Politechnika Gdańska
Wydzial Elektroniki Telekomunikacji i Informatyki

Projektowanie Oprogramowania Systemów

Projekt oprogramowania systemu optymalizacji sterowania silnikiem DC

Autorzy:

Pawel Zarembski
Maciej Zielonka
Kamil Andrzejewski
Pawel Splawski

Prowadzacy zajecia:
dr inż. Arkadiusz Szewczyk

Czerwiec 2018

SPIS TREŚCI

1. WSTĘP

Niniejsze opracowanie dotyczy projektu oprogramowania systemu, którego zadaniem jest automatyczne wyznaczanie nastaw dla regulatora silników DC oraz ich optymalizacja względem określonego kryterium.
Przedstawiony w specyfikacji funkcjonalnej system umożliwiać będzie realizację następujących zadań :

- wprowadzanie przez użytkownika wymaganych parametrów silnika prądu stałego,
- automatyczną identyfikację parametrów silnika prądu stałego,
- wyznaczanie nastaw regulatora na podstawie zadanej trajektorii odpowiedzi silnika,
- optymalizacja nastaw za pomocą algorytmu genetycznego i danych z rzeczywistego obiektu,
- wizualizacja wartości wielkości pomiarowych podczas pracy silnika.

Poniżej przedstawiono specyfikacje funkcjonalną systemu w której zdefiniowano klienta oraz użytkowników końcowych systemu, przedstawiono wymagania funkcjonalne oraz scenariusze użycia systemu.

2. SPECYFIKACJA FUNKCJONALNA

Identyfikacja klienta oraz użytkowników końcowych

Potencjalnym klientem dla omawianego systemu mogą być wszelkie firmy i zakłady produkcyjne, które posiadają w swych liniach technologicznych napędy wykorzystujące silniki prądu stałego. Z racji podobnych, docelowych warunków pracy i realizowanych zadań systemu, firma ta może zajmować się produkcją układów elektronicznych lub mechanicznych, których proces technologiczny wymaga realizacji określonego ruchu z zadanymi parametrami.

Docelowym użytkownikiem systemu będzie więc osoba na stanowisku technika utrzymania ruchu lub inżyniera automatyka, której zadaniem jest przygotowanie odpowiedniego układu sterowania silnikiem DC, umożliwiającego realizację zadanych przemieszczeń elementów maszyn.

Zaletą systemu z punktu widzenia potencjalnego klienta - firmy produkcyjnej, jest znaczne ułatwienie i uproszczenie skomplikowanego procesu strojenia regulatorów napędów, a co za tym idzie otrzymanie zadowalających wyników i efektów sterowania w procesie produkcyjnym, bez potrzeby zatrudnienia wysoko i ściśle wykwalifikowanej kadry inżynierów.

Podsumowując, dla omawianego systemu zdefiniować można następujące warunki docelowe:

- **Potencjalny klient** - firma produkcyjna Drutex,
- **Docelowe środowisko pracy** - hala produkcyjna, otoczenie zautomatyzowanej linii technologicznej,
- **Użytkownik końcowy** - dorosła osoba na stanowisku technika lub inżyniera utrzymania ruchu.

Wymagania funkcjonalne i pozafunkcjonalne systemu

Na podstawie ogólnej koncepcji założeń pracy systemu, zdefiniowano jego komponenty składowe, wymagania fukcjonalne i pozafunkcjonalne które opisują cechy oprogramowania i zadania które powinno ono realizować.

W strukturze omawianego systemu wyszczególnić można następujące komponenty składowe:

