**Spis treści** • *Contents*

Wstęp – Biuletyn

Drodzy Czytelnicy,

z wielką przyjemnością oddajemy w Państwa ręce kolejny numer czasopisma *Informatyka* ukazującego się w ramach Biuletynu Naukowego Wrocławskiej Wyższej Szkoły Informatyki Stosowanej.

Nie ma dziś żadnego prężniej rozwijającego się obszaru nauki od informatyki. Dynamiczny przyrost wiedzy na temat tak skomplikowanych zagadnień jak choćby sztuczna inteligencja to wielkie wyzwanie. Nie tylko dla świata, który przez to ciągle ewoluuje na naszych oczach, w efekcie czego nie każdy jest w stanie nadążyć za przemianami, ale i dla nas, badaczy, którzy biorą aktywny udział w tworzeniu nauki, a jednocześnie chcą przystępnie opowiadać o swoich odkryciach.

Pisanie artykułów naukowych z dziedziny informatyki nie jest rzeczą łatwą, ale daje wiele satysfakcji. Przedstawiamy Państwu wybór publikacji autorstwa naszych pracowników i studentów, które uznaliśmy za najlepsze.

W aktualnym numerze przybliżamy m. in. temat instrukcji warunkowych – rzecz wyjątkowo użyteczną, przydatną właściwie każdemu programiście niezależnie od tego, z jakiego języka korzysta, C++, Pythona, Javy czy jeszcze innych.

Nierzadko koderzy poświęcają większość swojego czasu nie na pisanie samych programów, a na tworzenie komentarzy do nich. Stąd prezentujemy też pracę omawiającą dobre praktyki w tym zakresie.

Osobom zainteresowanym programowaniem obiektowym proponujemy także lekturę artykułu poświęconego dziedziczeniu implementacyjnemu i delegacji, co wykorzystuje się właściwie w każdym większym projekcie czy kodzie.

Miłośników sztucznej inteligencji ucieszy pewnie tekst poświęcony sieci neuronowej, w szczególności sieciom Kohonena. Możliwości ich zastosowania są nadzwyczaj szerokie i często niespodziewane – kto by przypuszczał, że okażą się cenną pomocą w tworzeniu syntezatorów mowy albo w dziedzinie filozofii języka?

Dla chętnych do poszerzenia swojej wiedzy o algorytmice mamy też artykuł o Losowym Wczesnym Wykrywaniu RED wykorzystywanym m. in. w inżynierii biomedycznej, użytecznym zwłaszcza w fizykoterapii.

Czytelnikom z biznesowym zacięciem oferujemy także analizę funkcjonowania rynku NewConnect, chętnie wybieranego przez spółki liczące na szybkie pozyskanie kapitału bez martwienia się o spełnienie surowych wymagań głównego rynku Giełdy Papierów Wartościowych w Warszawie.

Z życzeniami przyjemnej lektury

# Algorytmy kolejkowania. Algorytm RED (Random Early Detection)

*Queueing algorithms. Algorithm RED (Random Early Detection)*

**Paweł Iljaszewicz**[1]

**STRESZCZENIE:** Artykuł omawia algorytm Losowego Wczesnego Wykrywania RED (*ang. Random Early Detection*) pozwalający bramce unikania przeciążeń w sieciach z komutacją pakietów. Brama wykrywa początkowe przeciążenie, obliczając średni rozmiar kolejki. Brama może powiadamiać o przeciążonych połączeniach lub o upuszczeniu pakietów przybywających do bramy, ustawiając bit w nagłówkach pakietów. Kiedy rozmiar średniej kolejki przekracza ustawiony próg, brama opada lub zaznacza każdy przybywający pakiet z pewnym prawdopodobieństwem, gdzie dokładny rozkład prawdopodobieństwa jest funkcją średniego rozmiaru kolejki. Bramki RED utrzymują średnią wielkość kolejki na niskim poziomie, jednocześnie zezwalając na sporadyczne impulsy pakietów w kolejce. Podczas przeciążenia prawdopodobieństwo, że brama powiadamia o konkretnym połączeniu, by zmniejszyć jego okno, jest mniej więcej proporcjonalne do udziału tego w przepustowości przez bramę. Bramki RED są zaprojektowane tak, aby dostarczyć protokół taki jak TCP, przeciążając warstwę transportową. Symulacje sieci TCP / IP są używane do zilustrowania wydajności bramki.

**SŁOWA KLUCZOWE:** Aktywne zarządzanie kolejkami (AQM), metody kolejkowania, algorytm RED, przepustowość sieci.

**ABSTRACT:** The subject of the study is to present the Random Early Detection (RED) algorithm that allows the gateway to avoid overloading in packet switched networks. The gateway detects the initial overload by calculating the average size of the queue. The gateway can notify about overload connections or by dropping packets arriving at the gate by setting a bit in the packet headers.
When the size of the average queue exceeds the set threshold, the gate descends or marks each arriving packet with a certain probability, where the exact probability distribution is a function of the average queue size.
RED gates maintain the average queue size at a low level, while allowing occasional packet bursts in the queue. During overload, probability that the gateway informs about a specific connection to reduce its window is more or less proportional to this connection involved in bandwidth through the gate. The RED gateways are designed to provide a protocol such as TCP to overload the transport layer. TCP / IP network simulations are used to illustrate the performance of the gateway.

**KEYWORDS:** Active queue management (AQM), queuing methods, RED algorithm, network bandwidth.

## 1. Wprowadzenie

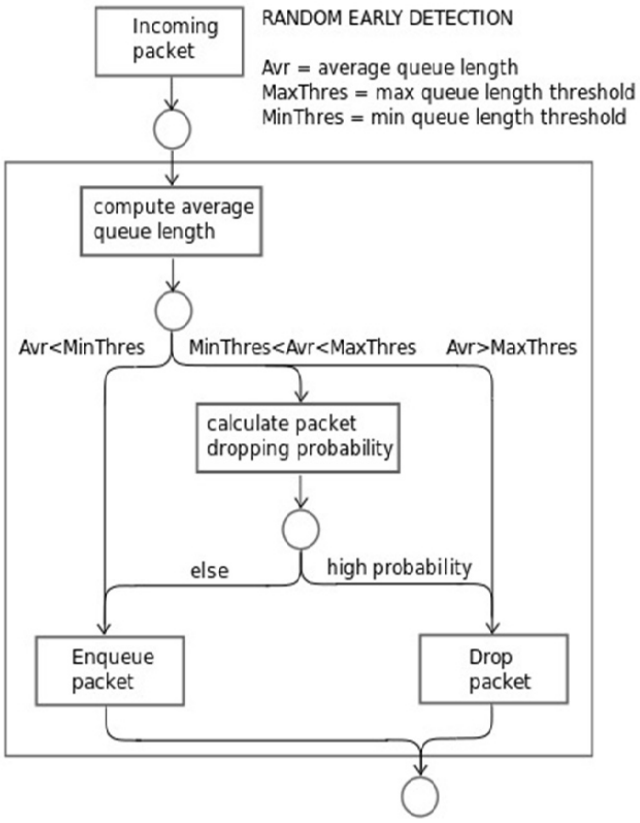
W przeciwieństwie do sieci domowych, problem przepustowości dla szybkobieżnych sieci z połączeniami powyżej 100 megabitów wymaga odpowiedniego projektowania. Aby uniknąć przejściowego przeciążenia, kolejkuje się przesyłanie danych, co powoduje wysyłanie ich z dużym opóźnieniem. W internecie protokół transportowy TCP wykrywa przeciążenie tylko po pakiecie przychodzącym do bramy. Taka implementacja dużych kolejek jest niepożądana, gdyż duże kolejki ( opóźnienie wynikające z przepustowości produktu) powodują znaczne zwiększenie średniego opóźnienia sieci. Dlatego przy coraz szybszych sieciach coraz ważniejsze jest posiadanie mechanizmów, które utrzymują przepustowość na wysokim poziomie, przy zachowaniu średniej wielkości kolejki na niskim. [1, 2]
Standardowym działaniem jest odrzucanie nadmiaru (*ang. Tail-drop*). Polega ono na przyjmowaniu określonej liczby pakietów do ustalonej (zadanej) wielkości, a wszystkie pozostałe są przekierowane na inną linię lub czekają na retransmisję, czyli powtórne wysłanie. Taka sytuacja po-

1. Instytut Technologiczno-Przyrodniczy, Falenty, al. Hrabska 3, 05-090 Raszyn. Email: p.iljaszewicz@itp.edu.pl

legająca na odrzuceniu serii pakietów, powoduje kolejne zapychanie się bramy. Aby temu zapobiec, tworzy się kolejki, co prawda powoduje to zwiększenie przepustowości, ale zwiększają się opóźnienia w wysyłaniu pakietów.
Jednym z rozwiązań jest zastosowanie algorytmu RED, (skrót od Random Early Detect), zwanego także Random Early Drop, co można przetłumaczyć na algorytm Losowego Wczesnego Wykrywania lub algorytm Losowego Wczesnego Odrzucania.[3, 4, 5]

## 2. Ogólny Algorytm RED

Na rysunku 1 przedstawiono blokowy schemat działania algorytmu RED.[6,7]
RED monitoruje średni rozmiar kolejki Avr (*ang.average queue length*) i przepuszcza (lub znakuje, gdy używane w połączeniu z ECN[2]) pakiety na podstawie statystycznych prawdopodobieństw. Jeśli bufor jest prawie pusty, wówczas wszystkie przychodzące pakiety są akceptowane. W miarę wzrostu kolejki rośnie również prawdopodobieństwo upuszczenia przychodzącego pakietu. Gdy bufor jest pełny, prawdopodobieństwo osiągnęło 1, a wszystkie przychodzące pakiety zostały odrzucone.
RED jest bardziej sprawiedliwy niż odrzucanie nadmiaru (*ang. Tail-drop*), w tym sensie, że nie ma tendencyjności przeciwko ruchowi seryjnemu (*ang. Bursty*), który wykorzystuje tylko niewielką część przepustowości. Im więcej host transmituje, tym bardziej prawdopodobne jest to, że jego pakiety są odrzucane, ponieważ prawdopodobieństwo, że pakiet hosta zostanie odrzucony, jest proporcjonalne do ilości danych, które ma w kolejce. Wczesne wykrywanie pomaga uniknąć globalnej synchronizacji TCP. MaxThres i MinThres oznacza odpowiednio maksymalny i minimalny próg długości kolejki pakietów. MinThres określa minimalny próg kolejki, od którego rozpocznie się odrzucanie. MaxThres to górna granica, której algorytm nie przekroczy, a seria to maksymalna liczba pakietów, która może zostać wysłana jednorazowo.
Ogólny algorytm RED bramki jest pokazany na rysunku 2.
Bramka RED oblicza średni rozmiar kolejki za pomocą filtra dolnoprzepustowego z wykładniczą ważoną średnią kroczącą. Średni rozmiar kolejki jest porównywany z dwoma progami: minimalnym i maksymalnym. Kiedy średni rozmiar kolejki jest mniejszy niż minimalny próg, żadne pakiety nie są oznaczone. Gdy średni rozmiar kolejki jest większy niż próg maksymalny, każdy pakujący pakiet jest oznaczony. Jeśli zaznaczone pakiety są w rzeczywistości zrzucone lub wszystkie węzły źródłowe są kooperatywne, zapewnia to nam, że średni rozmiar kolejki nie przekracza znacząco maksymalnego progu.
Gdy średni rozmiar kolejki jest w zakresie od minimalnego do maksymalnego progu, każdy przybywający pakiet jest oznaczony symbolem prawdopodobieństwa $p_a$, gdzie $p$ jest funkcją średniej kolejki $avg$.
Za każdym razem gdy jest zaznaczony pakiet, prawdopo




Ryc. 1 Schemat działania algorytmu RED[3]
*Fig. 1 RED principle of operation diagram*

dobieństwo, że pakiet jest oznaczony z określonego połączenia, jest proporcjonalne do udziału tego łącza w przepustowości na poziomie bramki.

```
for each packet arrival
   calculate the average queue size avg
   if min_th ≤ avg < max_th
      calculate probability p_a
      with probability p_a:
         mark the arriving packet
   else if max_th ≤ avg
      mark the arriving packet
```

Ryc. 2 Schemat algorytmu RED
*Fig. 2 Pseudocode of RED algorithm*

Zatem bramka RED ma dwa oddzielne algorytmy. Algorytm obliczania średniej wielkości kolejki określa stopień wielkości serii, który będzie dozwolony w kolejce bramy. Algorytm do obliczania prawdopodobieństwa oznaczania pakietów określa częstotliwość podawania znaczników bramek dla bieżącego poziomu przeciążenia. Celem jest brama znakująca paczki w dość równomiernie rozmieszczonych odstępach, aby uniknąć uprzedzenia i unikanie globalnej synchronizacji oraz zaznaczanie pakietów wystarczająco często, aby kontrolować średni rozmiar kolejki.

