AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Rozbudowa i uaktualnienie systemu GGSS detektora ATLAS TRT

Arkadiusz Kasprzak
Jarosław Cierpich

Opiekun pracy: dr hab. inż. Bartosz Mindur, prof. AGH

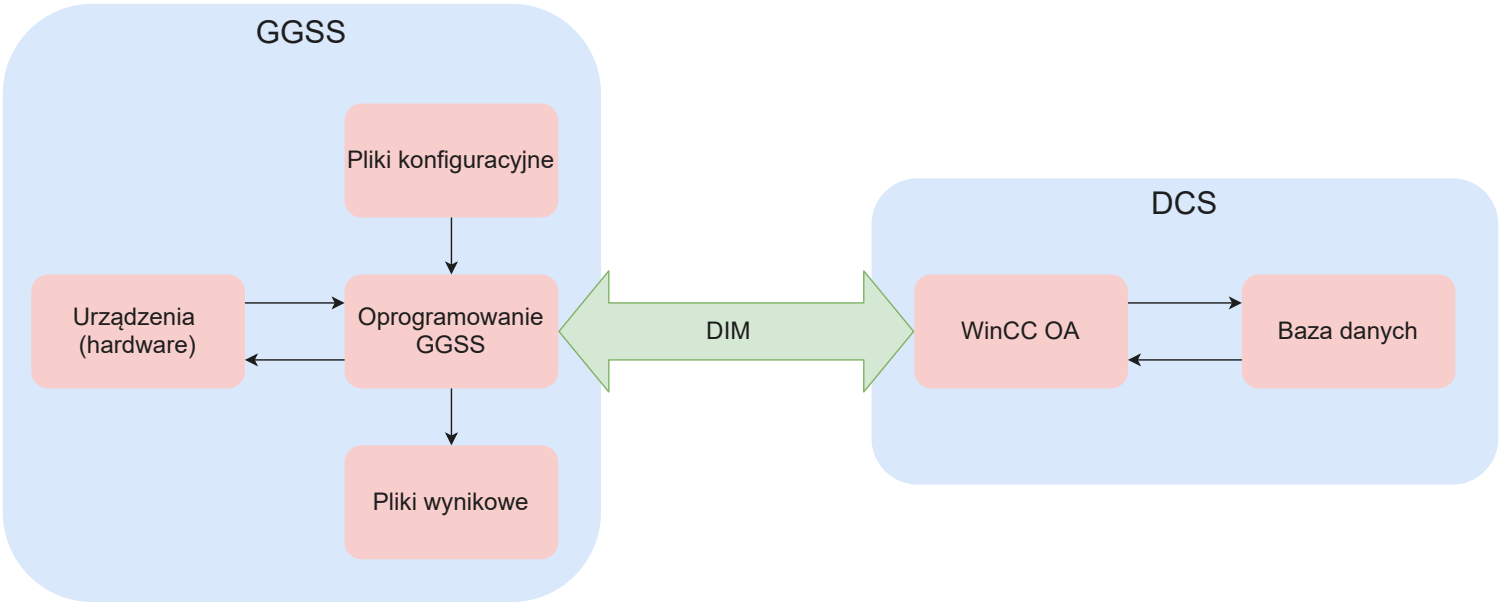
Plan prezentacji - Arkadiusz Kasprzak

1. Wprowadzenie do tematyki pracy
2. Modyfikacja kodu źródłowego projektu
3. Modyfikacja systemu budowania projektu

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 TRT (*Transition Radiation Tracker*) będącego częścią ATLAS
- składa się z warstwy oprogramowania (aplikacje oraz infrastruktura) oraz sprzętu (zasilacz wysokiego napięcia, multiplekser analogowy, wielokanałowy analizator amplitudy MCA)
- warstwa oprogramowania koordynuje działanie urządzeń i umożliwia sterowanie nimi za pomocą specjalnych komend
- aplikacje i biblioteki napisane z wykorzystaniem języka C++, infrastruktura: CMake, Bash oraz Python