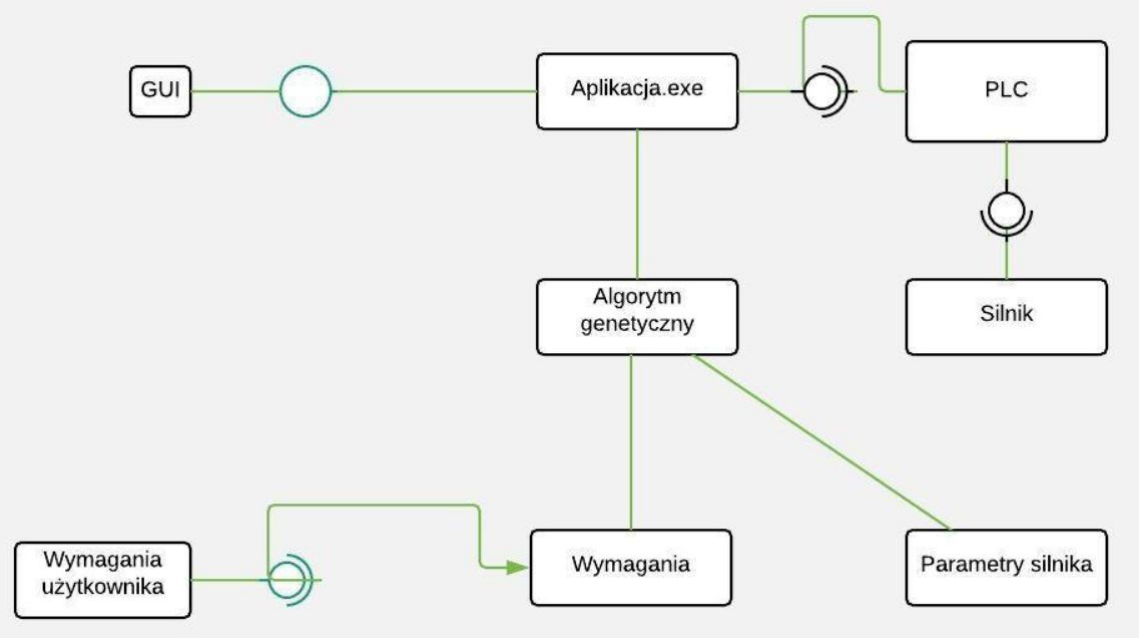
- **użytkownik** - osoba na stanowisku technika utrzymania ruchu,
- **urządzenie HMI** - komputer PC, laptop lub tablet umożliwiający uruchomienie aplikacji i komunikację z użytkownikiem,
- **aplikacja** - główny program realizujący zadanie optymalizacji nastaw, uruchomiony na urządzeniu HMI,
- **sterownik PLC** - podrzędne urządzenie skomunikowane z HMI, wykonujące zadania sterująco-pomiarowe na silniku DC,
- **silnik DC** - docelowy obiekt rzeczywisty.

Dla omawianego systemu określono wymagania funkcjonalne, definiujące zadania realizowane przez poszczególne komponenty składowe oraz zintegrowaną całość :

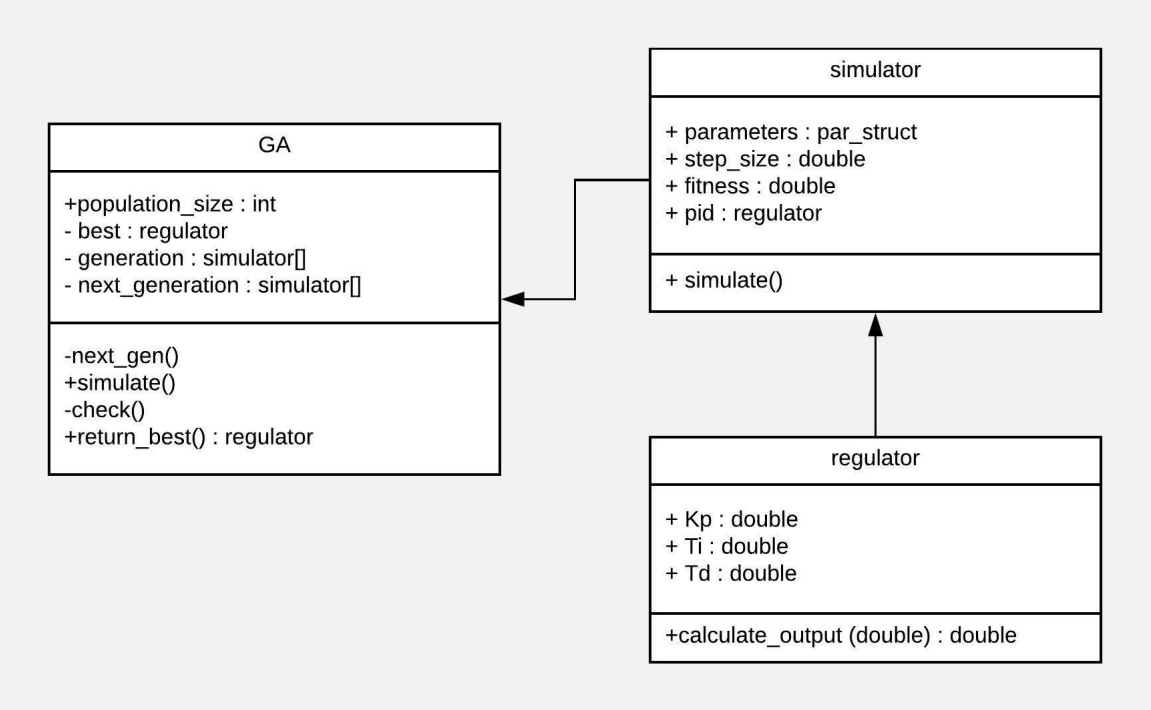
1. Główną funkcjonalnością systemu jest wyznaczenie optymalnych nastaw regulatora PID w systemie sterowania prędkością silnika prądu stałego.
2. Parametry modelu silnika mogą być wprowadzone przez operatora przy użyciu interfejsu.
3. W przypadku nieposiadania przez operatora parametrów silnika, system powinien umożliwiać odczyt niezbędnych parametrów (np. stałej czasowej mechanicznej) z rzeczywistego modelu silnika, przy użyciu sterownika PLC.
4. Aplikacja powinna posiadać możliwość generowania wykresów przedstawiających przebieg wielkości sygnałów sterujących i pomiarowych.
5. System powinien zapewniać komunikację ze sterownikiem PLC odpowiedzialnym za sterowanie silnikiem i pomiar jego parametrów.
6. Aplikacja oprócz optymalizacji sterowania na modelu symulacyjnym, powinna umożliwiać weryfikacje otrzymanych wyników na obiekcie rzeczywistym i uwzględnienie ewentualnych odchyłek.