2. Explicit Congestion Notification Jawne Powiadomienie o Zatorach rozszerzenie protokołu TCP
3. http://www.icir.org/floyd/red.html

## 3. Szczegółowy algorytm dla bramek RED

Na rysunku 3 przedstawiono dokładny algorytm dla bramek RED [5]. Widzimy, że algorytm bramki RED można efektywnie wdrożyć z niewielką liczbą instrukcji dodawania i zmian dla każdego pakietu przychodzącego. Ponadto algorytm bramki RED nie jest ściśle sprzężony z przekazywaniem pakietów, a jego obliczenia nie muszą być wykonywane w krytycznej czasowo ścieżce przekazywania pakietów. Znaczna część pracy algorytmu bramki RED taka jak obliczenie średniego rozmiaru kolejki i prawdopodobieństwa oznaczania pakietów $p_b$ może być wykonywane równolegle z przekazywaniem pakietów lub może być obliczona przez bramę jako zadanie o niższej liczbie priorytetów, jeśli pozwala na to czas. Oznacza to, że algorytm bramki RED nie musi zakłócać działania bramy i umiejętności przetwarzania pakietów, a algorytm bramki RED może być dostosowany do coraz szybszego wyjścia pakietów.

```
Initialization:
    avg ← 0
    count ← −1
for each packet arrival
    calculate the new average queue size avg:
        if the queue is nonempty
            avg ← (1 − w_q)avg + w_q q
        else
            m ← f(time − q_time)
            avg ← (1 − w_q)^m avg
    if min_th ≤ avg < max_th
        increment count
        calculate probability p_a:
            p_b ← max_p(avg − min_th)/(max_th − min_th)
            p_a ← p_b/(1 − count · p_b)
        with probability p_a:
            mark the arriving packet
            count ← 0
    else if max_th ≤ avg
        mark the arriving packet
        count ← 0
    else count ← −1
when queue becomes empty
    q_time ← time

Saved Variables:
    avg:   average queue size
    q_time:   start of the queue idle time
    count:   packets since last marked packet

Fixed parameters:
    w_q:   queue weight
    min_th:   minimum threshold for queue
    max_th:   maximum threshold for queue
    max_p:   maximum value for p_b

Other:
    p_a:   current packet-marking probability
    q:   current queue size
    time:   current time
    f(t):   a linear function of the time t
```

Ryc. 3 Dokładny algorytm dla bramek RED

Jeśli metoda znakowania pakietów przez bramę RED ma ustawić bit wskazania przeciążenia w nagłówku pakietu, zamiast upuszczania nadchodzącego pakietu, ustawienie samego wskaźnika sygnalizującego zator pakietów dodaje do niego znacznik algorytmu bramy. Jednakże, ponieważ bramki RED są zaprojektowane do oznaczania jak najmniejszej liczby pakietów, narzut ustawienia bitów sygnalizacji przeciążenia jest ograniczony do minimum.
W przeciwieństwie do bram DECbit[4], który ustawia bit wskazania przeciążenia w każdym pakiecie, który dociera do bramy, gdy średni rozmiar kolejki przekracza próg. Dla każdego przybycia pakietu do kolejki bramy, bramka RED oblicza średni rozmiar kolejki. To można przedstawić w następujący sposób:

$$avg \leftarrow avg + wq(q - avg)$$

Tak długo jak wq (*ang. queue weight*) kolejka zadań jest wybierana jako (ujemna) potęga dwóch, można to zrealizować za pomocą jednej zmiany i dwóch dodatków (ze skalowanymi wersjami parametrów) [7].
Ponieważ bramka RED oblicza średni rozmiar kolejki przy odbiorze pakietu, a nie w ustalonym czasie interwału, obliczanie średniego rozmiaru kolejki jest modyfikowane, gdy pakiet dociera do bramy do pustej kolejki. Gdy pakiet dotrze do bramy do pustej kolejki, bramka oblicza m liczbę pakietów, które mogły zostać przesłane przez bramę w czasie, kiedy linia była wolna.
Bramka oblicza średni rozmiar kolejki tak, jakby *m* przybyło do bramy z rozmiarem kolejki zero. Obliczenia są następujące:

$$(1-wq)^{(time-q\ time)/s}$$

$$avg \leftarrow (1-wq)^m avg$$

Gdzie *q*_time jest początkiem czasu bezczynności kolejki, a *s* jest typowym czasem transmisji małego pakietu. To całe obliczenie jest przybliżeniem, ponieważ jest oparte na liczbie pakietów, które mogły dotrzeć do bramy przez pewien czas. Po obliczeniu czasu bezczynności ($time - q\_time$) ma przybliżoną wartość poziomu dokładności, można użyć wyszukiwania tabeli, aby uzyskać warunek, $(1-wq)^{(time-q\ time)/s}$, który powinien w przybliżeniu być potęgą liczby 2.

## 4. Implementacja algorytmu RED

Gdy pakiet dotrze do bramy, a średni rozmiar kolejki *avg* przekracza próg maksymalny $max_{th}$, to taki przybywający pakiet jest oznaczany. Nie ma ponownego przeliczania prawdopodobieństwa oznaczania pakietów. Jednakże, gdy pakiet a dociera do bramy, a średnia wielkość kolejki *avg* między dwoma progami to $min_{th}$ i $max_{th}$, prawdopodobieństwo $p_b$ oznaczenia początkowego pakietu oblicza się w następujący sposób:

$$p_b \leftarrow C_1 avg - C_2$$

4. DECbit- Mechanizm decyzyjny ograniczania ruchu - został opracowany do użytku w architekturze cyfrowej sieci (Digital Network Architecture DNA), sieci bezpołączeniowej zawierającej protokół transportowy.

Dla:

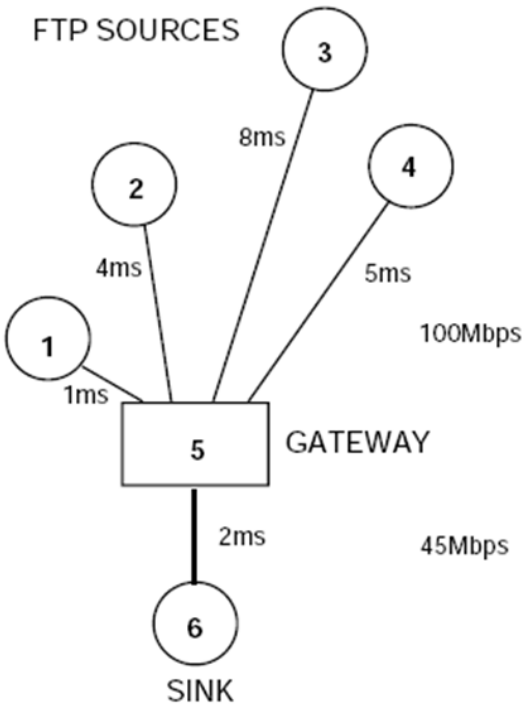
$$C_1 = \frac{max_p}{max_{th} - min_{th}}$$

$$C_2 = \frac{max_p min_{th}}{max_{th} - min_{th}}$$

Parametry $max_p$, $max_{th}$ i $min_{th}$ są ustalonymi parametrami określonymi wcześniej. Wartości dla $max_{th}$ i $min_{th}$ są określane przez pożądane ograniczenia średniego rozmiaru kolejki i mogą być ograniczone w sposób elastyczny. Jednak ustalony parametr $max_p$ można łatwo ustawić na zakres wartości. W szczególności $max_p$ można wybrać tak, aby $C_1$ był potęgą dwóch. W ten sposób obliczenie $p_b$ można osiągnąć za pomocą jednej zmiennej z jedną instrukcją dodawania.

## 5. Obliczenia i analiza wyników

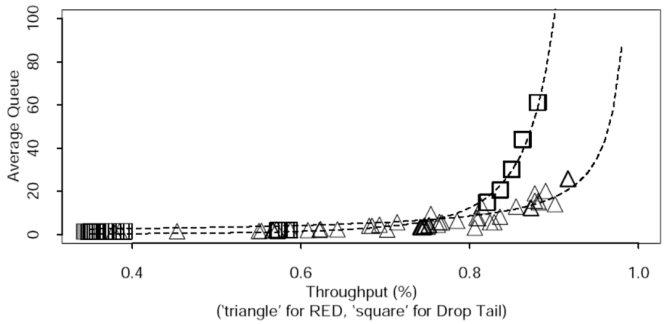
W pracach [8, 12] przedstawiono badanie symulacyjne algorytmu RED. Na rysunku 4 widzimy schemat symulowanej sieci i rozmieszczenie węzłów.




Ryc. 4 Schemat symulowanej sieci
*Fig. 4 Diagram of simulated network*

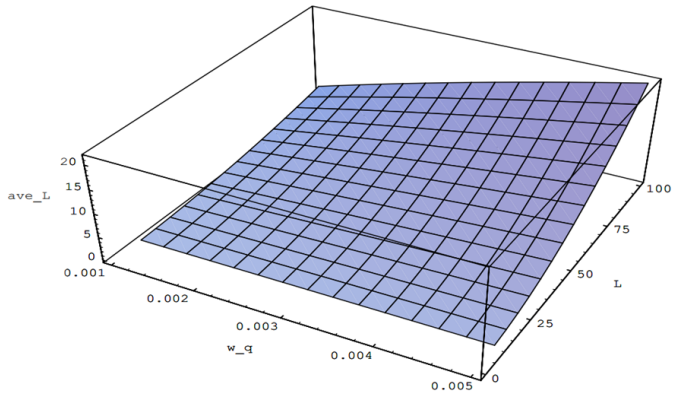
Na rysunku 5 porównano przepustowość pakietów na bramkach Drop Tail i RED.
Symulacje z bramkami RED były uruchamiane z rozmiarem bufora 100 pakietów, z dziewiątym zakresem od 3 do 50 pakietów. Dla bramek RED wartość $max_{th}$ jest ustawiona na $3min_{th}$, z $wq = 0{,}002$ i $max_p = 1/50$. Linie przerywane pokazują średnie opóźnienie (w funkcji przepustowości) przybliżone o $1{,}73/(1- x)$ dla symulacji z bramkami RED i przybliżone przez $0{,}1 = (1-x)^3$ dla symulacji bramy z Drop Tail.

Węzeł 1 zaczyna przesyłać w czasie 0,0 sek.
Węzeł 2 zaczyna przesyłać w czasie 0,2 sekundy
Węzeł 3 zaczyna przesyłać w czasie 0,4 s
Węzeł 4 zaczyna przesyłać w czasie 0,6 sekundy




Ryc. 5 Porównanie bram Drop Tail (kwadraty) i RED (trójkąty)
*Fig. 5 Comparison of gateways Drop Tail (square) and RED (triangle)*

Autorzy [12, 13] przypisują ulepszoną przepustowość do zredukowanej średniej długości kolejki. Pakiety oczekują mniej czasu w kolejce i szybciej się rozłączają.
Można porównać to do szybkiej jazdy samochodem. Wyprzedzanie większej liczby samochodów nie oznacza, że większa ilość pojazdów na jednostkę czasu dotrze do celu.




Ryc. 6 Graficzne przedstawienie zależności pomiędzy *avg* a *wq*
*Fig 6. Graphical representation of relation between avg and wq*

Jeśli kolejka zadań *wq* jest zbyt długa, wówczas procedura uśredniania nie odfiltruje chwilowego przeciążenia w bramie. Załóżmy, że kolejka jest początkowo pusta, ze średnim rozmiarem kolejki równym zero, a następnie kolejka wzrasta od 0 do *L* pakietów przez *L* pakietów przybycia. Po przybyciu pakietu *L* - tego do bramy, średnia kolejka ma rozmiar $avg_L$ dany wzorem[8]:

$$avg_L = L + 1 + \frac{(1-wq)^{L+1} - 1}{wq}$$

Na rysunku 6 pokazano zależność pomiędzy $avg_L$, czyli średnim rozmiarem kolejki a ilością pakietów *L* (oś y od 0 do 100 pakietów) i wq (*ang. queue weight*) kolejką zadań (oś x od 0,001 do 0,005). Przykładowo dla wartości *wq* = 0,001, po zwiększeniu pakietów *L* od 0 do 100 średnia wielkość pakietu $avg_{100}$ wyniesie 4,88 pakietu.

## 6. Podsumowanie

Algorytm RED nie jest lekarstwem na wszystko, aplikacje, które niewłaściwie realizują wykładniczy rozkład, wciąż mają nieuczciwy udział w przepustowości, jednak dzięki RED nie powodują one tak dużej szkody dla przepustowości i opóźnień innych połączeń.
RED statystycznie upuszcza pakiety z przepływów, zanim osiągnie swój twardy limit. Powoduje to, że zatłoczone łącze szkieletu zwolni bardziej płynnie i zapobiega też synchronizacji retransmisji. Pomaga to również w szybszym odczytywaniu przez TCP „uczciwej" prędkości, pozwalając na odrzucenie niektórych pakietów wcześniej, utrzymując niskie rozmiary kolejek i kontrolując opóźnienia.
Prawdopodobieństwo, że pakiet zostanie usunięty z określonego połączenia, jest proporcjonalne do jego wykorzystania przepustowości, a nie do liczby pakietów, które przesyła.
RED to dobra opcja dla rozbudowy routerów w konfiguracji szkieletowej, w której nie można sobie pozwolić na złożoność śledzenia stanu w sesji potrzebnego dla odpowiedniego kolejkowania.
Algorytm znalazł zastosowanie przy przesyłaniu dużych pakietów danych podczas analizy termograficznej w inżynierii biomedycznej. Szczególnie w fizykoterapii gdzie termowizja wykorzystywana jest do oceny skuteczności wykonywanych zabiegów do śledzenia i oceny zabiegów seryjnych [12].
Czysty algorytm RED nie uwzględnia zróżnicowania jakości usług (*QoS ang. quality of service*).
Istnieją odmiany algorytmu poprawiające jego działanie [5, 11, 13, 14]:
Weighted random early detection (WRED) zapewnia wczesne wykrywanie z uwzględnieniem QoS, Robust random early detection (RRED) zapewnia zwiększenie przepustowości TCP przeciwko atakom Denial-of-Service (DoS), w szczególności atakom niskopoziomowym typu Denial-of-Service.

