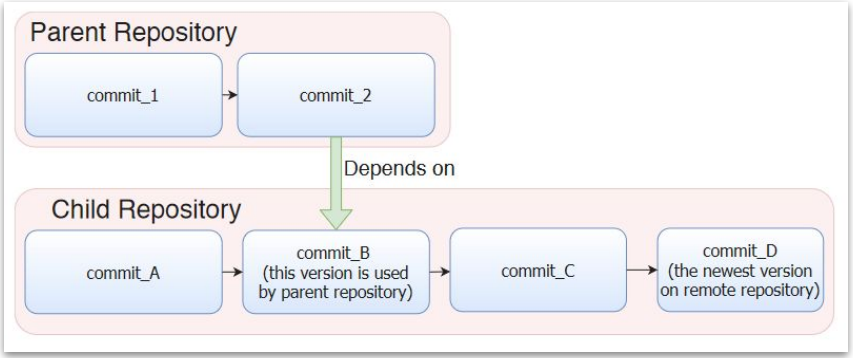
Aktualizacja struktury projektu



Nowa struktura projektu
(źródło: materiały własne)

Automatyzacja pracy z submodułami

- w celu zarządzania zależnościami wewnątrz projektu wykorzystano *git submodules*
- ze względu na braki w API gita przygotowano skrypty automatyzujące (klonowanie submodułów, wybranie odpowiedniego brancha, aktualizowanie połączeń, publikowanie zmian)
- status prac: ukończone

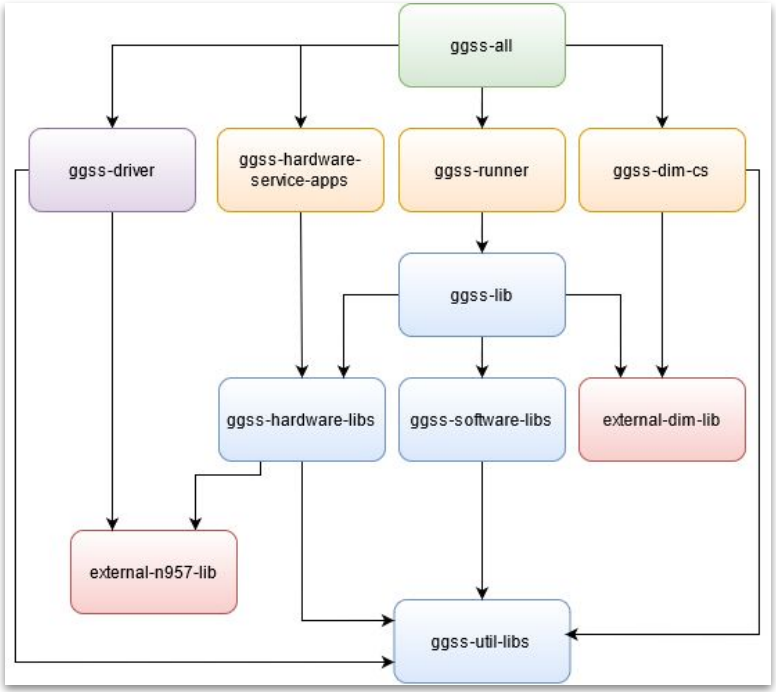


Sposób działania sub-modułów
(źródło: materiały własne)

```
root@host:/# python gitio.py -p ./ggss-all/
...(17 lines truncated)
INFO - Aligning ./ggss-all/mca-n957 repository
INFO - Aligning ./ggss-all/ggss-dim-cs repository
INFO - Aligning ./ggss-all/ggss-runner repository
INFO - Aligning ./ggss-all/ggss-spector repository
INFO - Aligning ./ggss-all/ggss-oper repository
INFO - Aligning ./ggss-all/ggss-driver repository
INFO - Aligning ./ggss-all repository
INFO - Aligning finished.
```

Działanie skryptu GGSS gitio
(źródło: materiały własne)

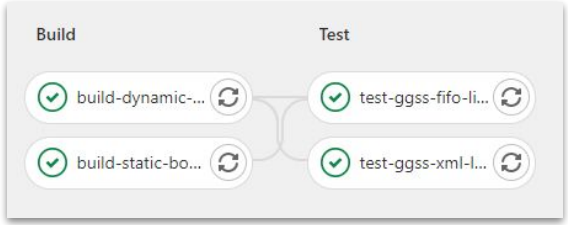
Automatyzacja pracy z submodułami



Nowa struktura projektu
(źródło: materiały własne)

