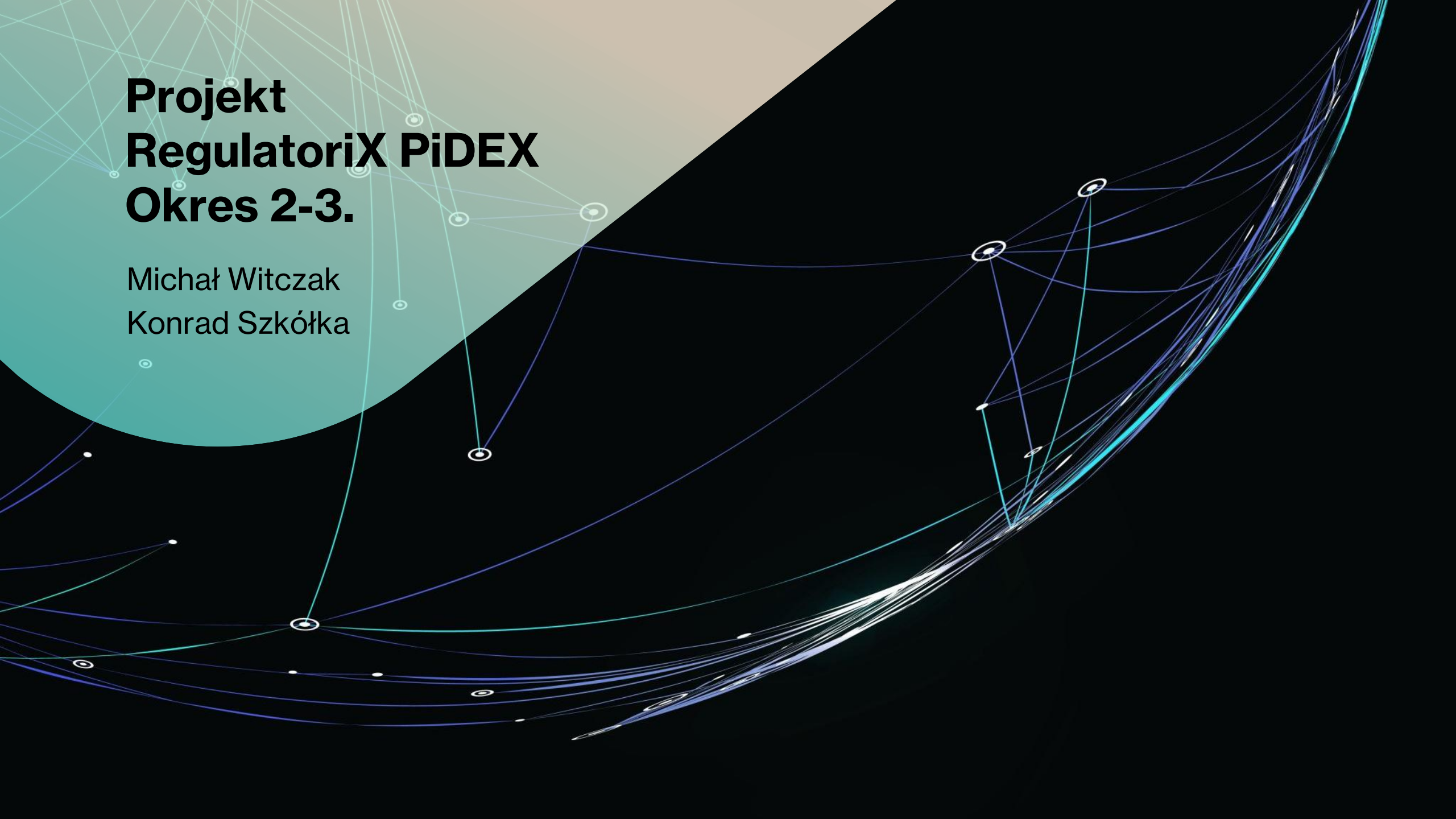
Projekt RegulatoriX PiDEX Okres 2-3.

