

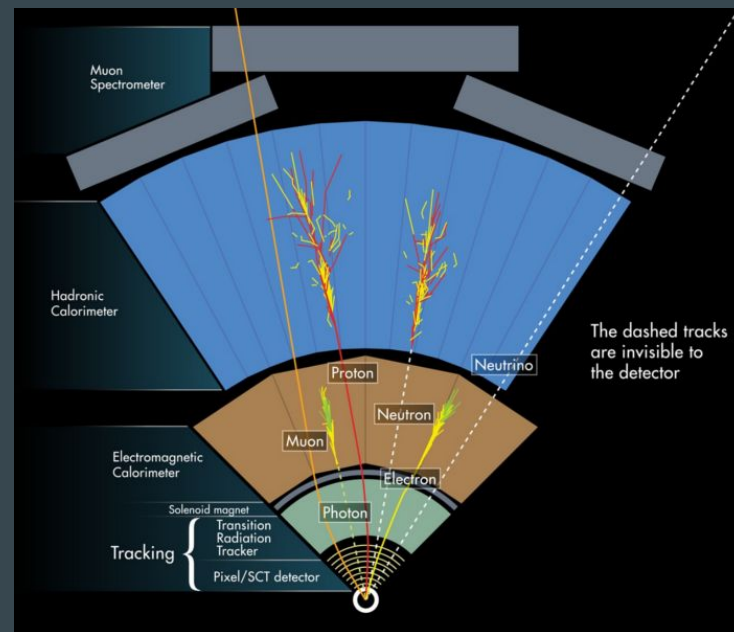
# Rozbudowa i uaktualnienie systemu GGSS detektora ATLAS TRT



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

# Wprowadzenie do systemu GGSS

- System Stabilizacji Wzmocnienia Gazowego (*GGSS - Gas Gain Stabilization System*)
- projekt zintegrowany z systemem kontroli detektora ATLAS w CERN
- umożliwia poprawne działanie detektora promieniowania przejścia (*TRT - Transition Radiation Tracker*) będącego częścią ATLAS

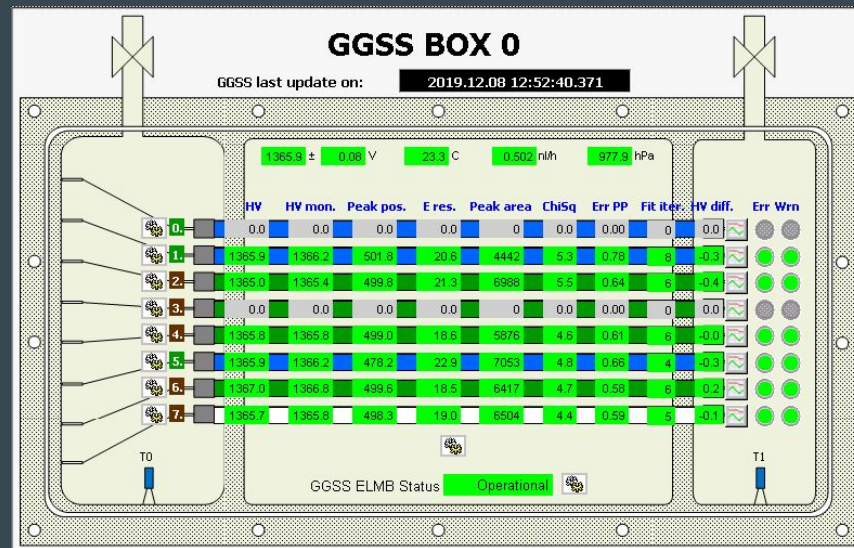


Warstwy detektora ATLAS

(źródło: <http://collider.physics.ox.ac.uk/atlas.html>)