Przygotowanie systemu ciągłej integracji i dostarczania (CI/CD)

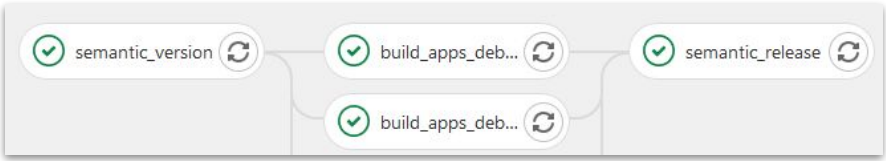
- system umożliwiający automatyczne budowanie i testowanie modułów projektu
- oparty o GitLab CI (pliki *.yaml* umieszczone w repozytorium)
- tworzenie tzw. *pipeline*-ów odzwierciedlających, jakie działania mają zostać automatycznie podjęte po opublikowaniu przez programistę swoich zmian
- możliwość automatycznego przygotowywania tzw. *artefaktów*, zawierających gotowe wydania aplikacji
- status prac: ukończone



Przykładowy pipeline
(źródło: materiały własne)

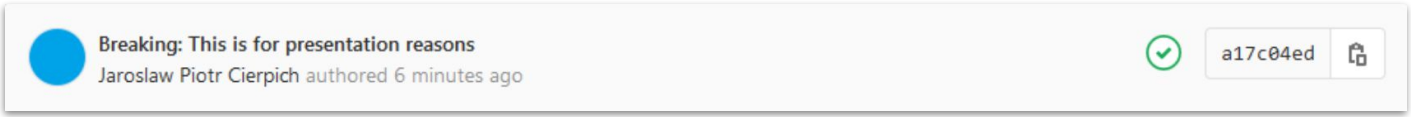
Automatyzacja i centralizacja wersjonowania projektu

- w trakcie wykonywania prac powstało wymaganie scentralizowanego wersjonowania
- w celu ujednolicenia wersjonowania w projekcie przystosowano *pipeline CI/CD* do pracy z *“eslint semantic-versioning”*
- dostosowano również skrypty budujące oraz pliki CMAKE
- status prac: ukończone



Pipeline realizujący automatyczne wersjonowanie
(źródło: materiały własne)

Automatyzacja i centralizacja wersjonowania projektu



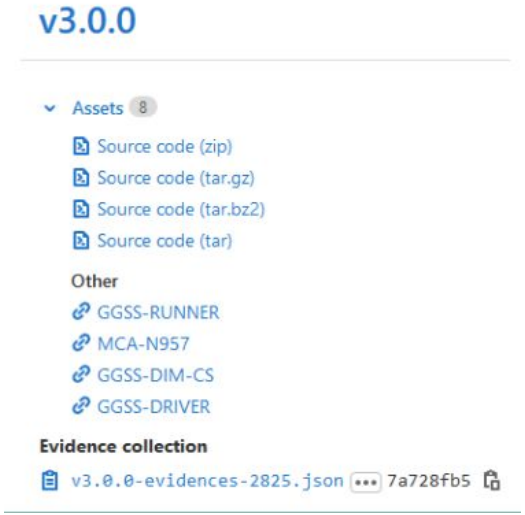
Przykładowy commit w formacie eslint
(źródło: materiały własne)

```
[2:49:15 PM] [semantic-release] [@semantic-release/commit-analyzer] › ℹ  Analyzing commit: Breaking: This is for presentation reasons
[2:49:15 PM] [semantic-release] [@semantic-release/commit-analyzer] › ℹ  The release type for the commit is major
[2:49:15 PM] [semantic-release] [@semantic-release/commit-analyzer] › ℹ  Analysis of 29 commits complete: major release
[2:49:15 PM] [semantic-release] › ✔  Completed step "analyzeCommits" of plugin "@semantic-release/commit-analyzer"
[2:49:15 PM] [semantic-release] › ℹ  The next release version is 1.0.0
```

Działanie semantic-release
(źródło: materiały własne)