## Literatura

[1] D. Agrawal, N.L.S. da Fonseca and F. Granelli, "Integrated ARM/AQM mechanisms based on PID controllers," Proc. ICC, pp. 6-10, May 2005.
[2] S. Athuraliya, S.H. Low, V.H. Li and Q. Yin, "REM: Active queue management," IEEE Network Mag., vol. 15, no. 3, pp. 48-53, 2001
[3] B. Braden, et al., "Recommendations on queue management and congestion avoidance in the Internet," IETF RFC2309, 1998
[4] G. Chen and T. T. Pham, "Introduction to fuzzy systems," Chapman & Hall/CRC, 2006.
[5] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith. "Tuning RED for web traffic," IEEE/ACM Trans. Networking, vol. 9, No. 3, pp. 249-264, 2001.
[6] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A self-configuring RED gateway," Proc. INFOCOM, pp. 1320-1328, Mar. 1999.
[7] S. Floyd, R. Gummadi and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management," available at http://www.icir.org/floyd/red.html
[8] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," IEEE/ACM Trans. Networking, vol. 1, no. 4, pp. 397-413, Aug. 1993.
[9] C. V. Hollot, V. Misra, D. Towsley and W. Gong, "On designing improved controllers for AQM routers supporting TCP flows," Proc. IEEE INFOCOM 2001, vol. 3, Anchorage, Alaska, pp. 1726-1734, April 2001.
[10] L. Hu and A. D. Kshemkalyani, "HRED: A simple and efficient active queue management algorithm," Proc. ICCCN 2004, pp. 387-393, 2004.
[11] C. Joo, S. Bahk and S. S. Lumetta, "Hybrid active queue management," Proc. ISCC'03, pp. 999-1004, 2003.
[12] P. Kardasz i inni Termografia w inżynierii biomedycznej Laboratorium-Przegląd Ogólnopolski, 31-38
[13] J. Koo, K. Chung, H. Kim and H. Lee, "A new active RED algorithm for congestion control in IP networks" LNCS 2343, pp. 469-479, 2002.
[14] S. S. Kunniyur and R. Srikant, "An adaptive virtual queue (AVQ) algorithm for active queue management," IEEE/ACM Transactions on Networking, vol. 12, no. 2, pp. 286-299, April 2004.

# Dziedziczenie implementacyjne, delegacja

*Implementation inheritance, delegation*

**Beata Skiba[1], Amadeusz Stasiak[1], Alicja Żyła[1]**

**STRESZCZENIE:** Celem tego artykułu jest przedstawienie dziedziczenia implementacyjnego oraz delegacji. Zostanie przedstawione jak działają obie techniki, a także jakie mają zastosowania, ograniczenia oraz problemy. Praca ma na celu przedstawienie również klasy dziedziczenia implementacyjnego oraz jego rodzajów elementów takich jak elementy prywatne, chronione i publiczne. Ograniczeniami dziedziczenia implementacyjnego są pojedynczość i niezmienność. Dziedziczenie implementacyjne posiada również swoje moduły programowe oraz topologię sieci. Delegacja ma własne projektowanie struktur danych oraz jest bardzo dobrą alternatywą dla dziedziczenia implementacyjnego. W artykule przedstawione jest jak jest tworzona delegacja oraz jakie posiada cechy. Podczas wykorzystywania delegacji pomocne jest programowanie zdarzeniowe. Zaprezentowane jest także działanie programów na starszych systemach przy użyciu deklaracji. Zarówno dziedziczenie implementacyjne jak i deklaracja są ważną częścią programowania obiektowego. Przedstawione są bardzo ważne różnice między dziedziczeniem implementacyjnym a delegacją. Na końcu referatu pokazane są spostrzeżenia odnośnie do dziedziczenia implementacyjnego oraz delegacji.

**Słowa kluczowe:** Dziedziczenie implementacyjne, delegacja, programowanie sterowane zdarzeniami.

**ABSTRACT:** The aim of this paper is to present the implementation inheritance and delegation. Will be presented how both techniques work, and what are their applications, limitations and problems. The article aims to present the implementation inheritance class and its types of elements such as private, protected and public elements. The constraints of implementation inheritance are singularity and immutability. Implementation inheritance has many limitations, such as composition, admixture, interface and features. Implementation inheritance also has its program modules and a network topology. The delegation has its own design of data structures and is a very good alternative to implementation inheritance. This paper outlines how a delegation is created and what features it has. Event programming is helpful when using delegations. The operation of programs on older systems using the declaration is also presented. Both implementation, inheritance and declaration are an important part of object-oriented programming. Very important differences between implementation inheritance and delegation are also presented. At the end of the paper, the observations about implementation inheritance and delegation are shown.

**Keywords:** Implementation inheritance, Delegation, Event-driven programming.

## 1. WSTĘP

Dziedziczenie implementacyjne i delegacja są jednymi z wielu elementów programowania obiektowego. Jak każde, mają swoje określone funkcje, które są niezbędne do napisania niektórych programów czy aplikacji. Ułatwiają nam swoją istotą pisanie kodu, który staje się krótszy o wiele linijek i zajmuje mniej pamięci operacyjnej na naszym sprzęcie. Niemalże każdy programista używa ich do swojego projektu na co dzień.

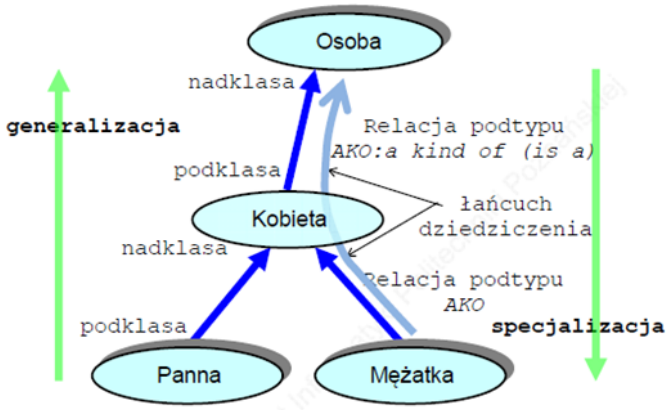
## 2. DZIEDZICZENIE IMPLEMENTACYJNE - definicja, zastosowanie, ograniczenia, rozwiązania ograniczeń

### 2.1 Dziedziczenie implementacyjne

Dziedziczenie implementacyjne jest to mechanizm współdzielenia funkcjonalności między klasami. Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych atrybutów oraz zachowań uzyskuje także te pochodzące z klasy, z której dziedziczy. Klasa dziedzicząca jest nazywana klasą pochodną lub potomną, zaś klasa, z której następuje dziedziczenie — klasą bazową.

1.Wrocławska Wyższa Szkoła Informatyki Stosowanej “Horyzont” Wejherowska 28, 54-239 Wrocław beata.skiba1994@gmail.com, stasiak.amadeusz@gmail.com, alicja.zyla96@gmail.com.

Z jednej klasy bazowej można uzyskać dowolną liczbę klas pochodnych. Klasy pochodne posiadają obok swoich własnych metod i pól również kompletny interfejs klasy bazowej. W językach programowania z prototypowaniem (np. JavaScript) nie występuje pojęcie klasy, dlatego dziedziczenie implementacyjne zachodzi tam pomiędzy poszczególnymi obiektami. Pojęcie dziedziczenia implementacyjnego zostało wprowadzone po raz pierwszy przez twórców języka Simula.[14]




Ryc. 1 Przykład dziedziczenia.
*Fig. 1 Example of inheritance*

## 2.2 Klasy oraz rodzaje dziedziczenia implementacyjnego

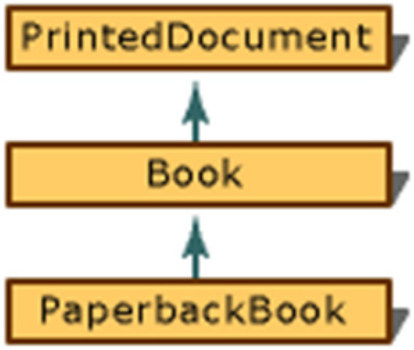
Zależności między klasami bazowymi i pochodnymi tworzą tak zwane hierarchie klas. Klasy pochodne otrzymują wszystkie metody i atrybuty swoich klas bazowych oraz mogą dodawać nowe. Dopuszczalne jest także nadpisywanie istniejących metod, przy czym poszczególne języki programowania mogą żądać spełnienia dodatkowych warunków, np. pozostawienia niezmienionej listy argumentów wejściowych i typu wyniku. Wiele języków programowania umożliwia deklarowanie klas jako abstrakcyjnych. Nie można tworzyć obiektu klasy abstrakcyjnej, lecz można po takiej klasie dziedziczyć. Klasa abstrakcyjna może zawierać metody czysto wirtualne, które muszą zostać zaimplementowane przez klasy pochodne. Mechanizmu tego używa się, jeśli twórca klasy chce dostarczyć jedynie części funkcjonalności, tworząc szkielet dla innych, bardziej wyspecjalizowanych klas. W części języków programowania istnieje możliwość ograniczania widoczności dziedziczonych pól i metod:

**Elementy publiczne** — nieograniczony dostęp, można je wywoływać zarówno z wnętrza klas, jak i spoza nich. Wadą ich jest to, że może dojść do przypadkowej zmiany ich wartości, co może zakłócić działanie reszty programu.

**Elementy chronione** — można je wywoływać jedynie z wnętrza klasy oraz z wnętrza wszystkich klas pochodnych. Najczęściej stosowane przy dziedziczeniu implementacyjnym w klasie bazowej.

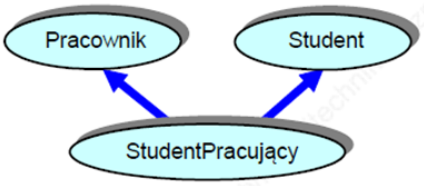
**Elementy prywatne** — można je wywoływać jedynie z wnętrza bieżącej klasy, natomiast nie ma do nich dostępu w klasach pochodnych. Najbezpieczniejsze zabezpieczenie danych ze względu na to, że można je zmienić tylko gotowymi funkcjami. Używane przy klasach pochodnych ostatniego rzędu.

W programowaniu obiektowym wyróżniane są dwa dziedziczenia implementacyjne. Jest to dziedziczenie pojedyncze oraz dziedziczenie wielokrotne. Z dziedziczeniem pojedynczym mamy do czynienia, gdy klasa pochodna dziedziczy po dokładnie jednej klasie bazowej (oczywiście klasa bazowa wciąż może dziedziczyć z jakiejś innej klasy), tak jak jest to pokazane na rysunku poniżej:




Ryc. 2 Dziedziczenie pojedyncze. [5]
*Fig. 2 Single inheritance*

Natomiast w dziedziczeniu wielokrotnym klas bazowych może być więcej. Wielokrotne dziedziczenie jest obsługiwane w takich językach, jak C++, Common Lisp czy Perl. Rysunek poniżej pokazuje nam dziedziczenie wielokrotne:

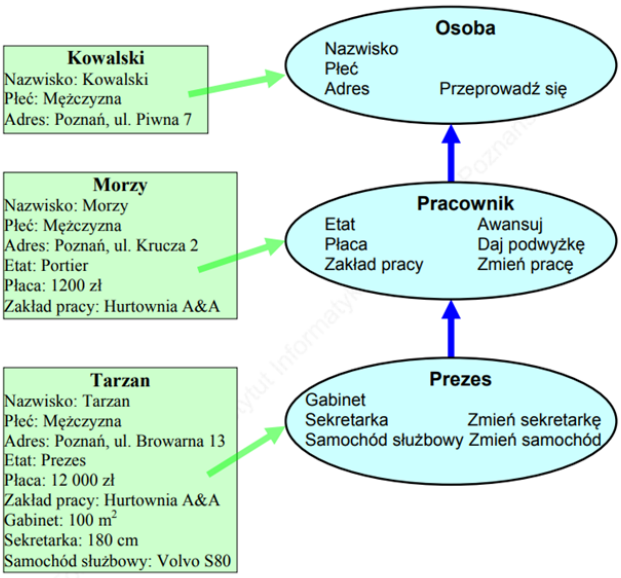



```
class Pracownik {
    ...
    float płaca;
    char *etat;
    ... };
class Student {
    ...
    char *uczelnia;
    int rokStudiów;
    float średniaOcen;
    ... };

class StudentPracujący : public Pracownik,
    public Student { ... };
```

Ryc. 3 Dziedziczenie wielokrotne w C++. [1]
*Fig. 3 Multiple inheritance in C++*

Zwiększa ono możliwości ponownego wykorzystania kodu, lecz jednocześnie jest krytykowane za:

**A. Niejednoznaczność semantyczną (tzw. Diamond problem),**
**B. Problemy z łańcuchowym wywoływaniem konstruktorów,**
**C. Trudności implementacyjne.**




Ryc. 4 Implementacja dziedziczenia implementacyjnego. [1]
*Fig. 4 Impementation of implementation inheritance*

Klasy abstrakcyjne są to klasy, które nie mają kompletnej implementacji. W związku z tym klasy te nie mogą mieć wystąpień. Mogą one służyć jedynie jako pośredni etap dla dziedziczących po nich klas zawierających implementację cech abstrakcyjnych. Zdefiniowanie danej klasy jako abstrakcyjnej uniemożliwia tworzenie wystąpień tej klasy. Własna implementacja klasy abstrakcyjnej wiąże się z występowaniem błędów w czasie wykonywania programów [1].
Powyższe problemy dotyczą przede wszystkim konfliktów implementacji. Dlatego nawet jeśli
w danym języku programowania wielokrotne dziedziczenie klas jest niedozwolone, można je stosować w przypadku interfejsów, które mogą być traktowane jak klasy abstrakcyjne zawierające wyłącznie metody czysto wirtualne [10][11].