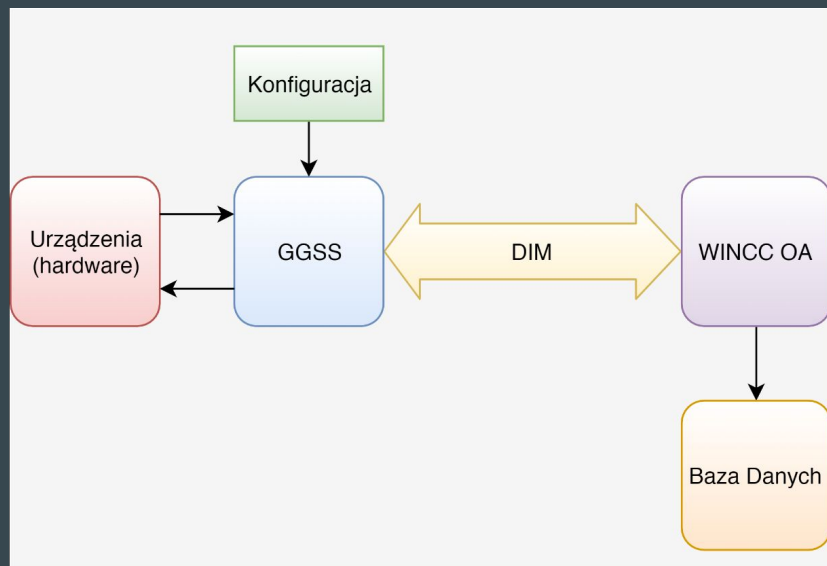
# Wprowadzenie do systemu GGSS

- GGSS składa się z warstwy oprogramowania i sprzętu
- w uproszczeniu: warstwa oprogramowania koordynuje działanie urządzeń (m.in. zasilacze, multipleksery analogowe) oraz umożliwia sterowanie nimi
- liczniki słomkowe
- cykliczny pomiar i aktualizacja



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

# GGSS - architektura



Architektura systemu GGSS  
(źródło: materiały własne)

- WinCC OA - system pozwalający na sterowanie i podgląd danych w interfejsie graficznym, współpracujący m. in. z systemem GGSS
- DIM - protokół komunikacji dla aplikacji rozproszonych
- konfiguracja - plik XML zawierający parametry startowe aplikacji GGSS

# GGSS - warstwa oprogramowania

- projekt napisany w języku C++ z wykorzystaniem pakietu Boost
- jedna główna aplikacja (*ggss-runner*) oraz kilka pomniejszych
- system budowania aplikacji oparty o narzędzie CMake
- infrastruktura (skrypty, pomniejsze aplikacje) napisane w języku Python oraz w formie skryptów powłoki Bash
- system musi funkcjonować w środowisku o zastrzeżonej kontroli (wewnętrzna sieć CERN, dedykowana maszyna produkcyjna)
- na aplikacji nałożone są pewne obostrzenia, które ograniczają spektrum możliwości dotyczących wykorzystywanego oprogramowania oraz jego wersji (CMake, g++, etc.)

# Cele pracy magisterskiej

- kontynuacja pracy inżynierskiej dotyczącej tego samego zagadnienia
- zwiększenie jakości kodu źródłowego (m. in. migracja do standardu C++11/14)
- 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 (w zależności od sytuacji - COVID)
- założenie: zachowanie wstecznej kompatybilności z poprzednimi wersjami WinCC OA i plików konfiguracyjnych

# Zwiększenie jakości kodu źródłowego

- migracja do standardu C++11/14 (m.in. pętla zakresowa, inteligentne wskaźniki, jednolita inicjalizacja)
- 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

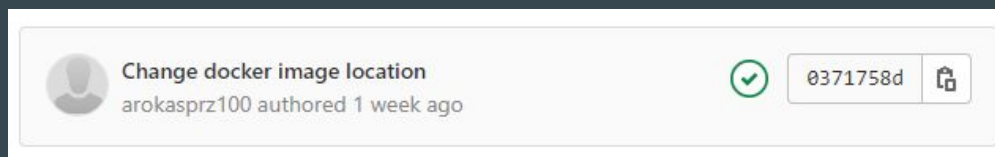
# Rozszerzenie możliwości aplikacji

- zwiększenie interaktywności: możliwość aktualizacji danych na żądanie
- rozbudowa biblioteki przeprowadzającej aproksymację funkcji
- 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



# Testy automatyczne

- jedną z najważniejszych cech systemu GGSS jest jego niezawodność (system do tej pory działał bezawaryjnie przez wiele lat)
- każda z wprowadzonych zmian musi więc zostać dokładnie przetestowana, by nie wprowadzić do systemu błędów
- do tworzenia testów jednostkowych wykorzystana została biblioteka *Boost.Test*
- każdy moduł projektu może być testowany niezależnie
- testy pozwoliły wykryć kilka pomniejszych błędów w działaniu projektu
- testy są uruchamiane automatycznie, gdy programista umieszcza zmiany w repozytorium projektu