Wprowadzenie do systemu GGSS



Rysunek: Wysokopoziomowa architektura systemu GGSS

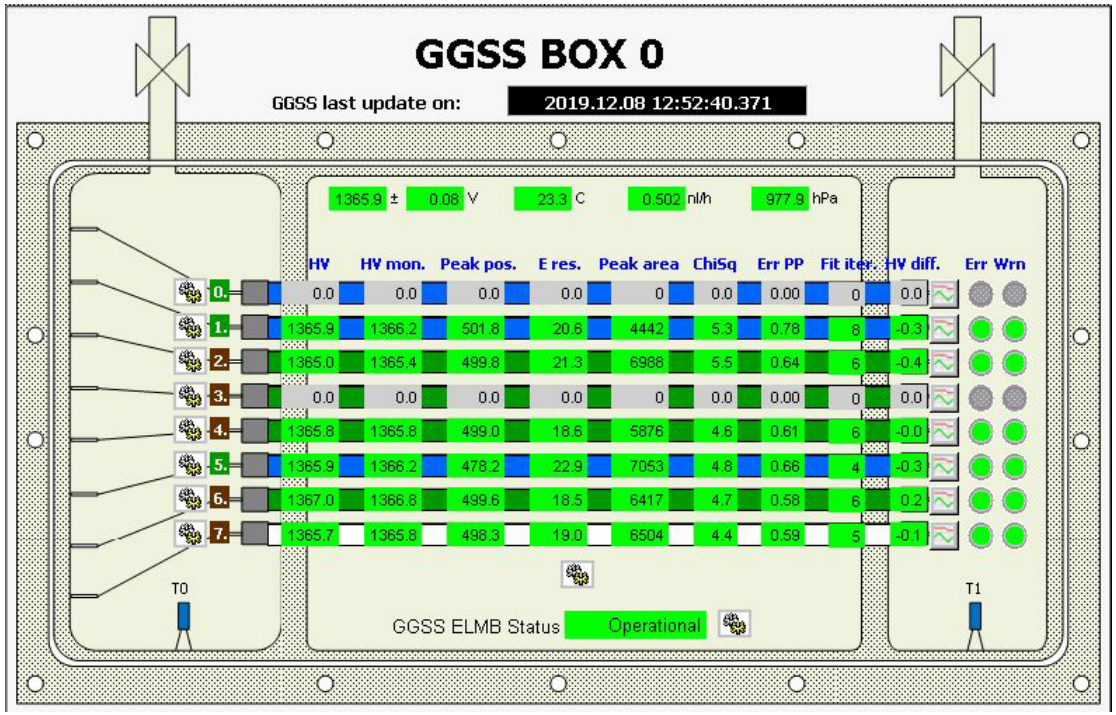
- WinCC OA - system system typu SCADA pozwalający na sterowanie i kontrolę działania podsystemów detektora
- DIM - protokół komunikacyjny dla aplikacji rozproszonych

Cele pracy

- kontynuacja pracy inżynierskiej, skupiającej się na infrastrukturze projektu
- rozbudowa przygotowanych w ramach wspomnianej pracy rozwiązań
- główny nacisk na aplikację *ggss-runner*, stanowiącą trzon systemu
- poprawa jakości kodu źródłowego oraz wprowadzenie nowych funkcjonalności
- rozbudowa infrastruktury pozwalającej na testowanie projektu
- udokumentowanie projektu (dokumentacja kodu źródłowego oraz pliki instruktażowe)
- migracja projektu - nowy komputer docelowy (początkowo planowany wyjazd do CERN, ostatecznie zrealizowana zdalnie)
- konieczność zachowania wysokiej niezawodności systemu - stosowanie praktyk takich jak *code review* i testy automatyczne