Pakietowanie oraz wdrażanie w środowisko docelowe

- w celu szybkiego wdrażania w środowisko produkcyjne, w ramach *pipeline CI/CD* tworzone są pakiety RPM
- w ramach projektu ggss wydzielono 3 pakiety RPM zawierające:
  - główną aplikację wraz z konfiguracją
  - aplikacje do testowania sprzętu
  - sterownik do sprzętu oraz jego konfigurację
- dostosowano również odpowiednio wymagania wyżej wymienionych pakietów
- wdrażanie w środowisko docelowe możliwe jest również “starym” sposobem, tj.: poprzez ręczne umieszczenie aplikacji - wyjście awaryjne
- możliwość pobierania gotowych do użycia aplikacji
- status prac: w trakcie



Nowe wydanie zawierające gotowe do użycia aplikacje (artefakty)
(źródło: materiały własne)

GGSS - warstwa oprogramowania

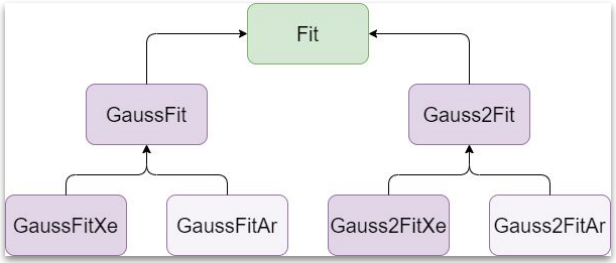
- znaczna część prac skupiona wokół modyfikacji kodu źródłowego projektu (w przeciwieństwie do pracy inżynierskiej)
- jedna główna aplikacja (*ggss-runner*) oraz kilka pomniejszych
- technologie: C++ (standard 11), pakiet Boost, biblioteka GSL
- ponadto infrastruktura oparta o: CMake (budowanie) oraz skrypty napisane w językach Python i Bash
- ograniczenia narzucone przez środowisko docelowe (dedykowana maszyna produkcyjna) - brak możliwości wykorzystania najnowszych wersji oprogramowania
- niezawodność jako kluczowy aspekt pracy projektu

Zwiększenie jakości kodu źródłowego

- migracja do standardu C++11 (m.in. pętla zakresowa, inteligentne wskaźniki, jednolita inicjalizacja)
- zmiany w architekturze wybranych bibliotek
- usunięcie nieużywanych części kodu (pojedyncze wyrażenia, funkcje, klasy)
- usunięcie nieużywanych zależności (w tym likwidacja zależności cyklicznych)
- poprawa niewykrytych dotąd błędów
- zwiększenie bezpieczeństwa poprzez eliminację zjawisk takich jak udostępnianie stanu wewnętrznego klas
- ujednolicenie oraz uzupełnienie dokumentacji
- status prac: ukończone, większość zmian poddana testom operacyjnym

Przykład: zmiana architektury biblioteki *fit-lib*

- biblioteka odpowiedzialna za aproksymację funkcji
- oryginalna architektura oparta o rozbudowaną hierarchię klas (dziedziczenie)
- klasy *GaussFitXe* i *GaussFitAr* oraz *Gauss2FitXe* i *Gauss2FitAr* różnią się jedynie parametrami początkowymi aproksymacji
- ponadto możliwe było stworzenie instancji klas *GaussFit* oraz *Gauss2Fit* - nie są one jednak poprawne z punktu widzenia domeny (brak parametrów początkowych) - możliwe było zatem wprowadzenie programu w niepoprawny stan
- rozwiązanie: zastąpienie rozbudowanego dziedziczenia wzorcem projektowym *Strategia* - wstrzykiwanie parametrów początkowych, wyeliminowanie dolnego poziomu hierarchii



oryginalna architektura biblioteki
(źródło: materiały własne)

Rozszerzenie możliwości aplikacji

- zwiększenie interaktywności: możliwość aktualizacji danych na żądanie
- rozbudowa biblioteki przeprowadzającej aproksymację funkcji (m. in. dodatkowe możliwości konfiguracji, zmiana architektury)
- poszerzenie możliwości komunikacji z zasilaczami wysokiego napięcia (przygotowanie specjalnych komend, pozwalających operować na kilku urządzeniach i kanałach jednocześnie)
- pomniejsze funkcjonalności, np. możliwość resetu konfiguracji odpowiedzialnej za kolejność pomiaru na poszczególnych detektorach słomkowych do konfiguracji domyślnej
- status prac: ukończone, większość zmian poddana testom operacyjnym