# Podejście Test Driven Development

- nowe funkcjonalności tworzone są z wykorzystaniem podejścia Test Driven Development (TDD)
- tworzenie testów przed tworzeniem funkcjonalności
- podejście trójfazowe
- przyspiesza tworzenie oprogramowania i zwiększa jego niezawodność

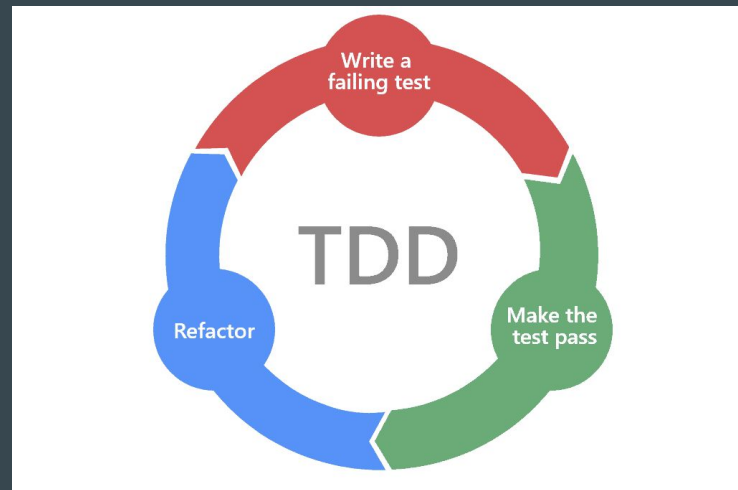












Diagram ilustrujący zasadę działania TDD  
(źródło: <https://marsner.com/blog/why-test-driven-development-tdd/>)

# Rozbudowa systemu budowania aplikacji

- znaczną jego część stanowi narzędzie CMake (wieloplatformowość)
- na podstawie zawartości plików *CMakeLists.txt* generowane są pliki *Makefile* (Linux) lub projekty *Visual Studio* (Windows)
- początkowo bardzo prosty: możliwość budowania jedynie całego projektu
- praca inżynierska: dodanie możliwości budowania pojedynczych modułów i aplikacji
- praca magisterska: wsparcie dla testów automatycznych, generowania dokumentacji, poprawa błędów, zwiększenie czytelności kodu

# Rozbudowa systemu budowania aplikacji

- system zbudowany został tak, by korzystanie z niego (z punktu widzenia programisty) przypominało pracę z popularnymi językami programowania (modularność, podział na funkcje)
- przygotowana została szczegółowa dokumentacja opisująca, jak korzystać z każdego modułu

Name	Last commit	Last update
..		
 BuildDependencies.cmake	Refactor CMake templates to add support fo...	1 week ago
 BuildStaticLibrary.cmake	Remove support for the header-only librarie...	1 week ago
 BuildTypeManagement.cmake	Structure refactoring - rename cmake templ...	4 weeks ago
 CheckPlatform.cmake	Structure refactoring - rename cmake templ...	4 weeks ago
 FindGSL.cmake	Structure refactoring - rename cmake templ...	4 weeks ago
 FindLibraryBoost.cmake	Refactor CMake templates to add support fo...	1 week ago
 FindLibraryGSL.cmake	Refactor CMake templates to add support fo...	1 week ago
 README.md	Remove support for the header-only librarie...	1 week ago
 SetupDoxxygen.cmake	Structure refactoring - rename cmake templ...	4 weeks ago
 SetupTests.cmake	Refactor CMake templates to add support fo...	1 week ago

Zrzut ekranu z repozytorium projektu, przedstawiający szablony CMake  
(źródło: materiały własne)

# Rozbudowa systemu budowania aplikacji

- przygotowano ponadto skrypt w języku Python pozwalający na łatwe budowanie poszczególnych części projektu w określonych konfiguracjach (debug/release, linkowanie statyczne/dynamiczne itp.)
- przykład użycia:

```
python ../build.py --staticboost --buildtype debug --apps runner dimcs mcan957
```