Aplikacja *ggss-runner*

- wielowątkowa aplikacja napisana z wykorzystaniem C++ i pakietu Boost
- zadanie: cykliczne gromadzenie danych w postaci widma poprzez komunikacje z wielokanałowym analizatorem amplitudy oraz wyznaczanie na ich podstawie wartości napięcia, komunikacja z systemem WinCC OA



Rysunek: Panel dostępny w ramach systemu WinCC OA

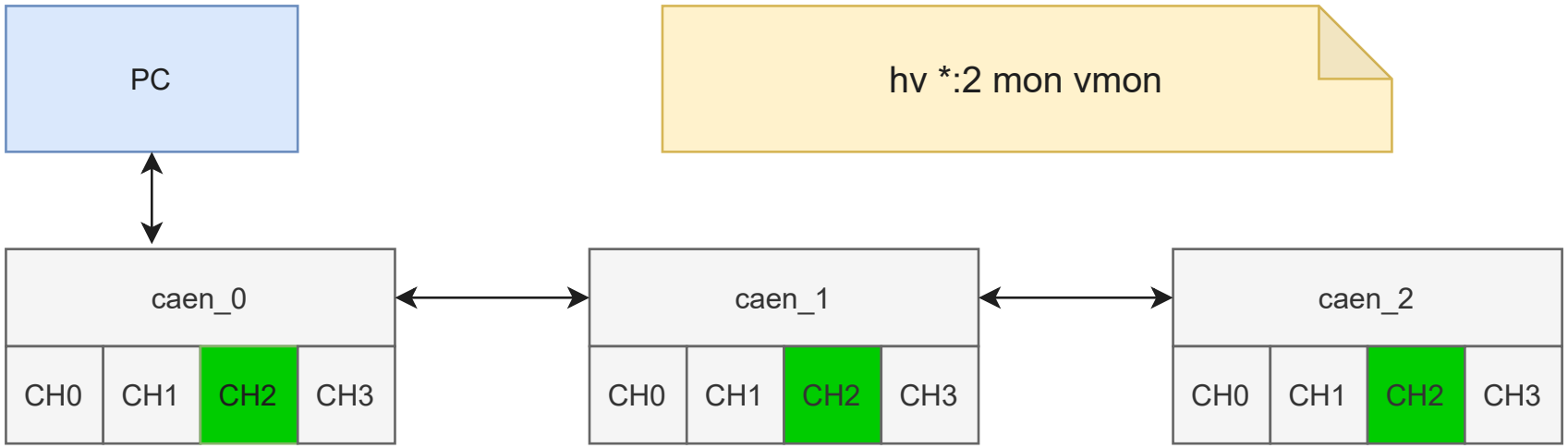
Modernizacja i poprawa jakości kodu źródłowego

- pierwotnie kod wykorzystywał standard C++03 oraz elementy C++11
- przeprowadzono migrację do standardu C++11 (pętle zakresowe, silne typy wyliczeniowe itd.)
- migracja niepełna z uwagi na ograniczenia kompilatora
- ujednolicono konwencje stosowane w kodzie (formatowanie, nazewnictwo)
- zlikwidowano niewykorzystywane lub wykomentowane fragmenty kodu
- usunięto nieliczne błędy (biblioteki *xml-lib* oraz *log-lib*)
- znaczne zmiany w 12 z 14 bibliotek wchodzących w skład systemu

Wprowadzone funkcjonalności

- dodanie obsługi zaawansowanych komend dla zasilaczy wysokiego napięcia
- rozbudowa biblioteki odpowiedzialnej za dopasowanie krzywej do zebranych danych
- dodanie możliwości aktualizacji parametrów i zebranych danych na żądanie
- dodanie zabezpieczenia przed przepełnieniem bufora urządzenia MCA
- dodanie możliwości przywracania domyślnej kolejności liczników słomkowych