Wymagania pozafunkcjonalne zdefiniowane dla omawianego systemu są następujące:

1. Aplikacja powinna posiadać czytelny dla użytkownika interfejs graficzny, umożliwiający korzystanie w warunkach przemysłowych.
2. Na wykresach przebiegów sygnałów procesowych generowanych przez aplikację, każdy z sygnałów powinien posiadać unikalny kolor.
3. Optymalne nastawy regulatora PID wyznaczane będą z wykorzystaniem algorytmu genetycznego.
4. Projektowana aplikacja powinna być przenośna pomiędzy urządzeniami takimi jak PC, laptop, tablet, posiadającymi system operacyjny Windows.



Rys. 2.1. Struktura systemu i wyszczególnienie jego komponentów składowych

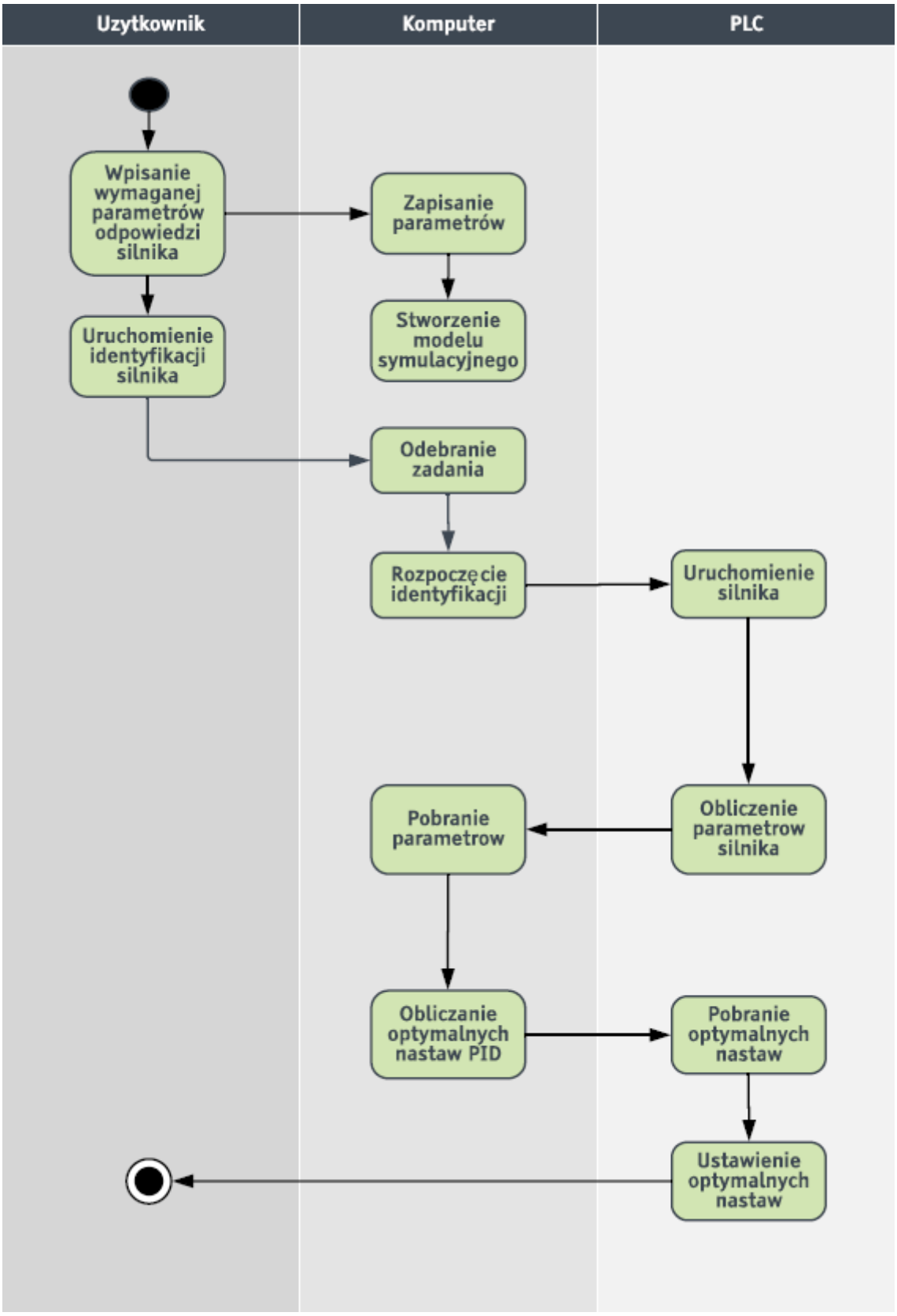


Rys. 2.2. Diagram przedstawiający strukturę klas w oprogramowaniu systemu

Na rysunkach 2.1 oraz 2.2 przedstawiono wyszczególnione komponenty całego systemu oraz klasy składowe jego oprogramowania.

Scenariusze użycia systemu

Przykładowy, zakładany scenariusz użycia systemu ukazujący interakcję pomiędzy jego poszczególnymi komponentami podczas realizacji określonych zadań, przedstawiony został na rysunku 2.3, znajdującym się na następnej stronie.