### 2.3 Zastosowanie dziedziczenia implementacyjnego.

Podstawowym zastosowaniem dziedziczenia implementacyjnego jest ponowne wykorzystanie kodu. Jeśli dwie klasy wykonują podobne zadania, możemy utworzyć dla nich wspólną klasę bazową, do której przeniesiemy identyczne metody oraz atrybuty. Ułatwi to testowanie oraz potencjalnie zwiększy niezawodność aplikacji w przypadku zmian. W razie ewentualnych problemów łatwiej będzie również odnaleźć przyczynę błędu. Poniżej znajduje się przykład dziedziczenia implementacyjnego w języku C++:

```
class Osoba {
protected:
    char nazwisko [Max];
    char płeć;
    unsigned wiek;
public:
    Osoba (char*, char, unsigned);
    Void ZmieńNazwisko(char*);
    void DaneOsobowe( );
};

class Pracownik : public Osoba {
protected:
    char Etat [Max];        //nowa cecha
    unsigned płaca;         //nowa cecha
public:
    Pracownik (char*, char, unsigned, char*);
    void Awansuj(char*);   //nowa cecha
    void DajPodwyżkę(unsigned); //nowa cecha
    void DaneOsobowe( );   //redefinicja cechy
};
...
Osoba Morzy("Morzy",'M',45);
Morzy.DaneOsobowe();
Pracownik Buła("Buła",'M',38,"portier");
Buła.Awansuj("prezes");
Buła.ZmieńNazwisko("Tarzan");
Buła.DaneOsobowe();
```

klasa bazowa

Ryc. 5 Przykład dziedziczenia implementacyjnego w języku C++. [1]
*Fig. 5 Example of implementation inheritance in C++*

W języku C++ nie są dziedziczone:

**A. konstruktory i destruktory klasy**
**B. przeciążony operator=()**
**C. „przyjaciele" klasy [1]**

### 2.4 Hierarchia klas

Hierarchia klas może przekładać się na hierarchię typów. Możliwe jest wtedy podstawienie pod zmienną (lub atrybut funkcji) typu T obiektu typu S będącego podtypem T i dalsze używanie go jakby był typu T. Jest to możliwe dzięki temu, że podklasa posiada kompletny interfejs swojej nadklasy.
W podklasie może być zdefiniowana metoda już istniejąca w nadklasie. Konstrukcja taka umożliwia wykonywanie operacji na obiektach bez informacji, z jakim właściwie obiektem mamy do czynienia. Rozpatrzmy typową aplikację GUI wyświetlającą na ekranie różne komponenty (np. przycisk, pole tekstowe czy listę rozwijaną). Reagują one na te same zdarzenia: kliknięcie myszką, naciśnięcie klawisza, lecz każdy z nich reaguje inaczej, stosownie do tego, czym jest. System obsługi zdarzeń najpierw określa, który z komponentów powinien obsłużyć zdarzenie, a następnie przekazuje mu je. Pobierając aktywny obiekt, możemy wywołać tę metodę bez zastanawiania się czy dany obiekt jest przyciskiem, czy polem tekstowym. Decyzja o tym, która wersja zachowania zostanie wywołana
w konkretnym miejscu, zależy od języka programowania i sposobu zdefiniowania metod. Rozpatrzmy następującą sytuację:

```
class A {
    method foo();
}

class B extends A {
    method foo();
}

A obiektBazowy = new A();
B obiektPochodny = new B();
obiektBazowy.foo(); // 1

obiektBazowy = obiektPochodny;
obiektBazowy.foo(); // 2
```

Ryc. 6 Przykład hierarchii klas. [1]
*Fig. 6 Example of class hierarchy*

Mamy klasę bazową A oraz dziedziczącą z niej klasę B. Klasa bazowa definiuje metodę foo(), która jest nadpisywana przez klasę pochodną. Przypadek pierwszy nie budzi żadnych wątpliwości: mamy utworzony obiekt klasy A, dlatego wywołujemy wersję metody foo() zdefiniowaną w tej klasie. W przypadku drugim pod zmienną obiekt Bazowy podstawiony jest obiekt klasy pochodnej. Jednak wtedy w linijce oznaczonej przez 2 możemy wywołać wersję metody foo() z klasy A, mimo iż zmienna wskazuje na obiekt klasy pochodnej posiadającej zmodyfikowaną wersję tej metody lub wywołać wersję metody foo() z klasy pochodnej B, mimo iż zmienna obiektBazowy jest typu A.
Jeśli zachodzi sytuacja druga, powiemy, że metoda foo() jest metodą wirtualną. W niektórych językach (np. C++) metody muszą być jawnie deklarowane jako wirtualne przez programistę.
W innych (np. Java) wszystkie metody są z definicji wirtualne.W ogólnym ujęciu podtypowanie i dziedziczenie to dwa różne pojęcia. Dziedziczenie dotyczy powtórnego wykorzystania klasy bazowej, natomiast podtypowanie (polimorfizm) możliwości wykorzystania podtypu w miejscu nadtypu.

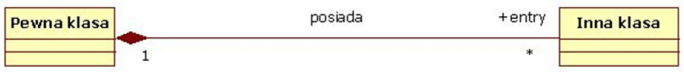
### 2.5 Ograniczenia dziedziczenia implementacyjnego

Dziedziczenie implementacyjne posiada kilka ograniczeń wynikających z faktu, że hierarchia klas jest ustalana w momencie kompilacji programu i nie może podlegać późniejszym zmianom. Wyobraźmy sobie klasę Osoba, z której dziedziczą klasy Pracownik oraz Student. Napotykamy tutaj na istotne problemy:

**Pojedynczość** — w językach z pojedynczym dziedziczeniem osoba może być albo pracownikiem, albo studentem.
**Niezmienność** — nawet jeśli skorzystamy z wielokrotnego dziedziczenia i utworzymy klasę StudentPracownik, klasę wybieramy w momencie tworzenia obiektu i nie możemy jej później zmienić. Oznacza to, że system nie może poprawnie reagować na sytuacje, gdy dotychczasowy student zostaje dodatkowo pracownikiem, gdyż konieczne jest wtedy utworzenie całkowicie nowego obiektu.
Innym istotnym ograniczeniem jest uzależnienie kodu od konkretnej implementacji klasy, które może doprowadzić do błędów przy jej zmianie. Tego typu problem pojawia się zwłaszcza gdy dziedziczymy między klasami znajdującymi się w różnych komponentach (tzw. Problem kruchości klasy podstawowej).

### 2.6 Rozwiązania ograniczeń dziedziczenia implementacyjnego

Istnieje kilka rozwiązań alternatywnych eliminujących poszczególne ograniczenia dziedziczenia implementacyjnego, są to kompozycja, domieszki oraz interfejsy i cechy. Przykładami rozwiązań ograniczeń dziedziczenia implementacyjnego są:
**Kompozycja** — polega na zastąpieniu dziedziczenia implementacyjnego składaniem mniejszych obiektów. Posługując się dalej powyższym przykładem, możemy zostawić klasę Osoba, która posiada listę ról takich, jak Pracownik czy Student. Obiektowi można przydzielać nowe oraz usuwać stare rolę w dowolnym momencie wykonania programu, eliminując tym samym problemy pojedynczości oraz niezmienności. Wadą kompozycji jest większe zużycie pamięci (dużo małych obiektów) oraz niewielki spadek wydajności podczas dostępu do metod. Kompozycję można stosować w każdym języku obsługującym programowanie obiektowe.




Ryc. 7 Przykład kompozycji w agregacji. [1]
*Fig. 7 Example of composition in aggregation*

**Domieszka** — pozwala uzyskać funkcjonalność podobną do wielokrotnego dziedziczenia, unikając jednocześnie trapiących je paradoksów. Jest to rodzaj klasy abstrakcyjnej, którą można „dodać" do właściwych klas. Klasa uzyskuje wszystkie atrybuty oraz metody zdefiniowane w dodanych do niej domieszkach. Oddzielenie klas od domieszek pozwala wprowadzić jasne reguły rozwiązywania konfliktów. Przykładem języka wykorzystującego domieszki jest Ruby.
**Interfejs** — to rodzaj klasy abstrakcyjnej, która może zawierać wyłącznie metody czysto wirtualne oraz stałe. Ponieważ paradoksy dotyczą wyłącznie implementacji, której tu nie ma, w interfejsach można bezpiecznie korzystać z wielokrotnego dziedziczenia. Również klasy mogą implementować więcej niż jeden interfejs jednocześnie. Przykładami języków wykorzystujących interfejsy są Java oraz C#. Język Java oprócz klas abstrakcyjnych obejmuje dodatkowo pojęcie interfejsu, który może być traktowany jako szczególna klasa abstrakcyjna, która w ogóle nie po-

siada implementacji. Definicja interfejsu nie może zawierać elementów prywatnych, definicji zmiennych obiektów oraz metod typu final. Interfejsy powinny być implementowane przez klasy. Pojedyncza klasa może implementować wiele interfejsów. Dany interfejs może reprezentować pojedynczy aspekt klasy. Mogą być one powiązane siecią dziedziczenia. [1]

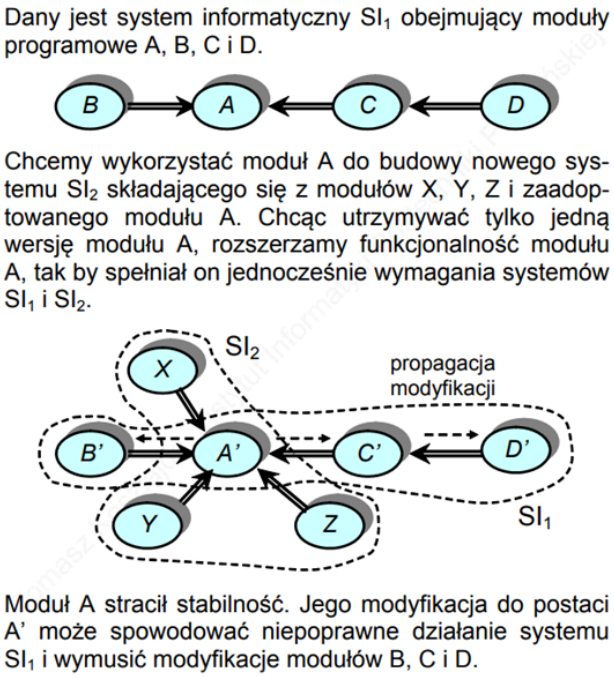
```
interface Interfejs{

  public void proc();

}

class Klasa implements Interfejs{

  @Override
  public void proc(){
    // ...
  }
}
```

Ryc. 8 Przykład interfejsu w Java. [1]
*Fig. 8 Java interface example*

**Cechy** — umożliwiają wielokrotne wykorzystanie tego samego kawałka kodu w różnych klasach.
W przeciwieństwie do domieszek kod ten zachowuje się tak, jakby był zapisany w tych klasach bezpośrednio, a w momencie wykonania programu nie ma możliwości stwierdzenia, czy dana metoda została zaimplementowana bezpośrednio w klasie, czy dodana przez cechę.

## 2.7 Moduły programowe

Moduły programowe powinny być jednocześnie otwarte i stabilne. Moduły programowe powinny być otwarte na dalszy rozwój: dodanie nowej funkcjonalności lub zmianę implementacji. Eksploatowane moduły powinny być stabilne. Modyfikowanie poprawnie działającego modułu może spowodować jego błędne działanie. Wymaganie otwartości modułów jest konsekwencją potrzeby rozwoju i utrzymania systemów informatycznych. Wymaganie stabilności modułów jest wynikiem potrzeby osiągnięcia stabilizacji eksploatowanych systemów informatycznych. Równoczesne spełnienie tych dwóch wymagań jest bardzo kłopotliwe.
Konsekwencją braku stabilności modułu jest współdzielenie kodu przez modyfikację.
Przykład możemy zaobserwować poniżej. [1]




Ryc. 9 Przykład braku stabilności. [1]
*Fig. 9 Example of no stability*

## 2.8 Topologia sieci dziedziczenia implementacyjnego

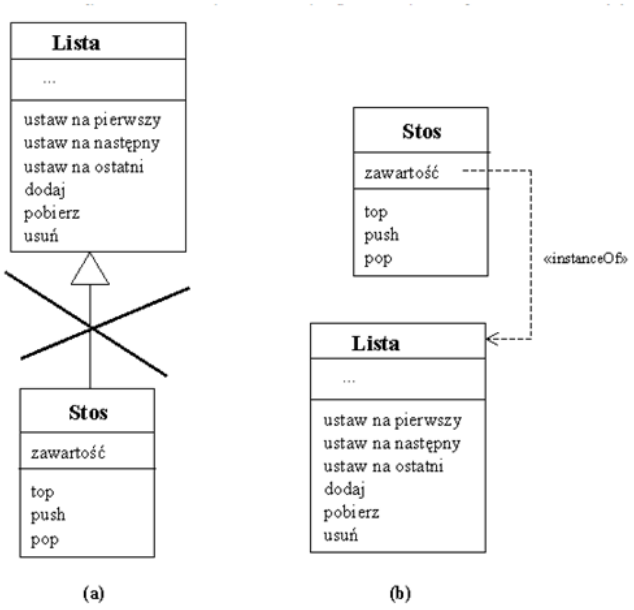
Dziedziczenie jednokrotne (SmallTalk 80, C#, klasy w języku Java) - każda klasa ma co najwyżej jedną nadklasę. Sieć klas ma kształt hierarchii. Dziedziczenie wielokrotne (C++, Eiffel, interfejsy
w języku Java) - klasy mogą dziedziczyć po wielu nadklasach. Sieć klas ma kształt grafu acyklicznego skierowanego. Sieć klas może być rozłączna (C++) lub być niepodzielna i mieć wyróżniony wierzchołek, który jest korzeniem sieci (SmallTalk 80, Java, C#). [1]

```
class Osoba {
protected:
    char *nazwisko;
    ...
public:
    void Nazwisko(char*);
    void wiek(int);
    ...
};
class Pracownik : public Osoba{ ... };
class Student : public Osoba { ... };
class StudentPracujący : public Pracownik,
    public Student { ... };

StudentPracujący sp(...);
sp.Nazwisko("Tarzan"); // niejednoznaczne wywołanie
sp.Student::Nazwisko("Tarzan"); // OK
sp.Pracownik::Nazwisko("Kowalski"); // OK
-- student sp ma dwa niezależne nazwiska
```