Michał Witczak
Konrad Szkółka



NAJWIĘKSZE ZADOWOLENIE KLASA SYMULACJI

```cpp
void SymulacjaUAR::wykonajKrok()
{
    m_wartoscZadana = m_generator.generujWartosc();
    m_uchyb = m_wartoscZadana - m_poprzedniaWartoscRegulowana;
    m_sterowanie = m_regulator.symuluj(m_uchyb);
    m_wartoscRegulowana = m_model.symuluj(m_sterowanie);
    m_poprzedniaWartoscRegulowana = m_wartoscRegulowana;
}

void SymulacjaUAR::reset()
{
    m_regulator.resetPamieci();
    m_generator.reset();
    m_uchyb = 0.0;
    m_sterowanie = 0.0;
    m_wartoscZadana = 0.0;
    m_wartoscRegulowana = 0.0;
    m_poprzedniaWartoscRegulowana = 0.0;
}
```

NAJWIĘCEJ KŁOPOTÓW SPRAWIŁA KLASA:

```cpp
double Model_ARX::symuluj(double sygnalSterujacy)
{
    // Ograniczenie sterowania (jeśli aktywne)
    if (m_ogrSterowania)
        sygnalSterujacy = nasycenie(sygnalSterujacy, m_minU, m_maxU);

    // Aktualizacja bufor opóźnienia transportowego
    m_buforOpoznienia.push_back(sygnalSterujacy);
    double tymczasoweOpoznione = m_buforOpoznienia.front();
    m_buforOpoznienia.pop_front();

    // Aktualizacja bufor sterowania do splotu z B
    m_buforU.push_back(tymczasoweOpoznione);
    m_buforU.pop_front();

    // Obliczenie B
    double sumaB = 0.0;
    for (size_t i = 0; i < m_wspolczynnikB.size(); ++i)
        sumaB += m_wspolczynnikB[i] * m_buforU[m_wspolczynnikB.size() - 1 - i];

    // Obliczenie A
    double sumaA = 0.0;
    for (size_t i = 0; i < m_wspolczynnikA.size(); ++i)
        sumaA += m_wspolczynnikA[i] * m_buforY[m_wspolczynnikA.size() - 1 - i];

    // Zakłócenie (jeśli aktywne)
    double szum = (m_oSSzum > 0.0) ? m_rozkladZaklocen(m_GeneratorZaklocen) : 0.0;

    // Obliczenie wyniku
    double y = sumaB - sumaA + szum;

    // Ograniczenie regulowanej wartości (jeśli aktywne)
    if (m_ogrRegulowania)
        y = nasycenie(y, m_minY, m_maxY);

    // Aktualizacja bufora regulowanej
    m_buforY.push_back(y);
    m_buforY.pop_front();

    return y;
}
```

```cpp
double Model_ARX::symuluj(double sygnalSterujacy)
{
    // Ograniczenie sterowania (jeśli aktywne)
    if (m_ogrSterowania)
        sygnalSterujacy = nasycenie(sygnalSterujacy, m_minU, m_maxU);

    // Aktualizacja bufor opóźnienia transportowego
    m_buforOpoznienia.push_back(sygnalSterujacy);
    double tymczasoweOpoznione = m_buforOpoznienia.front();
    m_buforOpoznienia.pop_front();

    // Aktualizacja bufor sterowania do splotu z B
    m_buforU.push_back(tymczasoweOpoznione);
    m_buforU.pop_front();

    // Obliczenie B
    double sumaB = 0.0;
    for (size_t i = 0; i < m_wspolczynnikB.size(); ++i)
        sumaB += m_wspolczynnikB[i] * m_buforU[m_wspolczynnikB.size() - 1 - i];

    // Obliczenie A
    double sumaA = 0.0;
    for (size_t i = 0; i < m_wspolczynnikA.size(); ++i)
        sumaA += m_wspolczynnikA[i] * m_buforY[m_wspolczynnikA.size() - 1 - i];

    // Zakłócenie (jeśli aktywne)
    double szum = (m_oSSzum > 0.0) ? m_rozkladZaklocen(m_GeneratorZaklocen) : 0.0;

    // Obliczenie wyniku
    double y = sumaB - sumaA + szum;

    // Ograniczenie regulowanej wartości (jeśli aktywne)
    if (m_ogrRegulowania)
        y = nasycenie(y, m_minY, m_maxY);

    // Aktualizacja bufora regulowanej
    m_buforY.push_back(y);
    m_buforY.pop_front();

    return y;
}
```