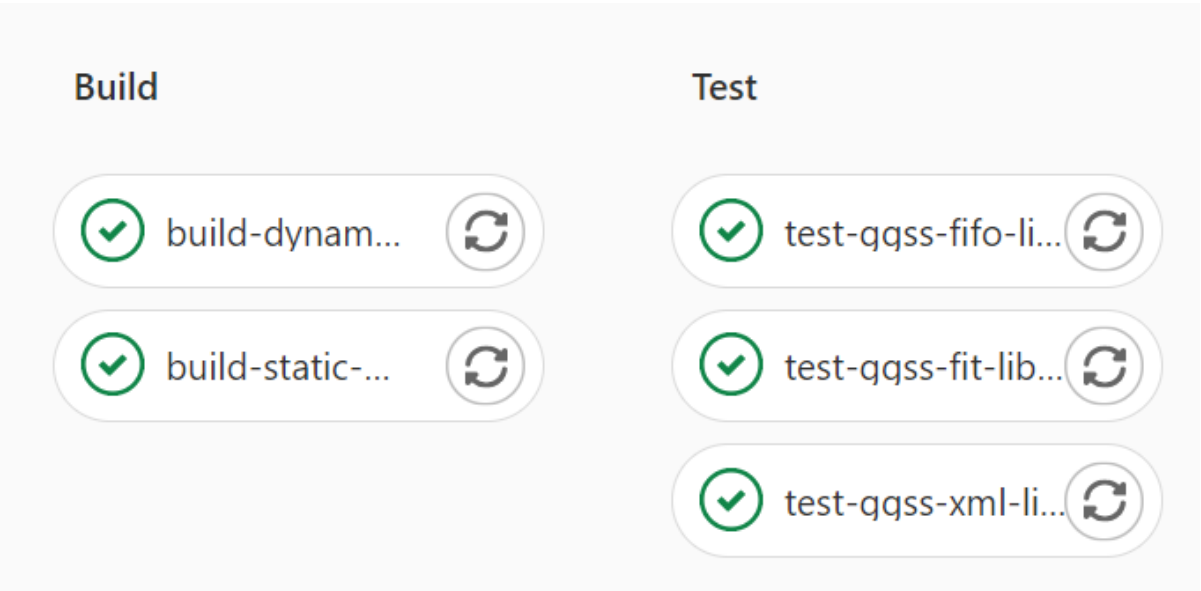
Przykład: nowe komendy dla zasilaczy



Rysunek: Schemat przybliżający sposób działania nowego formatu komend.

Testy automatyczne

- w celu zapewnienia niezawodności i szybkiego wykrywanie błędów wykorzystano testy automatyczne i metodykę TDD (*Test Driven Development*)
- testy przygotowane z wykorzystaniem biblioteki Boost.Test
- uruchamiane automatycznie po wdrożeniu zmian do repozytoriów projektu



Rysunek: *Pipeline CI/CD* wykonujący testy automatyczne po wdrożeniu zmian

Modyfikacja systemu budowania projektu

- system oparty o narzędzie CMake, przygotowany w ramach pracy inżynierskiej
- jego zadaniem jest obsługa projektu o hierarchicznej strukturze
- wsparcie dla testów automatycznych oraz generowania dokumentacji za pomocą narzędzia Doxygen
- system oparty o tzw. szablony - pliki *.cmake* zawierające często wykorzystywane fragmenty kodu
- zmiana sposobu implementacji szablonów - wykorzystanie funkcji i makr