Ryc. 10 Przykład dziedziczenia wielokrotnego z tego samego źródła. [1]
*Fig. 10 Example of multiple inheritance from the same source*

### 3.1 Projektowanie struktur danych w delegacji

Delegacja jest bardzo dobrą alternatywą dla dziedziczenia implementacyjnego oraz koncepcją projektowania struktur danych, zgodnie z którą operacje, które można wykonać na danym obiekcie, są własnością innego obiektu; innymi słowy, są "oddelegowywane" do innego obiektu. Delegację można uważać za alternatywę dla klasycznego mechanizmu dziedziczenia implementacyjnego, gdyż w przypadku delegacji mamy do czynienia z dziedziczeniem dynamicznym.
Na rysunku poniżej do implementacji stosu wykorzystana jest implementacja listy. Na diagramie (a) klasa Stos jest podklasą klasy Lista, w związku z czym dziedziczy jej wszystkie własności. Nie jest to jednak sytuacja pożądana, ponieważ stos powinien udostępniać użytkownikowi tylko właściwe dla siebie operacje takie jak push, pop, top itd., podczas gdy udostępnia także operacje charakterystyczne dla listy. Alternatywne rozwiązanie posiadające tę cechę przedstawia diagram (b), gdzie klasa Stos zawiera atrybut zawartość typu Lista - w ten sposób obiekt klasy Stos składa się z podobiektu będącego wystąpieniem klasy Lista, w wyniku czego obsługa obiektu modelującego stos jest (częściowo) oddelegowana do obiektu modelującego listę. Na poziomie implementacyjnym takie rozwiązanie oznaczałoby wywołanie w ciele metod push, pop i top odpowiednich metod zdefiniowanych w klasie Lista. Bezpośredni dostęp do metod klasy Lista dla wystąpień klasy Stos, jak na schemacie (a), nie jest tu możliwy. [2]




Ryc. 11 Przykład delegacji.[2]
*Fig. 11 Delegation example*

### 3.2 Jak jest stworzona delegacja oraz jakimi regułami się cechuje

Delegacja jest także specjalnym typem danych, który przechowuje referencję (adres) do procedury lub funkcji.
W środowisku .NET delegacja jest odpowiednikiem wskaźnika (pointer) do funkcji znanego z języka C/C++. W małych projektach można się obyć bez używania delegacji, w większych jest to praktycznie niemożliwe. Środowisko .NET korzysta z delegacji powszechnie, a ich dobrym przykładem jest obsługa zdarzeń z późnym wiązaniem (z wykorzystaniem instrukcji AddHandler).
Nazwa delegacji powinna być zgodna z regułami tworzenia nazw w .NET. Delegacja może zawierać (ale nie musi) parametry, w przypadku zbudowania delegacji do funkcji obowiązkowo musi być określony zwracany typ danych. Delegacja może wskazywać adres tylko tej procedury czy funkcji, której sygnatura (zestaw parametrów, zwracany typ danych) jest zgodna z sygnaturą delegacji.
Przed wykorzystaniem delegacji musimy utworzyć procedury czy funkcje, które będziemy chcieli kojarzyć z delegacją. Obowiązuje zasada, że z delegacją można skojarzyć dowolną procedurę czy funkcję pod jednym warunkiem: zgodności sygnatur. Jeżeli instancja delegacji już istnieje, to można zmienić adres procedury czy funkcji. Składnia języka C++ umożliwia definiowanie klas abstrakcyjnych. Kompilator języka uniemożliwia tworzenie zmiennych (typów funkcji i metod oraz parametrów wejściowych funkcji i metod), których typem jest klasa abstrakcyjna. [3]

```
' utworzenie delegacji do procedury z jednym parametrem typu String
Delegate Sub MessageDelegate(ByVal jg As String)

' utworzenie dwóch procedur, które będą wywoływane via instancja
 delegacji
Private Sub SendToMessageBox(ByVal txt As String)
   MessageBox.Show(txt, Me.Text)
End Sub


Private Sub SendToEventLog(ByVal txt As String)
   ' utworzenie logu aplikacji
   Dim AppLog As New EventLog
   AppLog.Source = Me.Text
   AppLog.WriteEntry(txt)
End Sub

' utworzenie instancji delegacji z jednym parametrem typu String
Dim SendMessage As MessageDelegate

' utworzenie instancji delegacji typu MessageDelegate wskazującej
  adres procedury SendToMessageBox
SendMessage = New MessageDelegate(AddressOf SendToMessageBox)

' utworzenie instancji delegacji typu MessageDelegate wskazującej
  adres procedury SendToEventLog
SendMessage = New MessageDelegate(AddressOf SendToEventLog)
```

Ryc. 12 Przykład skojarzenia delegacji z procedurą. [3]
Ryc. 12 Example of associating a delegation with procedure

### 3.3 Programowanie zdarzeniowe

Programowanie sterowane zdarzeniami jest metodologią tworzenia programów komputerowych, która określa sposób ich pisania z punktu widzenia procesu przekazywania sterowania między poszczególnymi modułami tej samej aplikacji. Programowanie sterowane zdarzeniami jest mocno powiązane ze środowiskami wieloprocesowymi, z graficznymi środowiskami systemów operacyjnych oraz z programowaniem obiektowym.
Jest to paradygmat programowania, według którego program jest cały czas „bombardowany" zdarzeniami, na które musi odpowiedzieć, i zakładający, że przepływ sterowania w programie jest całkowicie niemożliwy do przewidzenia z góry.
Programowanie zdarzeniowe jest dominującym typem programowania GUI – zdarzenia to naciśnięcia myszy,

klawiszy, żądania odświeżenia przez system okienkowy, różne zdarzenia sieciowe i inne.
Jest też używane przez wysoce wydajne serwery sieciowe – zdarzeniami są tu żądania połączenia, nadejście danych do odbioru, zwolnienie się miejsca w buforach wysyłania odbioru itd.
W systemach uniksowych zwykle wszystkie połączenia (np. z plikami, sieciowe, z relacyjną bazą danych) mają charakter deskryptorów plików i na ich zbiorze jest wywoływana funkcja systemu operacyjnego select lub poll, która informuje, na jakim deskryptorze wydarzyło się jakieś zdarzenie.
W programowaniu zdarzeniowym ważne jest żeby nie obsługiwać zbyt długo danego zdarzenia, bo blokuje się w ten sposób obsługę innych. W przypadku serwerów obniżyłoby to znacznie wydajność, w przypadku GUI program zbyt wolno odpowiadałby na akcje użytkownika. Można to osiągnąć za pomocą asynchronicznego I/O, wielowątkowości, rozbijania zdarzenia na podzdarzenia i wielu innych mechanizmów. [12][13]

### 4.1 Działanie programów na starszych systemach

W starych systemach operacyjnych takich jak na przykład DOS w jednej chwili w komputerze mogła pracować tylko jedna aplikacja. Sterowanie jej działaniem odbywało się bądź poprzez jawne zadeklarowanie kolejności wykonania kodu (w programie jednoprzebiegowym), bądź poprzez pętlę obsługi wyboru opcji (w programie interaktywnym). W miarę rozwoju środowisk wieloprocesowych ten stary sposób obsługi interaktywności przestał się sprawdzać. W jednej chwili w systemie operacyjnym mogło pracować wiele różnych programów. Projektanci systemów nie mogli pozwolić poszczególnym programom na całkowite przejęcie kontroli nad systemem w taki sposób, by inne programy nic nie mogły robić. W związku z tym zaczęły pojawiać się pomysły na zdarzeniowe sterowanie programami. Idea tego rozwiązania była zupełnie inna od dotychczas stosowanej. Pętla odbierająca zdarzenia od urządzeń zewnętrznych, które pozwalały użytkownikowi komunikować się z programem, została utworzona tylko w jednym egzemplarzu dla całego systemu operacyjnego. Obsługą tej pętli (w tym przekazywaniem sterowania do poszczególnych programów) zajmował się sam system. Wszystkie procedury reakcji na zdarzenia nie są więc wywoływane jawnie w każdym programie. Zamiast tego we wszystkich programach tworzone są delegacje do obsługi odpowiednich zdarzeń.
Kilka znaczących różnic między „starą” pętlą obsługi zdarzeń a programowaniem sterowanym zdarzeniowo, oto one:

**A. Nie występuje (w sposób jawny) pętla obsługi zdarzeń.**
**B. Rozdział zdarzeń scedowano na system operacyjny.**
**C. Konwersja zdarzeń z poziomu sprzętu (pochodzących od urządzeń zewnętrznych takich jak klawiatura, mysz czy ekran dotykowy) na zdarzenia typu znaczeniowego (na przykład „wybrano opcję menu”, „naciśnięto przycisk ekranowy” itd.).**
**D. Obsługa menu (zmiana wyglądu i przejście między jego opcjami) jest wykonywana przez system operacyjny (również niejawnie).**

Przykładem programowania zdarzeniowego może być porównanie metod sterowania programem.

```
PoczątekProgramu
WyświetlMenu(
  "1.Faktura",
  "2.Dodaj klienta",
  "3.Indeksacja bazy",
  "4.Koniec pracy")
AktywnaOpcja=1
PowtarzajPętlę:
  PodświetlMenu(AktywnaOpcja)
  PobierzDaneZKlawiatury(Znak)
  Jeżeli (Znak=Enter)
  wtedy WybierzAkcję(AktywnaOpcja):
    1: WystawFakturę
    2: DodajNowegoKlienta
    3: IndeksujBazęDanych
    4: PrzerwijPętlę
  KoniecWyboruAkcji
  Jeżeli (Znak=(Strzałka, Home, End, PgUp, PgDn))
  wtedy ZmieńAktywnąOpcję(AktywnaOpcja, Znak)
KoniecPętli
KoniecProgramu
```

Ryc. 13 Przykład fragmentu decyzyjnego zapisanego w pseudokodzie starsza wersja. [3]
*Fig. 13 Example of decision fragment written in pseudocode - older version*

```
PoczątekProgramu
ZaładujProceduryObsługiZdarzeńDoPamięci(
  WystawFakturę,
  DodajNowegoKlienta,
  IndeksujBazęDanych)
DodajDoSystemuMenu(
  "1.Faktura",
  "2.Dodaj klienta",
  "3.Indeksacja bazy",
  "4.Koniec pracy")
ZdefiniujPrzypisanieZdarzeńDoProcedur(
  NaZdarzenie WybranoOpcjęFaktura reaguj:
  WystawFakturę
  NaZdarzenie WybranoOpcjęDodajKlienta reaguj:
  DodajNowegoKlienta
  NaZdarzenie WybranoOpcjęReindeksacja reaguj:
  IndeksujBazęDanych
  NaZdarzenie WybranoOpcjęKoniecPracy reaguj:
  (UsuńZPamięciProceduryObsługiZdarzeń,
  UsuńDefinicjePrzypisaniaObsługiZdarzeńTegoProgramu,
  PrzekażSterowanieSystemowiOperacyjnemu)
)
PrzekażSterowanieSystemowiOperacyjnemu
KoniecProgramu
```

Ryc. 14 Przykład fragmentu decyzyjnego zapisanego w pseudokodzie- nowa wersja. [3]
*Fig. 14 Example of decision fragment written in pseudocode - new version*

### 4.2 Programowanie obiektowe

Programowanie obiektowe zaczęło być popularne wraz z rozpowszechnieniem się środowisk graficznych, które miały charakter obiektowy, mimo iż często powstawały jeszcze w językach strukturalnych. Jednocześnie ze względu

na swoje możliwości środowiska próbowały implementować wieloprocesowość. Nic więc dziwnego, że obie te metody tworzenia programów scaliły się ze sobą. Było to o tyle naturalne, że połączenie programowania obiektowego i sterowanego zdarzeniami znacznie podniosło walory obu tych technik. Zamiast uruchamiać jakieś procedury na podstawie zdarzenia, lepiej przesyłać je do konkretnego obiektu – niech on obsłuży je sobie we własnej piaskownicy. System zyskiwał na tym większą swobodę działania, nie musiał decydować o tym, jaką procedurę wykonać po naciśnięciu przycisku ekranowego 1, a jaką po 2. Odsyłał do obiektu-okna zdarzenie „naciśnięto przycisk ekranowy (numer przycisku)" i był wolny. Programowanie obiektowe zyskało hermetyzację i bezpieczeństwo. System nie wtrącał się do tego, co obiekt robi z tym zdarzeniem. Obecnie nikt już nie wyobraża sobie innego sposobu sterowania programem obiektowym niż za pomocą zdarzeń. [15][16]

### 4.3 Różnice między dziedziczeniem implementacyjnym a delegacją

Obie funkcje są różnymi narzędziami i pełnią inne funkcje w kodzie programisty. Dziedziczenie implementacyjne służy do ułatwienia procesu tworzenia klas i skrócenia kodu. Funkcje podrzędne w dziedziczeniu implementacyjnym zachowują pełne zachowanie klasy nadrzędnej,
a unikamy zbędnego kopiuj/wklej. Jednak możemy dziedziczyć tylko po jednej klasie, dlatego musimy uważać żeby jak najlepiej wykorzystać tę funkcję. Delegacja służy do wykonania pewnego działania. Nie tworzy się cała podklasa, tylko wykorzystana zostaje konkretna funkcja, bądź zmienione zostają konkretne dane z klasy, dlatego często wykonuje się ona szybciej i bez zbędnego zabierania pamięci systemowej. Dodatkowo możemy dzięki niej posiadać referencję do wielu klas oraz udostępniać ich metody. [4]