Rys. 2.3. Diagram aktywności dla omawianego systemu

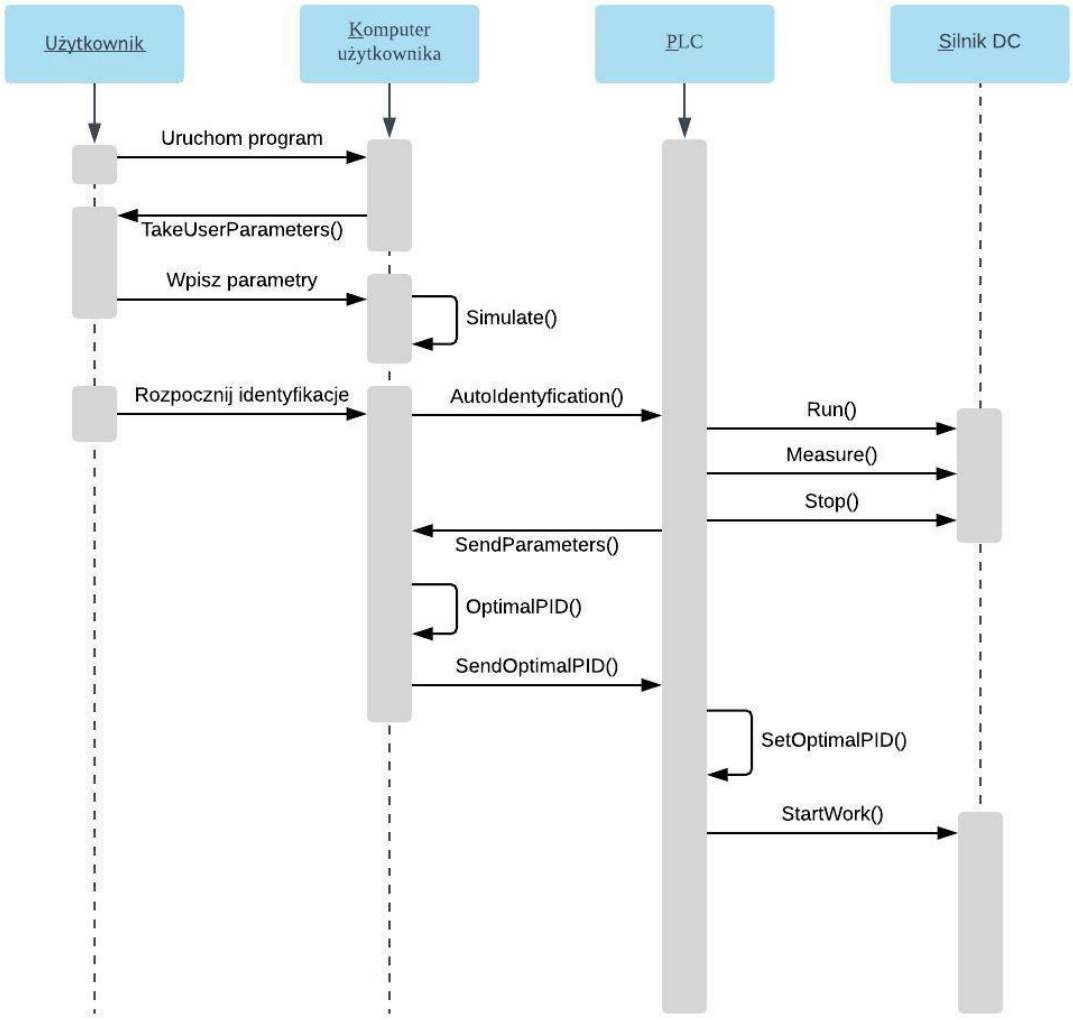
3. SPECYFIKACJA TECHNICZNA

Główna aplikacja opracowana zostanie przy użyciu środowiska Microsoft Visual Studio. Do implementacji głównej aplikacji zawierającej zaimplementowany algorytm genetyczny i interfejs użytkownika wykorzystane zostaną podstawowe biblioteki standardowe dostępne w środowisku Visual Studio.

Narzędziem dodatkowo wspomagającym opracowywanie oprogramowania systemu jest system kontroli wersji GIT - GitHub.

Sterownik PLC wykorzystywany do zrealizowania omawianego systemu to moduł S7-1200 firmy Siemens. Komunikacja pomiędzy stacją operatorską (PC, tablet, laptop) a sterownikiem PLC realizowana będzie przy użyciu sieci Ethernet.

Kluczowe elementy omawianego systemu przedstawiono na wcześniej zamieszczonych diagramach UML.Poniżej umieszczono diagram UML ukazujący interakcje pomiędzy składowymi elementami oprogramowania.



Rys. 3.1. Diagram UML przedstawiający interakcje pomiędzy składowymi oprogramowania i systemu

Harmonogram i podział pracy

Realizacja omawianego systemu, którego charakterystykę i opis przedstawiono w niniejszym opracowaniu wymaga odpowiedniego zaplanowania prac zespołu, które umożliwią zakończenie całego projektu sukcesem. Harmonogram prac związanych z opracowaniem oprogramowania i realizacją całości systemu przedstawiono w tabeli 3.1 znajdującej się na następnej stronie.

Na podstawie zdefiniowanych wymagań dotyczących funkcjonalności systemu zdefiniowano następujący podział pracy w zespole:

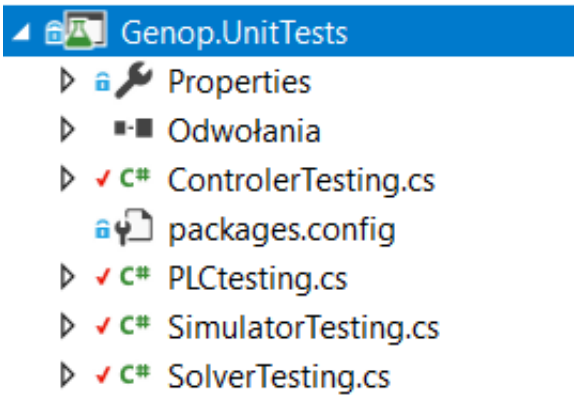
- **Kamil Andrzejewski** - prowadzenie porządku w repozytorium - GIT maintainer, komunikacja PLC
- **Maciej Zielonka** - implementacja algorytmu genetycznego, implementacja aplikacji optymalizującej
- **Paweł Zarembski** - zaprojektowanie i implementacja interfejsu użytkownika
- **Paweł Spławski** - tworzenie dokumentacji projektu, opracowanie modelu silnika DC