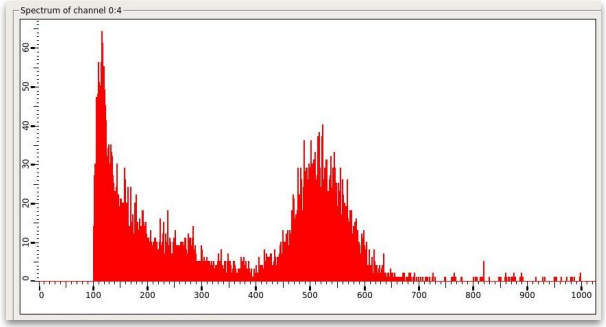
Przykład: biblioteka przetwarzająca komendy do zasilaczy

- GGSS wymaga do poprawnego działania zastosowania 3 zasilaczy wysokiego napięcia (CAEN N1470)
- do tej pory użytkownik mógł prowadzić z nimi interakcję za pomocą systemu komend wspieranego przez producenta
- przykład: $BD:00CMD:SET,CH:0,PAR:VSET,VAL:1368.0
- format taki jest niewygodny i nieintuicyjny z punktu widzenia użytkownika
- ponadto nie pozwala on na odczyt danych lub wprowadzanie zmian na kilku zasilaczach jednocześnie
- przygotowano więc nowy, przyjazny użytkownikowi format komend:
- przykład: hv *:0 set vset 1368.0
- przykład: hv *:* mon vset,vmon
- implementacja jako biblioteka statyczna - możliwość wielokrotnego użytku

zasilacze CAEN N1470
(źródło: materiały własne)

Przykład: możliwość aktualizacji danych na żądanie

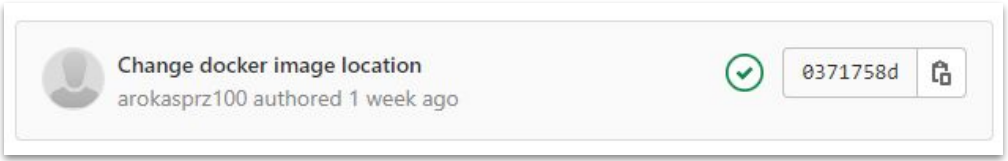
- system cyklicznie zbiera dane (widmo) za pomocą liczników słomkowych
- pojedynczy pomiar (jeden licznik) trwa około 5 minut (zależnie od konfiguracji)
- do tej pory nie istniała możliwość zobaczenia wyników pomiaru w trakcie jego trwania
- w ramach pracy dodano szereg komend zwiększających interakcję użytkownika z systemem, w tym komendę update spectrum
- pozwala ona zobaczyć dane, które zostały już zebrane, przed zakończeniem pomiaru

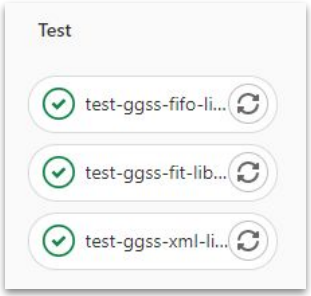


przykład zbieranych przez system danych
(źródło: materiały własne)

Testy automatyczne

- jedną z najważniejszych cech systemu GGSS jest jego niezawodność
- każda z wprowadzonych zmian musi więc zostać dokładnie przetestowana
- do tworzenia testów jednostkowych wykorzystana została biblioteka *Boost.Test* (rozważany był również m.in. *GoogleTest* oraz *FakeIt*)
- każdy moduł projektu może być testowany niezależnie
- prace prowadzone zgodnie z praktyką *Test Driven Development (TDD)*
- testy pozwoliły wykryć kilka pomniejszych błędów w działaniu projektu
- testy są uruchamiane automatycznie, gdy programista umieszcza zmiany w repozytorium projektu (mechanizm GitLab CI/CD)