### 4.4 Porównania

Dziedziczenie można porównać do tworzenia stylów akapitowych w programie InDesign. W programie tym tworzymy styl dla tekstu, dzięki czemu mamy cały czas dostęp do zmian. Wystarczy, że raz zaplanujemy jak ma ten tekst wyglądać, a gdy będziemy chcieli zmienić napisany przez nas tekst, nie będziemy musieli martwić się o upewnienie, czy na pewno nowy i stary tekst jest taki sam. Gotowe style możemy powielać i edytować podobnie jak z klasami pochodnymi w dziedziczeniu. Obie te rzeczy ułatwiają pracę, jedne dziennikarzom i autorom książek, a te drugie programistom. [9]
Delegacje można zobrazować na przykładzie aplikacji Trello w której możemy powierzać ludziom zadania na dany dzień lub też przydzielać je całym grupom osób. Dzięki temu prace mogą iść w lepszy, bardziej zorganizowany sposób, łatwiej jest też zaplanować przebieg prac oraz na jakim etapie się aktualnie znajdujemy. W delegacji jest podobnie, pozwala ona zapanować nad chaosem i umożliwia lepszą organizację tego kodu. [6][7][8]

## 5. WNIOSKI

Wykorzystywanie dziedziczenia implementacyjnego oraz delegacji jest bardzo potrzebne, skuteczne
a także przyjemne. Są przydatnymi narzędziami programistycznymi i powinny być wykorzystywane w każdym większym projekcie czy kodzie, gdyż dzięki nim tekst programisty staje się bardziej przejrzysty i łatwiej z niego korzystać oraz pisać czy też poprawiać. Funkcje te pozwalają łatwiej rozszerzyć kod i rozwijać go w przyszłości przy dalszych zmianach w projekcie.

## LITERATURA

[1] http://fc.put.poznan.pl/materials/138-4--dziedziczenie.pdf
[2] http://edu.pjwstk.edu.pl/wyklady/pri/scb/index70.html
[3]https://forum.pasja-informatyki.pl/19901/dziedziczenie-a-delegacja
[4]https://www.altkomakademia.pl/baza-wiedzy/qna/discussion/1788/jaka-jest-roznica-miedzy-delegacja-a-dziedziczeniem/p1
[5] https://msdn.microsoft.com/pl-pl/library/84eaw35x.aspx
[6]http://wniedoczasie.pl/narzedzia/trello-zarzadzanie-zadaniami/
[7]https://www.spidersweb.pl/2016/04/trello-po-polsku.html
[8]https://www.kreatywa.net/2016/04/trello-opinia.html
[9]https://www.youtube.com/watch?v=Ba5ZujkR21o
[10]http://cpp0x.pl/kursy/Programowanie-obiektowe-C++/Polimorfizm/Dziedziczenie/494
[11]https://pl.wikibooks.org/wiki/C%2B%2B Dziedziczenie
[12]https://pl.wikipedia.org/wiki Programowanie_sterowane_zdarzeniami
[13]https://4programmers.net/Forum/C_i_C++/248556-programowanie_zdarzeniowe
[14]http://icis.pcz.pl/~agrosser/test/mijp-I.pdf
[15]https://pl.wikipedia.org/wiki/Programowanie_obiektowe
[16]https://webmastah.pl/jak-programowac-obiektowo-cz-1-wstep/

# Instrukcje warunkowe

*Conditional statements*

**Daria Wawryk[1], Elżbieta Turkiewicz[1], Jakub Lachowicz[1]**

**STRESZCZENIE:** Tematem przewodnim artykułu są instrukcje warunkowe. Poniżej zostaną omówione poszczególne typy instrukcji warunkowych, ich zastosowań oraz składni. Zostaną również przedstawione schematy ideowe przedstawiające zasadę działania owych instrukcji.

**Słowa kluczowe:** instrukcja warunkowe, instrukcja if, instrukcja if-then, instrukcja else-if, instrukcja switch, dopasowanie do wzorca, pętla while, pętla until, instrukcja wyboru, pętla, wyrażenia symboliczne, instrukcja wyboru, warunek.

**ABSTRACT:**
This article focuses on conditional statements. Their different types, syntax and applications will be discussed. There will also be presented diagrams ilustrating their principle of operation.

**Keywords:**
conditional statements, if statments, if-then statements, else-if statement, pattern matching,while loop, until loop, ternary operator, symbolic statements, condition.

## 1.WSTĘP

Instrukcja warunkowa jest składową języka programowania, za pomocą której umożliwione jest wykonywanie różnych obliczeń zależnie od tego, czy definiowane przez programistę wyrażenie jest prawdziwe czy fałszywe. Współczesne komputery cechują się możliwością warunkowego decydowania o tym, jaki krok zostanie wykonany w dalszej kolejności. Każdy model obliczeniowy zdolny do wykonywania algorytmów musi zawierać instrukcję warunkową [1]. W imperatywnych językach programowania używa się terminu instrukcja warunkowa, podczas gdy w programowaniu funkcyjnym preferowane są nazwy wyrażenie warunkowe lub konstrukcja warunkowa, gdyż posiadają one inną zasadę działania. Instrukcja warunkowa sprawia, że pewne instrukcje wykonają się przy spełnionym bądź nie spełnionym warunku. Dzięki temu rozwiązaniu program nie wykonuje zawsze ciągu instrukcji w linearny sposób lecz w zależności od napotkanej sytuacji może wybrać właściwą drogę. Dzięki niej komputery są w stanie rozwiązywać różne problemy na wielu poziomach skomplikowania. Przykładowa instrukcja warunkowa napisana w języku C++ wygląda następująco:

if(warunek) instrukcja$_X$; else instrukcja$_Y$;
warunek jest wyrażeniem logicznym, które może przyjmować wartość false albo true. W zależności od jego wartości wykonywana jest jedna z instrukcji:
warunek = true ─ wykonywana jest instrukcja$_X$, a instrukcja$_Y$ zostaje pominięta
warunek = false ─ pominięta zostanie instrukcja$_X$, a wykonana instrukcja$_Y$. [11]

## 2.INSTRUKCJE WARUNKOWE, INSTRUKCJE WYBORU, PĘTLE - ZASADY DZIAŁANIA

Spośród instrukcji warunkowych wyróżniamy:

- Instrukcję warunkową if-then
- Instrukcję warunkową else-if
- Operator warunkowy
- Instrukcję wyboru( instrukcję switch-case)
- Dopasowanie do wzorca

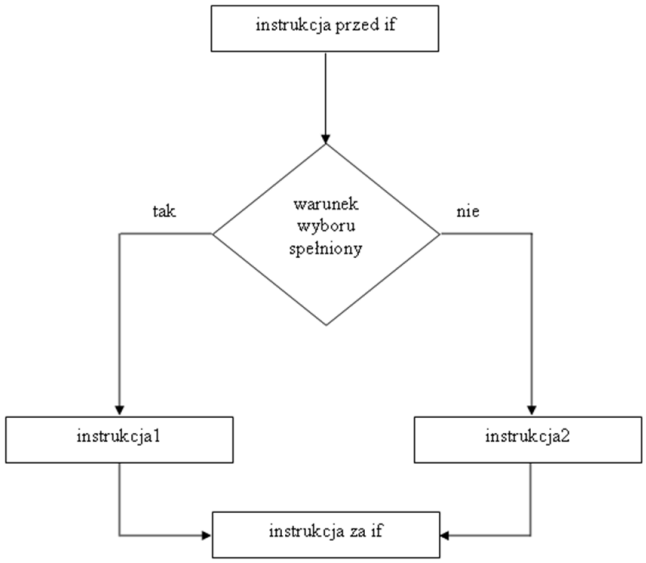
1.Wrocławska Wyższa Szkoła informatyki Stosowanej “Horyzont” Wejherowska 28, 54-239 Wrocław elzbietaturkiewicz0@gmail.com,

W dalszej części publikacji zostaną dokładnie omówione poszczególne typy instrukcji warunkowych. Natomiast w ostatniej części została omówiona jedynie pętla warunkowa.

**If-Else, If-Then oraz Else- If**

Podstawowym rodzajem instrukcji warunkowej jest If-Then. Wykorzystywana jest w większości języków programowania w celu warunkowego wykonania określonego bloku kodu, a w przypadku gdy warunek jest niespełniony, odpowiada za wykonanie bloku alternatywnego. W różnych językach programowania instrukcje różnią się nieznacznie składnią, natomiast schemat ogólny zawsze wygląda następująco:

Pierwszym krokiem jest analiza warunku podanego w postaci wyrażenia logicznego. Następnie, w przypadku gdy wynikiem jest true, wykonywany jest właściwy blok kodu, a gdy wynikiem jest false – alternatywny blok kodu.




Ryc. 1Schemat działania instrukcji warunkowej If-Else
*Fig. 1 If- Else statement principle of operation*

W wielu językach programowania istnieje możliwość zdefiniowania więcej niż jednego warunku sprawdzającego przy użyciu opcjonalnego bloku else-if. Dzięki temu możemy rozszerzyć podstawową instrukcję warunkową if, przy użyciu słowa else. Słowo kluczowe else jest jednoznaczne z ”w przeciwnym wypadku”, co oznacza, że jeśli warunek nie zostanie spełniony wykonany zostanie inny kod. Generalny zapis instrukcji warunkowej, który zawiera instrukcje else wyglądać będzie następująco: if (…) … else… , gdzie trzy ostatnie kropki oznaczają instrukcje które mają zostać wykonane w przypadku, gdy warunek if nie został spełniony.
Przykład działania instrukcji warunkowej if-then:

if warunek x **then**
  pierwszy blok kodu
**else-if** warunek y **then**
  drugi blok kodu
**else-if** warunek z **then**
  trzeci blok kodu
**else**
  alternatywny blok kodu
**end if**

W takim wypadku warunki analizowane są kolejno aż do momentu, gdy któryś z nich nie da wartości true – wtedy zostaje wykonany przypisany mu blok kodu. Gdy wystąpi sytuacja, w której żaden z warunków nie okaże się prawdziwy, wykonany zostanie blok alternatywny. Ilość prawdziwych warunków nie ma znaczenia, ponieważ zawsze wykona się tylko pierwszy z nich, a pozostałe zostaną pominięte.

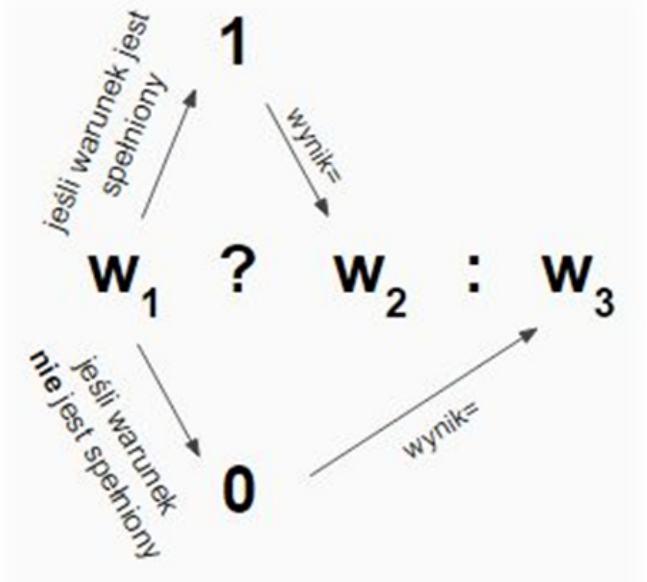
Istnieją języki programowania w których występuje konstrukcja alternatywna do opisanej powyżej instrukcji warunkowej, natomiast jej działanie jest odwrotne do instrukcji warunkowej if-else. Polega ona na tym, że pierwszy blok kodu jest wykonywany gdy pierwszy warunek jest niespełniony, a w przeciwnym razie wykonywane są inne warunki fraz else. Jest to więc równoważnik konstrukcji if not (warunek) then (instrukcja). Aby odróżnić tę konstrukcję od pierwotnej strosuje się inne słowo kluczowe, np. Unless. [3]

**Operator warunkowy**

Wyrażenie warunkowe różni się od instrukcji warunkowej If-Then tym, że wykonany blok kodu musi zwrócić jakąś wartość, która jednocześnie jest wynikiem całego wyrażenia. Wyrażenia warunkowe często występują w funkcyjnych językach programowania. W programowaniu jest konstrukcją języka programowania, w których odróżnia się wyrażenia od instrukcji, będącą formą instrukcji warunkowej wyrażoną za pomocą operatora trójargumentowego; bywa ona instrukcją wyrażeniową. Umożliwia ona sprawdzenie warunku na poziomie wyrażenia, co w pewnym stopniu zaciera rozróżnienie między wyrażeniami a instrukcjami, dzięki czemu przy jej rozsądnym używaniu kod źródłowy może zyskać na zwięzłości i prostocie.

W języku C/ C++ operator ten ma postać:
warunek ? wyrażenie1 : wyrażenie2

Celem działania operatora jest sprawdzenie wartości logicznej wyrażenia i zwrócenie jej na podstawie jednej z dywiz: wyrażenie 1 gdy warunek jest prawdziwy oraz wyrazenie 2 gdy nie jest, gdy wyrażenie nie zostało zwrócone, zwykle wartość tego wyrażenia jest niepoliczalna.