- takie podejście pozwala użytkownikowi w łatwy i szybki sposób zbudować wybrane aplikacje w odpowiedniej dla niego konfiguracji

# Rozbudowa i uaktualnienie systemu GGSS detektora ATLAS TRT



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

# 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



Przykładowy pipeline  
(źródło: materiały własne)

# Rozbudowa systemu wersjonowania i zarządzania zależnościami

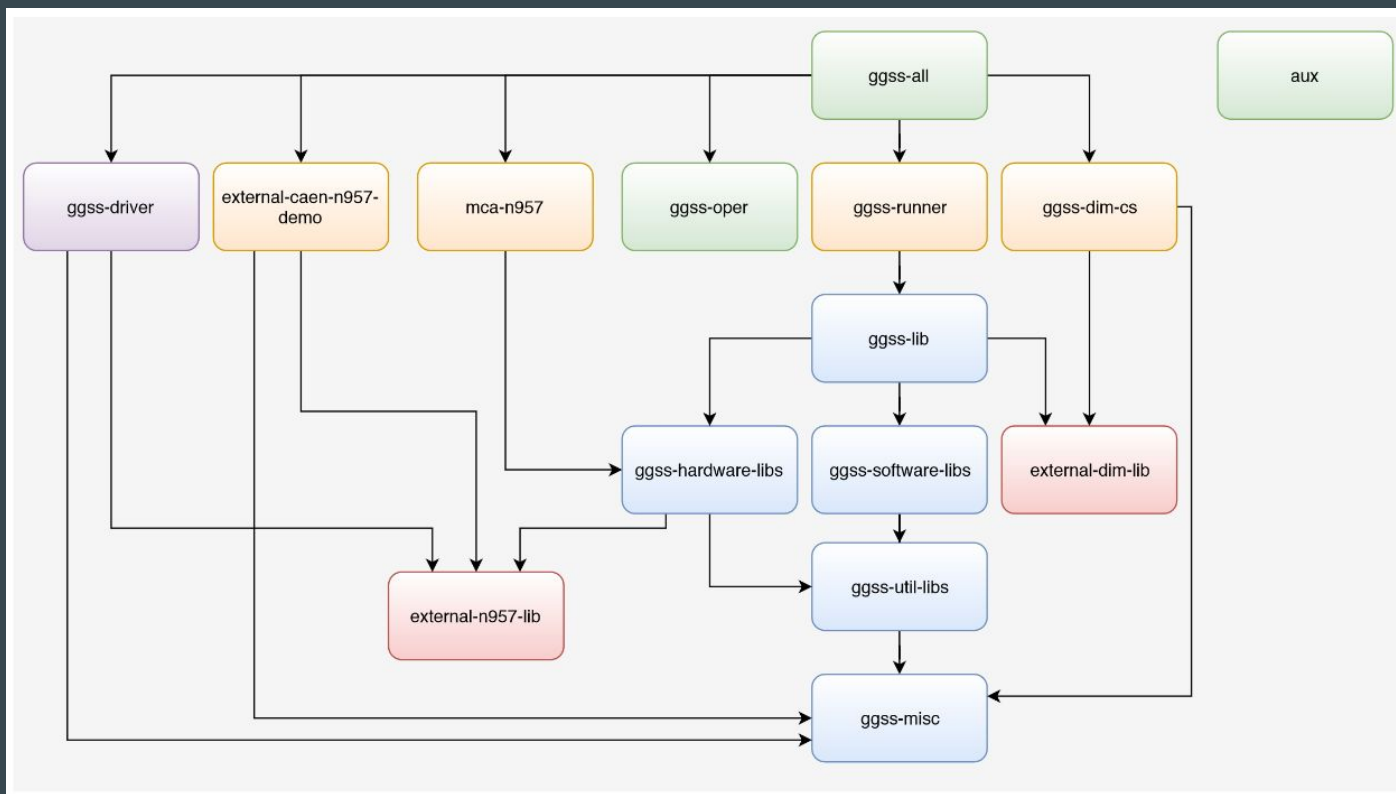


Diagram ilustrujący podział na moduły w projekcie GGSS oraz zależności między nimi  
(źródło: materiały własne)