Konsola debugowania progra

```
Model_ARX (-0.4 | 0.6 | 1 | 0 ) -> test zerowego pobudzenia: OK!
Model_ARX (-0.4 | 0.6 | 1 | 0 ) -> test skoku jednostkowego nr 1: OK!
Model_ARX (-0.4 | 0.6 | 2 | 0 ) -> test skoku jednostkowego nr 2: OK!
Model_ARX (-0.4, 0.2 | 0.6, 0.3 | 2 | 0 ) -> test skoku jednostkowego nr 3: OK!
```

```cpp
double Regulator_PID::symuluj(double uchyb)
{

    double czP = m_kp * uchyb;

    double czI = 0.0;
    if (m_ti > 0.0)
    {
        m_sumUchybow += uchyb;
        m_sumCalkiWew += (uchyb / m_ti);

        if (m_liczCalk == LiczCalk::Zew)
            czI = (1.0 / m_ti) * m_sumUchybow;
        else
            czI = m_sumCalkiWew;
    }

    double czD = 0.0;
    if (m_td > 0.0)
        czD = m_td * (uchyb - m_uchybPoprzedni);

    m_uchybPoprzedni = uchyb;

    return ograniczDoZakresu(czP + czI + czD, m_ogrMin, m_ogrMax);
}
```

Konsola debugowania progra

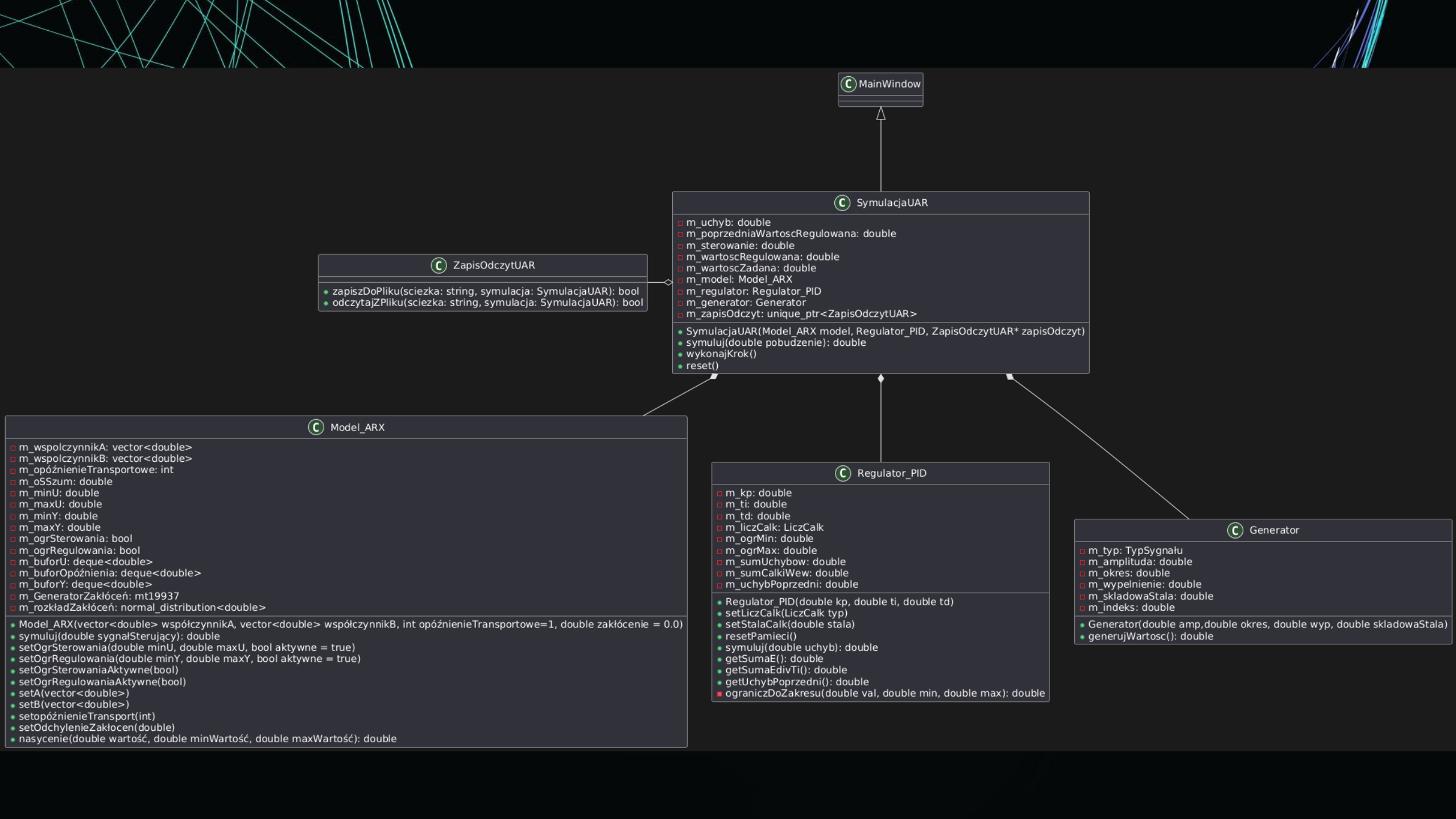
```
RegP (k = 0.5) -> test zerowego pobudzenia: OK!
RegP (k = 0.5) -> test skoku jednostkowego: OK!
RegPI (k = 0.5, TI = 1.0) -> test skoku jednostkowego nr 1: OK!
RegPI (k = 0.5, TI = 10.0) -> test skoku jednostkowego nr 2: OK!
RegPID (k = 0.5, TI = 10.0, TD = 0.2) -> test skoku jednostkowego: OK!
RegPI (k = 0.5, TI = 10.0 -> 5.0 -> 10.0) -> test skoku jednostkowego nr 3: OK!

C:\Users\FreddekMW\Documents\GitHub\PK_Symulator_UAR_poprawka\BACKEND\x64\Debug\BA
kodem 0 (0x0).
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzi
znie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Konsola debugowania progra