Ryc. 2 Schemat sprawdzania warunku [12]
*Fig. 2 Ternary operator working principle*

**Instrukcja wyboru**

Jest to instrukcja decyzyjna, czyli instrukcja w danym języku programowania, dająca możliwość wyboru spośród wielu opcji. Składnia instrukcji wyboru zależna jest od języka programowania, lecz istnieją charakterystyczne elementy takie jak: nagłówek, ciało, koniec.

nagłówek instrukcji wyboru:
- słowo kluczowe rozpoczynające instrukcję (np. case, select, switch)
- opcjonalnie wyrażenie, na podstawie którego następuje wybór
- słowo łączące (np. of, do)

ciało instrukcji wyboru:
- kolejne instrukcje podlegające selekcji
- opcjonalnie poprzedzone frazami zawierającymi wartości porównywane z wyrażeniem z nagłówka instrukcji
- opcjonalnie poprzedzone słowem kluczowym (np. case, when)
- słowo lub symbol łącznika (np. ”do”)
- opcjonalnie fraza domyślna wykonywana, gdy żadna z fraz nie spełni warunku (np. else, otherwise, other, default, when else)

koniec instrukcji wyboru – słowo zamykające blok (np. end, end case, end select itp.)

Poniżej zaprezentowany został podstawowy kod zawierający instrukcję wyboru switch-case. Oraz jego omówienie

```
switch( <wyrażenie> )
{
case <wartośćA>:
instrukcjaX;
instrukcjaY;
 //...
break;
case <wartośćB>:
instrukcjaX;
instrukcjaY;
//...
break;
//...
default:
instrukcjaX;
instrukcjaY;
//...

}
```

Konstrukcja switch-case umożliwia wybór więcej niż jednej opcji. Instrukcja switch oblicza wartość <wyrażenia> i przypisuje go do jednej z podanych opcji. Wszystkie opcje muszą być zawarte w bloku. Po słowie kluczowym case należy podać <wartość>, a następnie dwukropek. Po dwukropku występuje instrukcja, która nie musi być zawarta w nowym bloku.
Gdy wartość wyrażenia będzie odpowiadała jednej z wartości, to wykonywane są wszystkie instrukcje występujące pomiędzy nią, a kolejnym słowem kluczowym-break.
W wypadku gdy żadna z wartości nie będzie pasowała do wartości wyrażenia, wdrożone zostaną instrukcje znajdujące się po słowie kluczowym default.
Główną różnicą między instrukcją warunkowa if a switch jest to, że w regularnej instrukcji if możemy określić dokładnie co ma się wydarzyć w zależności od stanu jednej lub kilku zmiennych. Instrukcja if daje nam pełną kontrolę nad przebiegiem programu. W języku C++ jest jednak dostępna również instrukcja wielokrotnego wyboru switch. W przypadku niej możemy wykonywać decyzje tylko na podstawie wartości jednej zmiennej. Możliwości instrukcji switch są nieporównywalnie mniejsze, jednak używanie jej w niektórych przypadkach jest znacznie korzystniejsze dla szybkości działania programu i estetyki kodu niż użycie instrukcji if. Aby poprawnie używać instrukcji switch-case należy zapoznać się z pewnym warunkiem, który określa, jakie dozwolone typy danych można używać. Tak więc dozwolonymi typami danych w instrukcji switch są liczby całkowite. Oznacza to, że możemy użyć tylko zmiennych, które są typów takich jak: char, short, int, long, long long. Do tego dochodzi również typ wyliczeniowy enum [12].

**Dopasowanie do wzorca**

Jest to operacja, w której pewne wyrażenie sprawdza się ze wzorcem, w którym może znajdować się jedno lub więcej "wolnych miejsc". W wyniku, o ile nastąpiło dopasowanie, otrzymuje się listę wyrażeń, które dopasowały się do wolnych miejsc wzorca.
Dopasowywanie do wzorca jest bardzo ekspresywną techniką programistyczną. Dwa najpopularniejsze systemy to:

- wyrażenia regularne
- wzorce symboliczne

Wyrażenie regularne

W większości nowych języków wyrażeń regularnych można używać jako wzorców, np (Perl):

```
{
   # pasuje do wzorca, kolejne liczby w $1 $2 $3 $4
} else {
   # nie pasuje do wzorca
}
```

Alternatywa wyżej przedstawionego kodu w języku C musiałaby wykorzystywać bibliotekę, która umożliwiłaby dopasowanie do wzorców (np. PCRE), w przeciwnym wypadku istniały by duże szanse na wystąpienie błędów.

Wyrażenia symboliczne

Wzorce symboliczne wykorzystywane są w językach funkcyjnych, są to termy ze zmiennymi, które dopasowują się do danego wyrażenia przez unifikację. Dostępny jest również specjalny symbol uniwersalny pasujący do każdego obiektu; w wielu językach to znak podkreślenia _.
Na przykład (w Ocamlu) zamiana wartości liczbowej na ciąg znaków z użyciem dopasowania można zrealizować następująco:

```
match liczba with
    0 -> "zero"
|   1 -> "jeden"
|   2 -> "dwa"
|   _ -> "inna liczba"
```

To samo można uzyskać instrukcją warunkową:

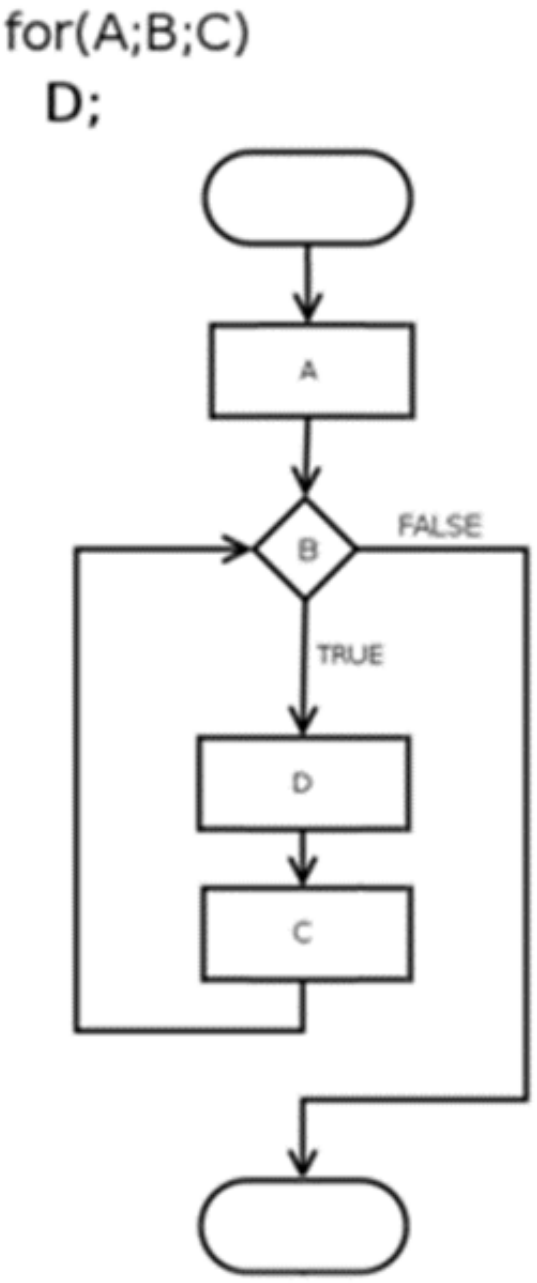
```
if liczba = 0 then "zero" else
if liczba = 1 then "jeden" else
if liczba = 2 then "dwa" else "inna liczba"
```

Jednak ten zapis jest bardziej rozwlekły i mniej czytelny.
Jeszcze inny przykład wykorzystujący konstruktory list; zapis x::lista oznacza doklejenie na początek listy elementu x, wynikiem jest nowa lista.

```
match zmienna with
        [] -> "Lista jest pusta"
|     x::[] -> "Lista ma jeden element: " ^ x
|   x::y::[] -> "Lista ma dwa elementy: " ^ x ^ " i " ^ y
| x::y::z::[] -> "Lista ma trzy elementy: " ^ x ^ ", " ^ y ^" i " ^ z
| x::y::z::_  -> "Lista ma więcej niż trzy elementy: " ^ x ^ ", " ^ y ^", " ^ z ^ "..."
```

**Pętla**

Pętla jest to jedna z trzech podstawowych konstrukcji używanych w programowaniu strukturalnym. Daje ona możliwość wykonywania ciągu instrukcji w sposób cykliczny określoną liczbę razy aż do momentu, kiedy zajdą pewne warunki dla każdego elementu, lub w nieskończoność.




Ryc. 3 schemat działania pętli for [16]
*Fig. 3 For loop principle of operation diagram*

**Pętla warunkowa**

Pętla warunkowa jest to rodzaj pętli, w której programista definiuje warunek, od którego zależne jest wykonanie kolejnej iteracji. Warunek zawarty w definiowanej pętli jest pewnym wyrażeniem, które zwraca wartość typu logicznego, (np. Pascal, Visual Basic, VBA) natomiast nie w każdym jezyku programowania składnia przewiduje taki typ danych. W niektórych jezyka definiowane są wyrażenia które zwracają wartość innego typu. Następnie zostaje ona poddana interpretacji, np. wartość 0 może być jednoznaczna z wartością false typu logicznego, a pozostałe z wartością true. Zależnie od tego, czy wartość logiczna uzyskana w wyniku ewaluacji wyrażenia przedstawiajace-

go warunek jest równa wartości logicznej true (prawda), czy false (fałsz), wykonywanie pętli jest kontynuowane bądź przerywane [8] [9] [11] [14].
Podsumowując, pętla na początku definiowana jest przy użyciu określonego wyrażenia, za pomocą którego określone zostaje, czy nastąpi przejście do kolejnej iteracji, w przeciwnym wypadku nastąpi zakończenie wykonywania pętli, czego konsekwencją będzie przejście do kolejnej instrukcji znajdującej się już poza pętlą. W zależności od składni konkretnego jezyka programowania. Powszechnym jest zapis wyrażeń kontrolnych analogicznie do zapisu tych wyrażeń dla instrukcji warunkowej. Ogromną rolę w tym wypadku odgrywają operatory porównawcze, mimo że warunek może zostać wyrażony również w inny sposób, np. w postaci wywołania funkcji zwracającej wartość logiczną bądź jako identyfikator zmiennej logicznej, której wcześniej przypisano rezultat ewaluacji wyrażenia warunkowego. W programowaniu bardzo użyteczna jest również dostępność operatorów logicznych dająca możliwość budowania warunków złożonych z kilku warunków łącznych lub alternatywnych (zdaża się, że w konkretnym języku mogą być dostępne inne operatory logiczne, np. alternatywy wykluczającej czy implikacji).
Kluczową rolę odgrywa możliwść decydowania o kontynuacji lub zaprzestaniu wykonywania pętli. Możliwość sprawdzenia warunku może występować na różnych etapach działania pętli. Sprawdzanie może odbywać się na początku pętli, tzn. przed wykonaniem pierwszej instrukcji zawartej w bloku danej pętli. Inną możliwością jest weryfikacja warunku wewnątrz pętli, tzn. w jej bloku, pomiędzy wykonaniem części instrukcji, lecz przed zakończeniem wykonywania pozostałych). Ostatnią możliwością jest sprawdzenie warunku na końcu pętli – po wykonaniu wszystkich instrukcji, które są zawarte w definiowanej pętli).
W sytuacji, kiedy warunek sprawdzany jest na początku pętli, istnieje duże prawdopodobieństwo, że instrukcja zawarta w pętli nigdy się nie wykona. Taka możliwość istnieje gdy przy pierwszym wykonaniu warunek nie będzie spełniony. Zupełnie inaczej jest w przypadku, kiedy warunek sprawdzany jest na końcu pętli, wtedy instrukcje muszą wykonać się chociaż jeden raz. Sytuacją łączącą obie te możliwości jest warunek sprawdzany wewnątrz pętli. Oznacza to, że instrukcje zapisane przed sprawdzeniem warunku zostaną wykonane minimum raz, a instrukcje występujące po warunku sprawdzającym mogą się w ogóle nie wykonać.

Podstawowymi konstrukcjami składowymi są :
- while
- until
[12] [13] [14] [15]

**Zastosowania i budowa przedstawionych warunków**

Warunek **while** wykorzystuje się w sytuacjach, kiedy niezbędne jest wykonywanie danej operacji, dopóki nie zostanie spełniony warunek. Warunek while ma następującą składnię:

*while* (warunekKoncowy)
  lista_instrukcji

Pętla while podobnie jak pętla **for** oraz jak pętla **do while** umożliwia powtarzanie instrukcji tak długo jak warunek końcowy jest spełniony, a warunek jest zapisywany na samym początku pętli dlatego, że warunek jest sprawdzany zanim cokolwiek innego zostanie wykonane i w ten sposób jeśli warunek nie jest spełniony, nic nie zostanie wykonane.
Brak jest nawiasów klamrowych, bowiem nie są one wymagane, kiedy chcemy wykonać tylko jedną instrukcję. Należy również zauważyć to, że po nawiasach okrągłych znajdujących się za słowem while nie ma średnika.
Innym przypadkiem jest warunek **Until**, gdzie słowo kluczowe w różnych językach ma różne implikacje:
- warunek który określa, iż pętla będzie powtarzana, dopóki warunek nie stanie się spełniony,
- warunek, który jest sprawdzany na końcu pętli (choć zapis warunku znajduje się na jej początku w nagłówku definiującym), a pętla jest powtarzana jeżeli warunek jest spełniony.