# Rozbudowa systemu wersjonowania i zarządzania zależnościami

- w celu zarządzania zależnościami wewnątrz projektu wykorzystano *git submodules*
- ze względu na braki w API gita przygotowano skrypty automatyzujące

## Parent Repository

commit\_1

commit\_2

Depends on

## Child Repository

commit\_A

commit\_B  
(this version is used  
by parent repository)

commit\_C

commit\_D  
(the newest version  
on remote repository)

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)

# Rozbudowa systemu wersjonowania i zarządzania zależnościami

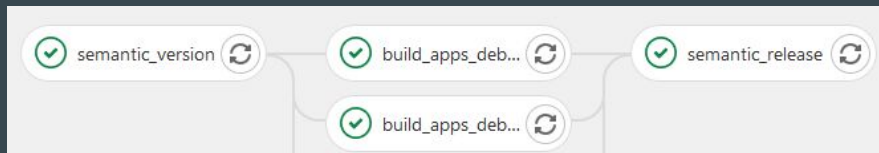
- w celu ujednolicenia wersjonowania w projekcie przystosowano *pipeline CI/CD* do pracy z “*eslint semantic-versioning*”.



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
```

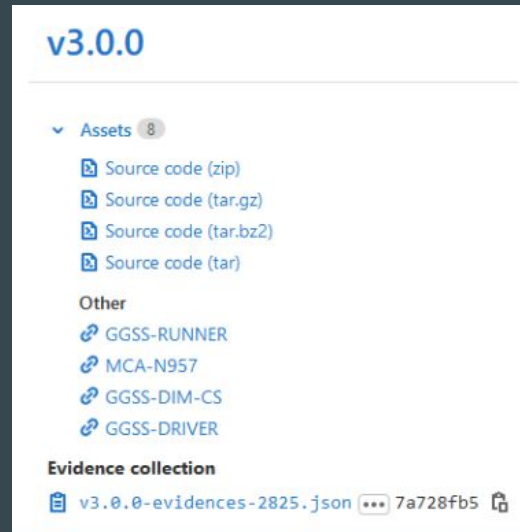
Działanie semantic-release  
(źródło: materiały własne)



Pipeline realizujący automatyczne wersjonowanie  
(źródło: materiały własne)

# System wdrażania w środowisko produkcyjne

- w celu szybkiego wdrażania w środowisko produkcyjne, w ramach *pipeline CI/CD* tworzone będą pakiety RPM
- pakiety będą zawierały gotowe do użycia aplikacje GGSS oraz logikę pozwalającą na ich poprawną instalację
- możliwe jest również pobranie pojedynczych aplikacji projektu GGSS



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

# Narzędzia do testowania warstwy sprzętowej

- system GGSS wymaga do prawidłowej pracy połączenia z kilkoma fizycznymi urządzeniami (np.: zasilacz wysokiego napięcia)
- w.w. urządzenia muszą być sprawne przed uruchomieniem GGSS
- początkowo w projekcie przygotowane było nieliczne oprogramowanie pozwalające na sprawdzenie działania urządzeń
- w trakcie przygotowania są ustandaryzowane narzędzia pozwalające w prosty sposób testować sprawność urządzeń w oparciu o scenariusze

## HighVoltageTestScenario:

```
- exec hv hv_caen_n1470_0:1 set vset 1368
- check hv hv_caen_n1470_0:1 mon vset 1368
- check hv hv_caen_n1470_0:1 mon vmon 1368 10
- exec hv hv_caen_n1470_0:1 set vset 0
- check hv hv_caen_n1470_0:1 mon vset 0
- check hv hv_caen_n1470_0:1 mon vmon 0 1
```

# Testy działania nowej wersji systemu

- ze względu na wymóg wysokiej niezawodności GGSS każda zmiana w systemie wymaga, oprócz testów automatycznych, wymaga dodatkowych testów manualnych w środowisku docelowym
- 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 wskazaniem samych urządzeń
- tego typu testy wykonywane są regularnie, po każdej większej zmianie

# Migracja systemu na nową infrastrukturę sprzętową

- jednym z celów pracy magisterskiej jest migracja systemu na nową infrastrukturę sprzętową
- wykonanie tej części 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

**Dziękujemy za uwagę.**