```
Testy zakonczone powodzeniem: 24/24

C:\Users\szkol\Documents\GitHub\PK_Symulator_UAR_poprawka\BACKEND\x64\Debug\BACKEND.exe (proces 13140) zakończono z kode
m 0 (0x0).
Naciśnij dowolny klawisz, aby zamknąć to okno...
```


MainWindow
SymulacjaUAR
- m_uchyb: double
- m_poprzedniaWartoscRegulowana: double
- m_sterowanie: double
- m_wartoscRegulowana: double
- m_wartoscZadana: double
- m_model: Model_ARX
- m_regulator: Regulator_PID
- m_generator: Generator
- m_zapisOdczyt: unique_ptr<ZapisOdczytUAR>
+ SymulacjaUAR(Model_ARX model, Regulator_PID, ZapisOdczytUAR* zapisOdczyt)
+ symuluj(double pobudzenie): double
+ wykonajKrok()
+ reset()
ZapisOdczytUAR
+ zapiszDoPliku(sciezka: string, symulacja: SymulacjaUAR): bool
+ odczytajZPliku(sciezka: string, symulacja: SymulacjaUAR): bool
Model_ARX
- m_wspolczynnikA: vector<double>
- m_wspolczynnikB: vector<double>
- m_opóźnienieTransportowe: int
- m_oSSzum: double
- m_minU: double
- m_maxU: double
- m_minY: double
- m_maxY: double
- m_ogrSterowania: bool
- m_ogrRegulowania: bool
- m_buforU: deque<double>
- m_buforOpóźnienia: deque<double>
- m_buforY: deque<double>
- m_GeneratorZakłóceń: mt19937
- m_rozkładZakłóceń: normal_distribution<double>
+ Model_ARX(vector<double> współczynnikA, vector<double> współczynnikB, int opóźnienieTransportowe=1, double zakłócenie = 0.0)
+ symuluj(double sygnałSterujący): double
+ setOgrSterowania(double minU, double maxU, bool aktywne = true)
+ setOgrRegulowania(double minY, double maxY, bool aktywne = true)
+ setOgrSterowaniaAktywne(bool)
+ setOgrRegulowaniaAktywne(bool)
+ setA(vector<double>)
+ setB(vector<double>)
+ setopóźnienieTransport(int)
+ setOdchylenieZakłocen(double)
+ nasycenie(double wartość, double minWartość, double maxWartość): double
Regulator_PID
- m_kp: double
- m_ti: double
- m_td: double
- m_liczCalk: LiczCalk
- m_ogrMin: double
- m_ogrMax: double
- m_sumUchybow: double
- m_sumCalkiWew: double
- m_uchybPoprzedni: double
+ Regulator_PID(double kp, double ti, double td)
+ setLiczCalk(LiczCalk typ)
+ setStalaCalk(double stala)
+ resetPamieci()
+ symuluj(double uchyb): double
+ getSumaE(): double
+ getSumaEdivTi(): double
+ getUchybPoprzedni(): double
- ograniczDoZakresu(double val, double min, double max): double
Generator
- m_typ: TypSygnału
- m_amplituda: double
- m_okres: double
- m_wypelnienie: double
- m_skladowaStala: double
- m_indeks: double
+ Generator(double amp,double okres, double wyp, double skladowaStala)
+ generujWartosc(): double

Podział na warstwy