Tabela 3.1. Harmonogram prac związanych z realizacją projektu

Lp.	Data zakończenia zadań	Zadania do zrealizowania
1.	16.05.2018	1. Określenie użytkowników końcowych systemu, na podstawie rozmów z klientem. 2. Zdefiniowanie wymagań funkcjonalnych i pozafunkcjonalnych dla opracowywanego systemu. 3. Opracowanie dokumentacji funkcjonalnej, zawierającej powyższe informacje oraz scenariusz użycia systemu i odpowiednie diagramy UML. 4. Prezentacja aktualnych efektów prac nad projektem przed docelowym klientem.
2.	23.05.2018	1. Utworzenie oraz odpowiednia organizacja repozytorium projektu w serwisie GitHub.com. 2. Wybór języka programowania, środowiska IDE, bibliotek zastosowanych do opracowania oprogramowania. 3. Opracowanie specyfikacji technicznej zawierającej m.in. modele UML kluczowych elementów systemu. 4. Prezentacja aktualnych efektów prac nad projektem przed docelowym klientem.
3.	06.06.2018	1. Opracowanie modelu matematycznego silnika DC. 2. Rozpoczęcie opracowywania oprogramowania aplikacji i jej interfejsu graficznego. 3. Rozpoczęcie implementacji optymalizatora nastaw i komunikacji z PLC. 4. Opracowanie dokumentacji wykonanych prac. 5. Konsultacja otrzymanych wyników prac z docelowym klientem.
4.	13.06.2018	1. Wykonanie odpowiednich cykli testów jednostkowych oprogramowania. 2. Profilowanie aplikacji i próba optymalizacji hot spotów. 3. Odpowiednie udokumentowanie przebiegu wykonywanych testów. 4. Przedstawienie klientowi dotychczasowych wyników i wykonanie ewentualnych poprawek.
5.	30.06.2018	1. Implementacja aplikacji na rzeczywistym obiekcie. 2. Wykonanie serii testów działania systemu w warunkach docelowych. 3. Naniesienie poprawek do oprogramowania systemu i ich dokumentacja. 4. Oddanie systemu do użytku klienta.

4. TESTY JEDNOSTKOWE - RAPORT

W celu zweryfikowania poprawności działania pojedynczych elementów programu takich jak metody klas i obiekty została wykorzystana metoda testowania tworzonego oprogramowania poprzez wykonywanie testów jednostkowych *(ang. unit tests)*.
Do pliku rozwiązania w środowisku Visual Studio został dodany projekt, który składa się z klas odpowiednio testujących poszczególne klasy głównego projektu oprogramowania. Drzewo tego projektu przedstawiono na poniższym rysunku 4.1.



Rys. 4.1. Drzewo projektu oprogramowania testów jednostkowych

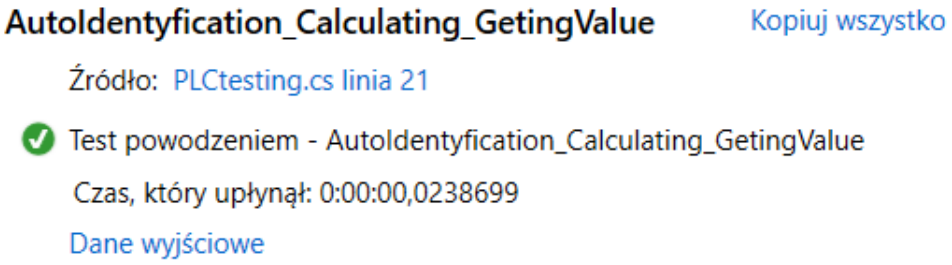
Poniżej przedstawiono listing przykładowej klasy wykonującej testy jednostkowe metod występujących w klasie impelemtującej obiekt sterownika PLC.

```
1  namespace Genop.UnitTests
2  {
3      /**
4       * Klasa wykonujaca testy jednostkowe
5       * niektorych z metod wystepujacych w
6       * klasie PLC.
7       */
8      [TestClass]
9      public class PLCTests
10     {
11         /**
12          * Metoda testujaca metode
13          * AutoIdentyfication(), kalkulujaca dane
14          * i zwracajaca wartosc liczbowa
15          */
16         [TestMethod]
17         public void
               AutoIdentyfication_Calculating_GetingValue
               ()
18         {
19             // Arrange
20             PLC plc1 = new PLC();
21
```

```
22                  // Act
23                  var resoult = plc1.AutoIdentyfication();
24
25                  // Assert
26                  Assert.IsNotNull(resoult);
27              }
28          }
29  }
```