Rozbudowa systemu budowania aplikacji

- znaczną jego część stanowi narzędzie CMake (wieloplatformowość - generowanie plików *Makefile* lub projektów *Visual Studio*)
- większość zmian w systemie wykonana w ramach pracy inżynierskiej
- pomniejsze zmiany w ramach pracy magisterskiej: wsparcie dla testów automatycznych, generowania dokumentacji, poprawa błędów, zwiększenie czytelności kodu (korzystanie z systemu przypomina teraz korzystanie z języka programowania)
- przygotowano szczegółową dokumentację
- rozbudowano skrypty pomocnicze (pozwalające m.in. na wybór konfiguracji *debug/release*)
- status prac: ukończone

Narzędzia do testowania warstwy sprzętowej

- urządzenia współpracujące z systemem GGSS muszą być sprawne przed jego uruchomieniem
- powstaje zatem konieczność przygotowania dodatkowych skryptów i aplikacji, pozwalających na łatwe sprawdzenie działania urządzeń
- początkowo w projekcie przygotowane było nieliczne oprogramowanie na to pozwalające
- przygotowano ustandaryzowane narzędzia, pozwalające w prosty sposób prowadzić interakcję z urządzeniami
- wspierane urządzenia: zasilacze wysokiego napięcia, multipleksery
- narzędzie opierają się o dwa możliwe tryby pracy:
    - manualny - użytkownik za pomocą wiersza poleceń wpisuje komendy opisujące akcje, jakie powinny zostać wykonane na urządzeniu
    - oparty o scenariusze - użytkownik przygotowuje plik opisujący przebieg testu, następnie na jego podstawie test wykonywany jest automatycznie
- status prac: ukończone

Narzędzia do testowania warstwy sprzętowej

- do parsowania scenariuszy testowych przygotowana została osobna biblioteka
- scenariusze mają format podobny do popularnego formatu YAML

```
# This file contains some example scenarios that can be used with the multiplexer app.

SetAndCheckActiveChannelsScenario:
  - getsn         # Get serial number
  - setgetch 0    # Switch to channel 0 and check
  - setgetch 1    # Switch to channel 1 and check
  - setgetch 2    # Switch to channel 2 and check
  - setgetch 3    # Switch to channel 3 and check
  - setgetch 4    # Switch to channel 4 and check
  - setgetch 5    # Switch to channel 5 and check
  - setgetch 6    # Switch to channel 6 and check
  - setgetch 7    # Switch to channel 7 and check
  - setgetch 8    # Switch to channel 8 and check


ReadSerialNumberAndActiveChannelScenario:
  - getsn         # Get serial number
  - getch         # Get current channel
```

przykładowy plik ze scenariuszami testowymi dla multipleksera
(źródło: materiały własne)

Testy działania nowej wersji systemu

- ze względu na wymóg wysokiej niezawodności GGSS, każda większa zmiana w systemie wymaga dodatkowych testów manualnych w środowisku docelowym
- tego typu testy wykonywane były regularnie, przez cały okres prac nad systemem
- w ramach testów manualnych analizowane są:
  - logi systemu GGSS
  - zużycie pamięci przez główną aplikację GGSS
  - obserwowalny sposób zachowania systemu
  - porównanie pomiarów z urządzeń dokonywanych przez GGSS ze wskazaniami samych urządzeń
- ponadto planowane jest przeprowadzenie w najbliższym czasie bardziej szczegółowych testów, sprawdzających m.in. jak system reagował będzie na nietypowe sytuacje

Migracja systemu na nową infrastrukturę sprzętową

- jednym z celów pracy magisterskiej jest migracja systemu na nową infrastrukturę sprzętową
- wykonanie tej części potencjalnie wymaga wyjazdu do CERN
- nowa infrastruktura ma obejmować komputer na którym uruchamiane są aplikacje systemu GGSS oraz hub USB
- działanie hub'u USB zostało wstępnie sprawdzone w ramach testów manualnych
- aktualny status: wyjazd w lipcu niemożliwy (z uwagi na ograniczenia wynikające z COVID), prawdopodobnie prace wykonane zostaną zdalnie, z pomocą osób dostępnych w CERN

Podsumowanie

- większość prac została wykonana
- pozostało przeprowadzić finalne, dogłębne testy systemu oraz ewentualnie wprowadzić po nich poprawki
- aktualnie przygotowywany jest manuskrypt pracy magisterskiej

Dziękujemy za uwagę.