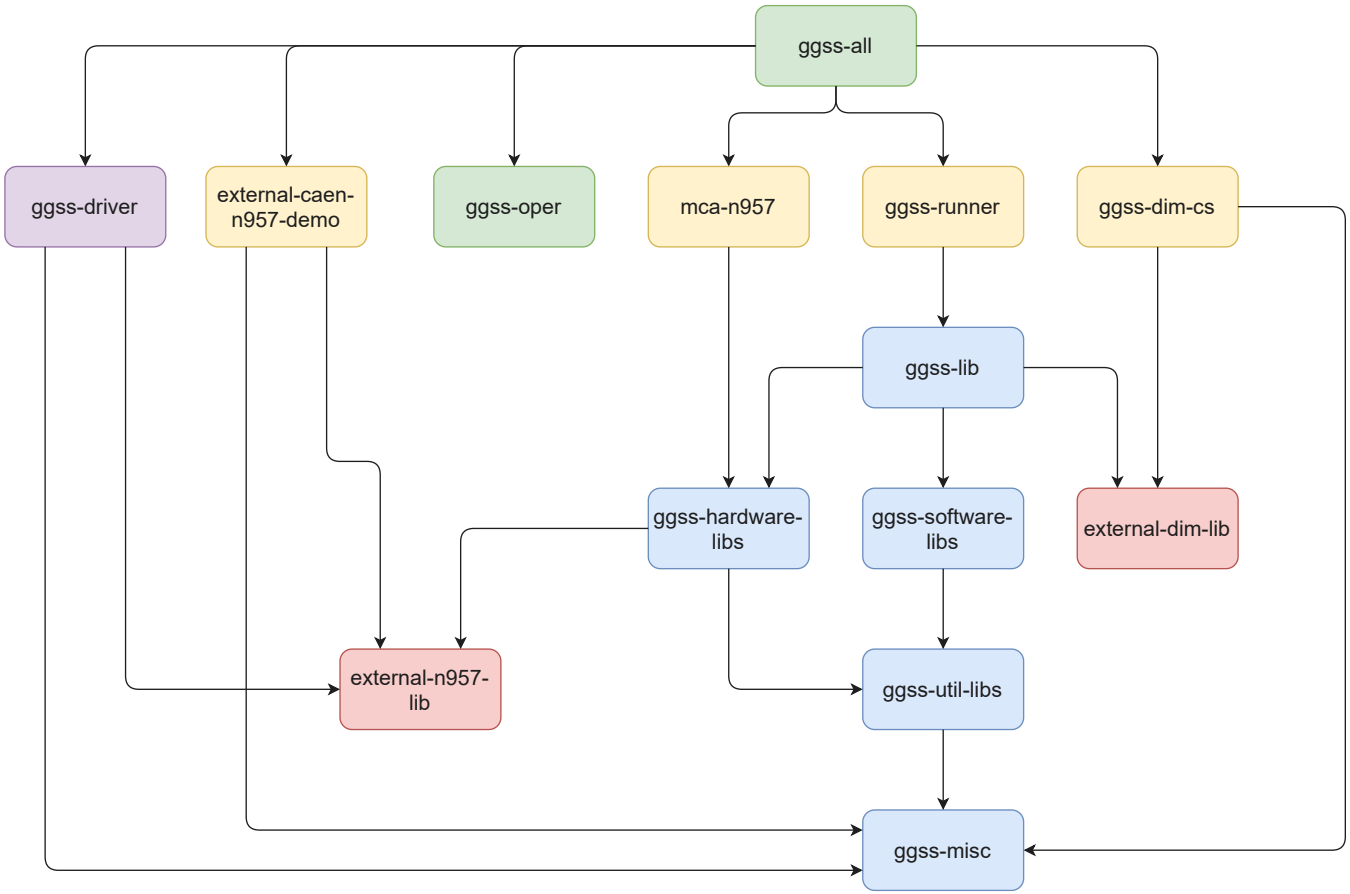
```
ggss_build_static_library(
    TARGET_NAME "thread"
    DEPENDENCY_PREFIX "${CMAKE_CURRENT_SOURCE_DIR}/.."
    DEPENDENCIES "sigslot" "ggss-util-libs/log"
)
```

Plan prezentacji - Jarosław Cierpich

4. Modyfikacja architektury projektu
5. Modyfikacja infrastruktury projektu
6. Aplikacje do testów urządzeń
7. Testy projektu
8. Podsumowanie