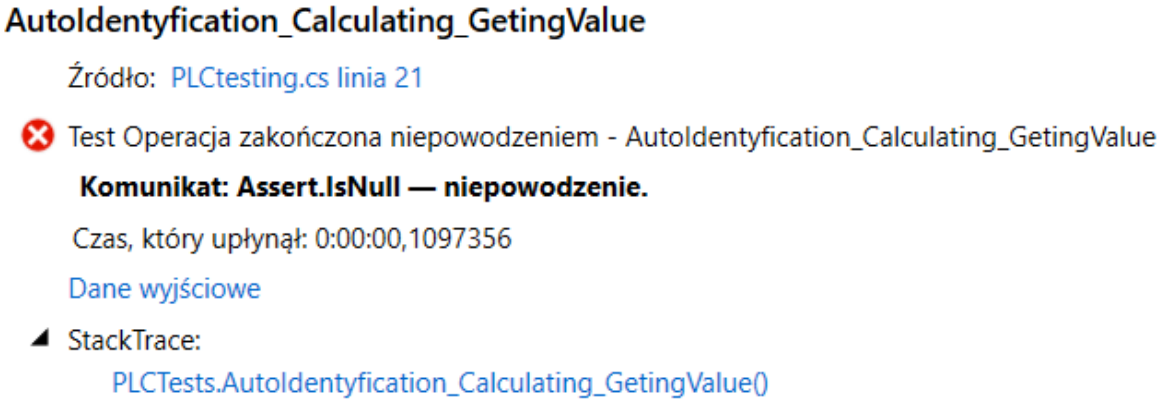
Klasa oznaczona jako [TestClass] wskazuje na to, że zajmuje się ona testami. Podobnie oznaczenie metody [TestMethod] mówi o tym, że dana metoda jest testem. Test przeprowadza się zgodnie z konwencją AAA, czyli Arrange, Act and Assert. W miejscu Arrange, program zajmuje się stworzeniem zmiennych pomocniczych oraz testowanych obiektów, Act oznacza wykonanie wymaganych czynności testujących, natomiast na końcu wykonywana jest sekcja Assert - czyli porównanie wyników z wymaganiami danej metody.

Na poniższym rysunku 4.2 podano przykład metody, która pozytywnie przeszła test jednostkowy. Zwrócona przez nią wartość była, zgodnie z założeniami, wartością nie pustą.



Rys. 4.2. Pozytywny wynik przeprowadzonego testu jednostkowego

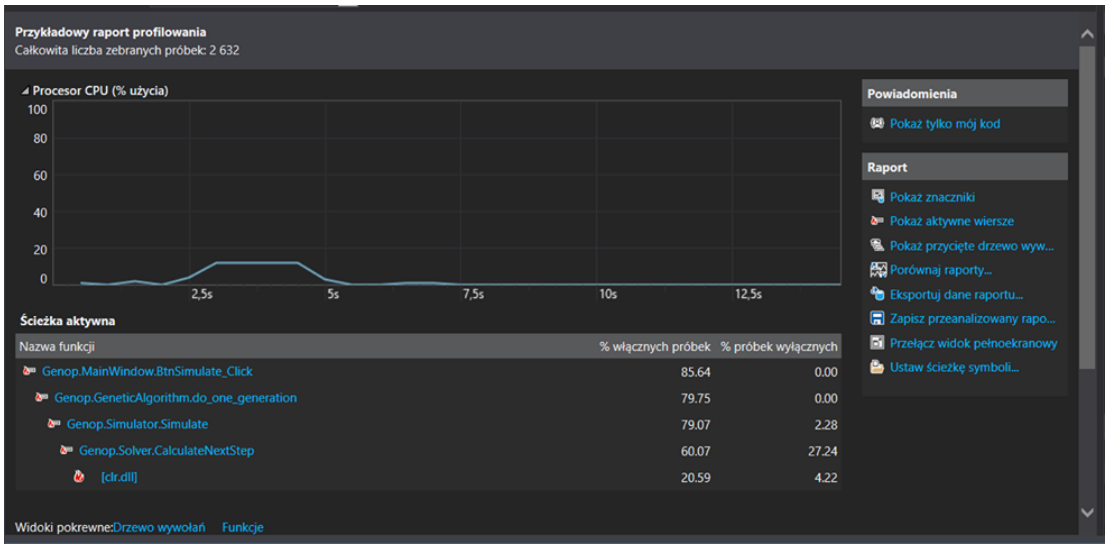
Na poniższym rysunku 4.3 podano przykład metody, która nie przeszła testu jednostkowego. Zwrócona przez nią wartość była nie zgodna z założeniami wartością pustą.