Modyfikacja architektury projektu

Poprzednia architektura projektu



Rysunek: Architektura projektu na podstawie modułów oraz ich powiązań

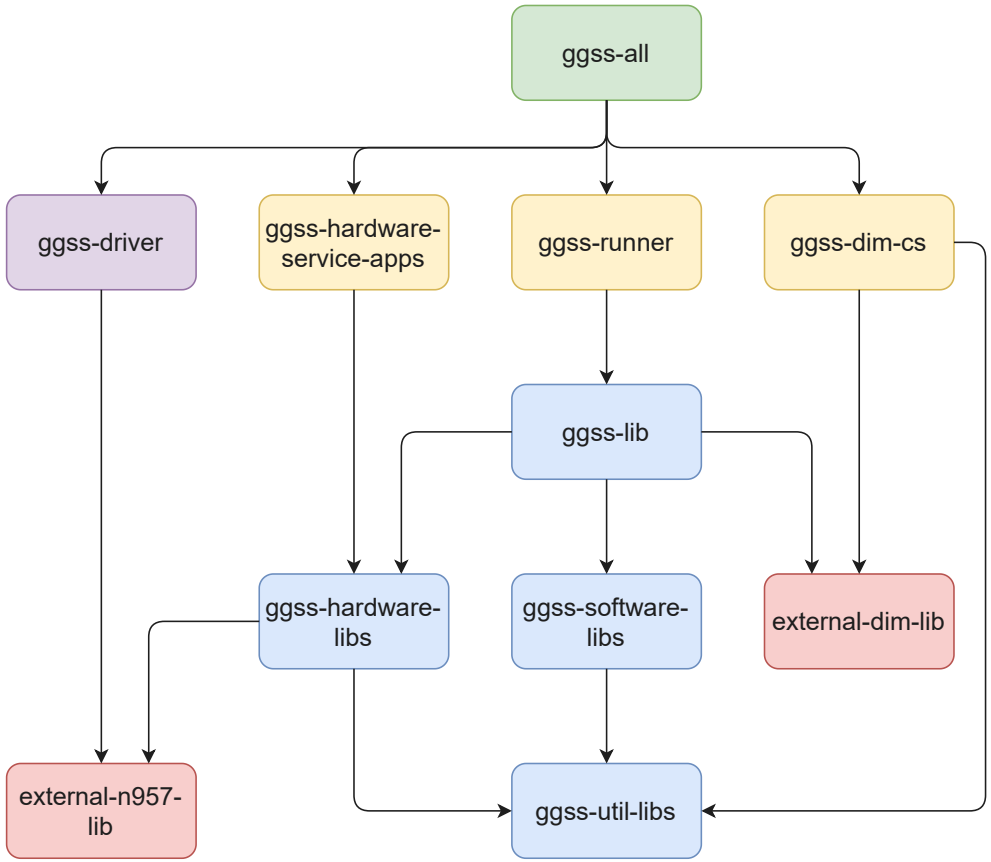
Modyfikacja architektury projektu

Wprowadzone zmiany

- zmniejszenie złożoności poprzez zmniejszenie ilości modułów
- pozbycie się modułów, które, w trakcie prac, okazały się niepotrzebne (na przykład: *ggss-misc*)
- wydzielenie powielonego kodu do nowej biblioteki zawartej w ramach repozytorium *ggss-hardware-libs*
- gałęzie *legacy* oraz archiwizacja projektów

Modyfikacja architektury projektu

Nowa architektura projektu



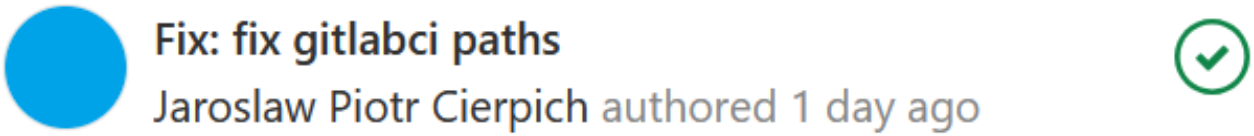
Rysunek: Architektura projektu na podstawie modułów oraz ich powiązań

Mechanizm ciągłej integracji i dostarczania

- mechanizm pozwalający na automatyczne testowanie, budowanie oraz przygotowywanie kodu do dostarczania do środowiska docelowego
- pierwsza wersja została utworzona w ramach pracy inżynierskiej, następnie została ona rozwinięta w ramach pracy magisterskiej
- do mechanizmu zostało dodane tworzenie artefaktów, a wraz z wprowadzeniem testów automatycznych został dodany krok *test*
- automatyzacji poddane zostało również wersjonowanie projektu

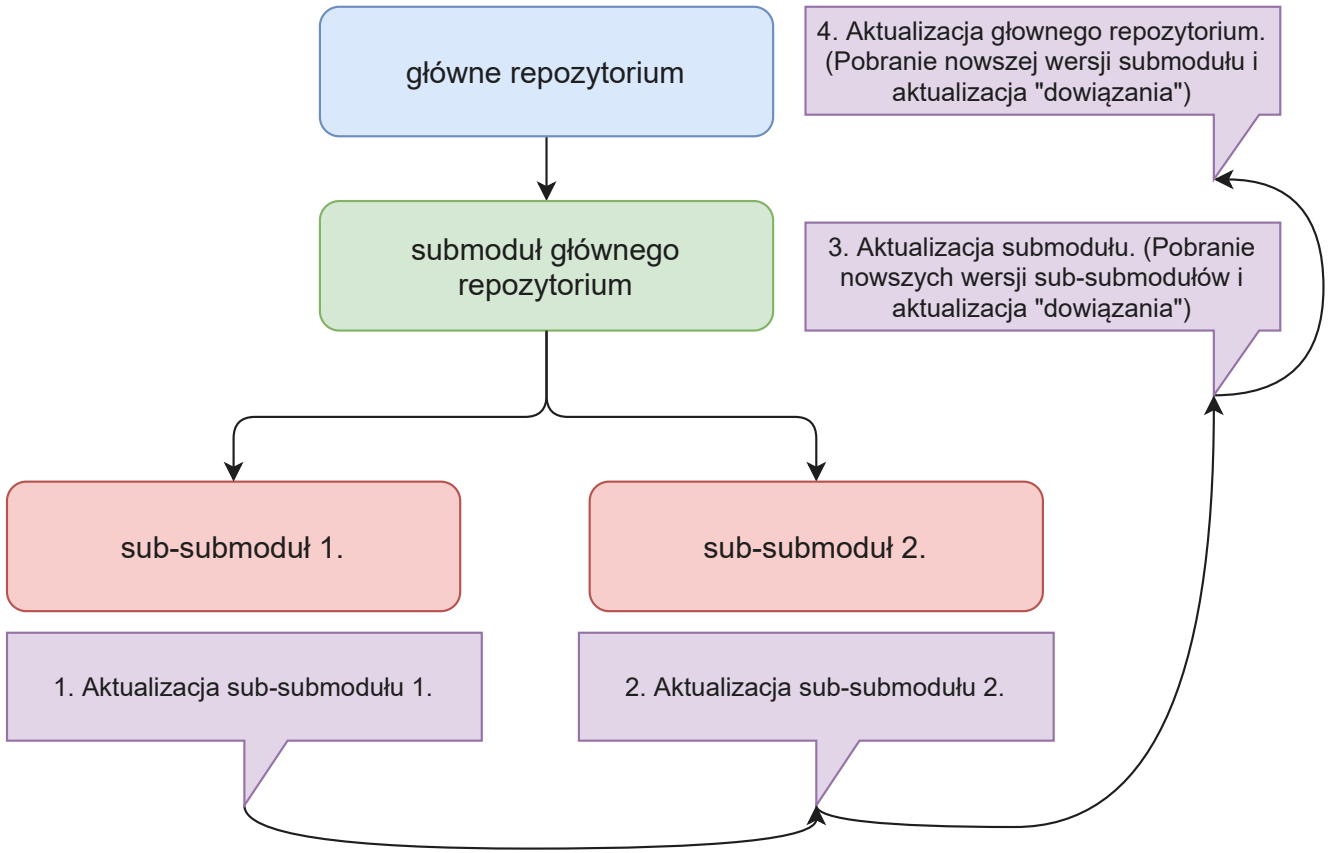
Automatyzacja i centralizacja wersjonowania projektu