Rys. 4.3. Negatywny wynik przeprowadzonego testu jednostkowego

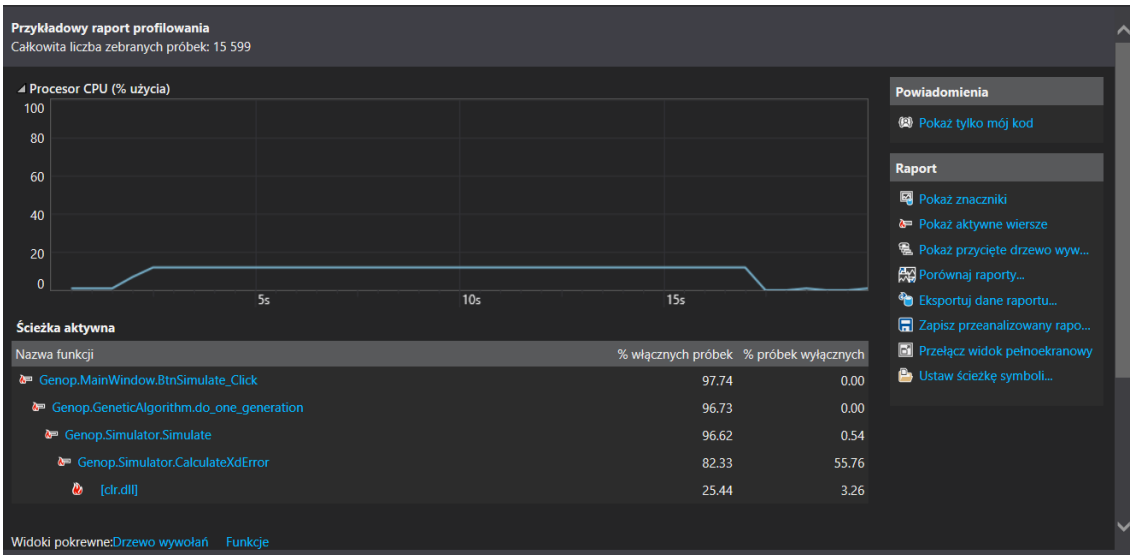
5. PROFILOWANIE OPROGRAMOWANIA - RAPORT

W celu przebadania opracowanego oprogramowania i skompilowanego programu pod względem wykonywania jego poszczególnych sekcji i występowania tzw. hot-spotów, wykorzystano narzędzie profilera dostępne w środowisku Visual Studio. Profiler pokazał, że najwięcej czasu procesor (program) spędza na wykonywaniu kroków algorytmu RK4 - sprawdzono czy zmienne służące do iterowania zadeklarowane jednorazowo jako pola klasy zamiast być tworzone z każdym obiegiem pętli skrócą w jakiś sposób czas wykonania programu. Wynik eksperymentu okazał się niejednoznaczy, jednak z całą pewnością można stwierdzić, że nie wpływa to na zużycie procesora.
Poniżej na rysunku 5.1 przedstawiono wynik przeprowadzonego badania profilerem w środowisku Visual Studio.



Rys. 5.1. Wynik przeprowadzonego testu występowania sekcji hot-spot

Optymalizacja funkcji *CalculateNextStep* była niemożliwa z racji zachowania odpowiednich wymagań związanych z realizacją numerycznego rozwiązywania równań różniczkowych.
Jednakże w metodzie *simulate* wywyłującej symulację, znajdował się względnie nieefektywny algorytm obliczania błędu sygnału sterującego. Algorytm ten został zastąpiony wydajniejszą metodą - wyznaczaniem wartości całki uchybu sterowania, co spowodowało znaczące skrócenie czasu wykonywania całej symulacji co zauważyć można na rysunku 5.1.
Wynik badania programu profilerem z zaimplementowanym nieoptymalnym algorytmem, przedstawiono na poniższym rysunku 5.2.



Rys. 5.2. Wynik badania profilerem z nieoptymalną sekcją programu