- automatyzacja oparta o rozszerzenia:
  - plików *.cmake* w celu wsparcia parametru version
  - skryptu *build.py*, który jest nakładką narzędzia CMake
  - mechanizmu ciągłej integracji w celu automatycznego generowania wersji na podstawie wiadomości zawartych w ramach rewizji
- rozwiązanie oparte o standard *semantic-versioning*, bibliotekę *semantic-release* oraz konwencję oznaczania wiadomości zgodną z *eslint-semantic-release*



Rysunek: Rewizja zgodna z konwencją *eslint-semantic-release*

Automatyzacja pracy z submodułami



Rysunek: Opis kroków jakie należy podjąć, aby wprowadzone zmiany były widoczne w całym projekcie.

Pozostałe zmiany

- zaktualizowane zostały skrypty operacyjne, czyli skrypty obsługujące projekt GGSS w środowisku docelowym
- nowy skrypt monitorujący zasoby wykorzystywane przez główną aplikację projektu GGSS został utworzony
- przygotowana została aplikacja *high-voltage-killer*, która zabezpiecza system poprzez wyłączenie zasilania na zasilaczu wysokiego napięcia
- dokonano modyfikacji struktury katalogów oraz nazw w środowisku docelowym

Aplikacje do testów urządzeń

- przygotowanie aplikacji służących do testowania urządzeń - potrzeba poprawnie skonfigurowanej i działającej warstwy sprzętowej
- istniejące rozwiązania o ograniczonych możliwościach i niejednolitym interfejsie
- przygotowane rozwiązanie pozwala na dogłębne przetestowanie urządzeń z wykorzystaniem dwóch trybów: interaktywnego oraz scenariuszowego
- w celu dokonania odpowiednich testów wymagana jest jedynie wiedza domenowa

Aplikacje do testów urządzeń

Listing 1: Przykładowy scenariusz dla zasilacza wysokiego napięcia.

```
HighVoltageTestScenario:
  - hv module1:3 set vset 10.0
  - sleep 20000
  - assert hv module1:3 mon vset 10.0
  - assertTol hv module1:3 mon vmon 10.0 1.0
  - hv module1:3 set vset 0
  - sleep 20000
  - assert hv module1:3 mon vset 0
  - assert hv module1:3 mon vmon 0
```

Testy projektu

- testy nowych funkcjonalności w środowisku docelowym - po każdej znaczącej zmianie
- testy w trakcie i po migracji do nowego środowiska docelowego, rola autorów w migracji
- testy finalnej wersji projektu, testy zużycia zasobów - dedykowane skrypty oraz platforma Valgrind

Podsumowanie

- przygotowany system automatyzacyjny pozwala na łatwe oraz szybkie wprowadzanie nowych zmian w projekcie
- wszystkie rozwiązania cechują się stosowaniem ogólnie przyjętych standardów co w połączeniu z dużym naciskiem na odpowiednią dokumentację znacząco zmniejsza próg wejścia
- w celu zapewnienia poprawnego działania wszystkie zmiany zostały przetestowane w środowisku docelowym
- stosowanie nowoczesnych praktyk takich jak testy automatyczne pozytywnie wpłynęło na proces rozwoju oprogramowania