Instrukcja repeat ... until (*nazywana pętlą "powtarzaj"*)
Instrukcja ta wykonuje cyklicznie inne instrukcje zawarte pomiędzy słowami repeat i until do momentu gdy *wyrażenie* znajdujące się za słowem until nie przyjmie wartości *prawda* (czyli *true*).
Efekt zastosowania pętli repeat jest bardzo podobny do działania pętli while. Pętla ta także może być wykonywana ogromną liczbę razy. Jedyna różnica polega na tym, że w pętli repeat warunek zakończenia sprawdzany jest dopiero po wykonaniu instrukcji. Oznacza to, że pętla repeat zawsze będzie wykonana co najmniej raz. Dopiero po tej iteracji program sprawdzi, czy można zakończyć działanie pętli. W przypadku pętli while warunek jest sprawdzany bezpośrednio przed jej wykonaniem, co w rezultacie może spowodować, że taka pętla nigdy nie zostanie wykonana.

Budowa pętli repeat jest następująca:
**repeat**
  { instrukcje do wykonania }
**until** { warunek zakończenia }
[16]

**Warunki w maszynach Turinga**

Jednym z wybitnych przykładów zastosowania warunków w praktyce jest użycie ich w maszynie Turinga. Maszyna ta jest abstrakcyjnym modelem komputera służącego do wykonywania algorytmów. Jej działanie polega na tym, że na nieskończenie długiej taśmie podzielonej na pola zapisuje się dane. Taśma może być nieskończona jednostronnie lub obustronnie. Każde pole może znajdować się w jednym z N stanów. Maszyna zawsze jest ustawiona nad jednym z pól i znajduje się w jednym z M stanów. Zależnie od

kombinacji stanu maszyny i pola maszyna zapisuje nową wartość w polu, zmienia stan, a następnie może przesunąć się o jedno pole w prawo lub w lewo. Taka operacja nazywana jest rozkazem. Maszyna Turinga jest sterowana listą zawierającą dowolną liczbę takich rozkazów. Liczby N i M mogą być dowolne, byle skończone. Czasem dopuszcza się też stan M+1, który oznacza zakończenie pracy maszyny. Lista rozkazów dla maszyny Turinga może być traktowana jako jej program. W teorii złożoności obliczeniowej maszyna Turing jest wzorcowym, matematycznym modelem obliczeń komputerowych zdolnym do wykonywania algorytmów. [10]

## 3. ZAKOŃCZENIE

Na podstawie analizy wszystkich instrukcji warunkowych oraz warunków jakie są stawiane, możemy stwierdzić, że są one niezbędne w życiu programistów,sterują przebiegiem naszego programu. Można stwierdzić, że są nieodłącznym elementem naszego życia. Każdy program, każdy sterownik i każda maszyna wykorzystują mnóstwo warunków, aby były bezpieczne, spełniały określone standardy i wymogi, gdyż to właśnie dzięki instrukcjom warunkowym program może „zachowywać" się zależnie od spełnienia pewnych warunków, tak jak od niego oczekujemy. Instrukcje te wykonują jedynie wybrany kod w zależności od tego czy wartość danego wyrażenia jest prawdą (true) czy fałszem (false). A jak powszechnie wiadomo, każdy program oparty jest na wszelkiego rodzaju warunkach.

## 4. PODSUMOWANIE

- **Zapis instrukcji warunkowej if-then wygląda następująco: if(...). Wartość, którą umieszczamy w nawiasach zaokrąglonych, to** wyrażenie logiczne. Początkowo wykonywana jest analiza warunku podanego w postaci wyrażenia logicznego. Jeśli wynikiem jest true, wykonywany jest właściwy blok kodu, a jeśli false – alternatywny.

- Słowo kluczowe **else** oznacza „w przeciwnym wypadku", czyli jeśli warunek nie zostanie spełniony, program wykona inny kod. Ogólny zapis instrukcji warunkowej **else** będzie więc wyglądał nastepująco; if(...)...else...

- **Wyrażenie warunkowe** jest odmianą instrukcji warunkowej if-then z tą różnicą, że wykonany blok kodu musi zwrócić jakąś wartość, która staje się jednocześnie wynikiem całego wyrażenia.Umożliwia ona sprawdzenie warunku na poziomie wyrażenia

- Jedynymi dozwolonymi typami danych w instrukcji **switch** są liczby całkowite.

- Instrukcja wielokrotnego wyboru **switch** są nieporównywalnie mniejsze, jednak używanie jej w niektórych przypadkach jest znacznie korzystniejsze

- W zależności od tego, czy wartość logiczna, uzyskana w wyniku ewaluacji wyrażenia reprezentującego warunek jest równa wartości logicznej true (prawda), czy false (fałsz), wykonywanie pętli jest kontynuowane bądź przerywane.

- Bardzo ważną rolę odgrwają warunki pętli decydujące o kontynuacji lub zaprzestaniu wykonywania pętli , które mogą być sprawdzane na początku, wewnątrz i na końcu pętli.

- Pętlę while wykorzystuje się w sytuacjach, kiedy niezbędne jest wykonywanie danej operacji dopóki nie zostanie spełniony warunek.

- W pętli until warunek zakończenia sprawdzany jest dopiero po wykonaniu instrukcji. Oznacza to, że pętla repeat zawsze będzie wykonana co najmniej raz.

- Dopasowanie do wzorca to operacja, gdzie pewne wyrażenie sprawdza się ze wzorcem, w którym może znajdować się jedno lub więcej wolnych miejsc.

## Literatura:

1. Arindama Singh: *Logics for Computer Science*. PHI Learning Pvt. Ltd., 2004, s. 283. ISBN 81-203-2284-3

2. Martin Richards: T*he BCPL Cintsys and Cintpos User Guide.* Cambridge: Computer Laboratory University of Cambridge, January 28, 2011. [dostęp 2011-01-31]. (ang.)

3. Michał Iglewski, Jan Madey, Stanisław Matwin: Pascal. *Język wzorcowy – Pascal 360*. Wyd. wydanie trzecie – zmienione. Warszawa: Wydawnictwa Naukowo-Techniczne, 1984, seria: Biblioteka Inżynierii Oprogramowania. ISSN 0867-6011. ISBN 83-85060-53-7. (pol.)

4. Andrzej Marciniak: Borland *Pascal 7.0*. Poznań: Nakom, 1994, seria: Biblioteka Użytkownika Mikrokomputerów. ISSN 0867-6011. ISBN 83-85060-53-7. (pol.)

5. John Walkenbach: *Excel 2003 PL. Programowanie w VBA*.. HELION, 2004. ISBN 837361-504-0. (pol.)

6. Brian W. Kernighan, Dennis M. Ritche: *Język C*. Warszawa: Wydawnictwa Naukowo-Techniczne, 1988, seria: Biblioteka Inżynierii Oprogramowania. ISBN 83-204-1067-3. (pol.)

7. Jan Bielecki: *Turbo C z grafiką na IBM PC*. Warszawa: Wydawnictwa Naukowo-Techniczne, 1990, seria: Mikrokomputery. ISBN 83-204-1101-7. (pol.)

8. Jan Bielecki: *Od C do C++, programowanie obiektowe w języku C*. Warszawa: Wydawnictwa Naukowo-Techniczne, 1990. ISBN 83-204-1332-X. (pol.)

9. Maszyna Turinga w serwisie edukacyjnym I LO w Tarnowie [Autor artykułu: mgr Jerzy Wałaszek
Wersja 2.1 - zakończono 14 IV 2004]

10. Strona internetowa www.eduinf.waw.pl [strona odwiedzona ostatnio 30.08.2018]

11. Operator_warunkowy na stronie inyternetowej https://pl.wikipedia.org/wiki [Treść udostępniana na licencji CC BY-SA 3.0,strona ostatni raz edytowana 14 stycznia 2018]

12. https://pl.wikipedia.org/wiki/Instrukcja_wyboru [Tę stronę ostatnio edytowano 25 lip 2018, 19:57.]

13. Podręcznik Visual Basic na stronie internetowej www.Wikibooks.pl [Tę stronę ostatnio edytowano 1 sierpnia 2014, 19:06.]

14. Podręcznik języka C na stronie internetowej www.Wikibooks.pl [Tę stronę ostatnio edytowano 8 lip 2018, 21:01.]

15. Strona internetowa : https://4programmers.net/Delphi/Repeat [strona odwiedzona 30.08.2018]

16. Dopasowanie do wzorca, portali wikipedia [Tę stronę ostatnio edytowano 18 sty 2016, 09:31. ]

# Komentarze w kodach wybranych programów

*Comments in the codes of selected programs*

**Sviatoslav Skhut[1], Kateryna Iholkina[1],**

**ABSTRACT:** Writing comments is as important as writing code. The main purpose of using comments is to improve readability of our code but frequently thoughtless comment writing decrease understandability of source code. Comments must be concise and precise simultaneously. Also, when our code is changed, comments for this code must be changed too. While using comments in our code we must realize that if expressiveness of our programming language allows us to express clearly what we want in code, there is no need to write comments at all. And if we decide to use comments, they must be extremely accurate and understandable, because another person must understand, what we do and most importantly, why we do it.
Frequently comments can be replaced with good clear names of variables, functions or classes. Also, we can replace our comments with assertions. Comments should clarify and explain our intentions. Copyrights and an authorship can be implemented using comments too. But our IDE can do these things automatically.
Classification of comments depends on their place in code, for which type of code they are attached and format.

**Keywords:** source code, software and its engineering, documentation, software management, code comments

**INTRODUCTION:** Improving code commenting techniques is important for programmers for several reasons: code maintenance takes about 40-80% of the lifetime cost of a piece of software [1], besides software is rarely maintained by its original author for its whole life. It's obvious that, the author`s perception of the code is very different from this of the following developers. Some moments and solutions in the code, self-explanatory for the author, can create a lot of problems for programmers supporting or refactoring the program. Due to poor documentation and poor-quality comments it is often even easier to write a new program than to change an old one. Thus, writing a code with useless, unreliable and inaccurate comments provides to large increase in cost.
Unfortunately, there is no general standards and conventions of writing and formatting code comments. In last twenty years there was published a lot of books, thesis and blog posts on the topic of programming style, clean code writing and code documentation. Namely, "The elements of programming style" by Brian W. Kernighan and P. J. Plauger, a study supporting point of view that source code should be, first of all, human readable. We should also note Robert C. Martin`s book "Clean Code. A Handbook of Agile Software Craftsmanship". A huge part of this study describes patterns and practices of writing maintainable code and efficient comments. In addition to these books our essay is based on "Java code conventions" published by Sun Microsystems Inc. in 1997 and "C++ style guide" powered by Google Inc.
The aim of our paper is to treat a problem of code commenting in the source code of selected programs. In the first two chapters we pay attention on the different code comment`s classification based on their function, placement and format. In the third chapter we provide an analysis of costs and benefits of using comments. On the examples of different programs, we consider the best practices of code commenting that should be in each program such as legal comments, clarifications and TODO comments and explanations of intent. Also, we study examples of useless and sometimes even dangerous comments, that should never appear in the source code. The last part of this paper is devoted to techniques that make the code more clear and readable without using comments.

## 1. COMMENT TYPES

Comments can be used for different purposes in different parts of the program. They fall into one of three categories: header comments, block comments and trailing comments, which describe successively smaller areas of code.

### 1.1. Header comments.

This class of comments serves to help reader navigate, understand a general purpose of the code and use the code itself. This category of comments makes a maintenance

1.Wrocławska Wyższa Szkoła Informatyki Stosowanej „Horyzont”, ul. ks. Marcina Lutra 4, 54 - 239 Wrocław Email: K_Iholkina@gmx.com, sgshkut@gmail.com

of code easier. Thus, they should be included in any code planned to be in use more than a few weeks. The header comments usually occupy a number of lines (typically between 10 and 50) [2] and contains following elements:

- Filename
- Source control version history
- Creation date
- Revision history
- Author's name
- Copyright notice
- License summary
- Purpose
- Change history
- Restrictions
- Special hardware requirements (e.g. Analog/Digital signal converters)

Header comments are placed at the beginning of the program which makes them stand out and easier to remove or copy. The recommended practice is dividing a comment block onto the section to ameliorate its readability. Using capital letters for the section headings and tabbing information out allows to navigate and read them more quickly:

```
/*
 ...
 * GLOBAL DATA:
 *      int     DB_ErrStatus          Contains most recent database error
```

**1.2. Class comments**

If we have a non-obvious class, the comments are required. These types of comments should describe what this class serves for and how it should be used. The class comments should provide such information as: interface of the class, multiple threads (if any) and if it is possible a few small examples of code demonstrating its usage. When we separate implementation and declaration (e.g. .cpp and .h files), comments that describe use of the class should go together with declaration, comments that describe class operation and implementation should be insert into implementation file.

**1.3. Function comments**

As with class function, comments should appear when usage of the function is non-obvious. Comments attached to function declaration should describe the usage of the function. They shouldn't describe how the function performs its tasks, just tell reader in descriptive way what the function must do, what inputs and outputs are, in which way user must free memory (if the function allocates memory) also function override should be described if it is not trivial. Comments in function definition should describe operations. These types of comments describe how function works.

**1.4. Variable comments**

The actual name of variable should be enough for description of what it is used for. Comment can be attached if this variable need additional clarification. In classes we have data members. Their names must be descriptive enough too, and the comments are required if there are some non-obvious instances. Global variables should have accompanying comments that describe why they need to be global.

**1.5. Explanatory comments**

Well written and concise code will contain a lot of explanatory comments, which highly increase code readability and clearness. Even though explanatory comments are not necessary for each line of code, there are some items which definitely should have them.

For example, startup code, exit code, weird logic, regular expressions, sub routines and functions, long and complicated loops [3].

In startup code explanatory comments shoulo to mention how the program is initialized, what #defines do, what arguments are expected etc.

While writing the exit code explanator comments, we should properly treat normal and abnormal exit situations, error codes etc.

Subrouting and function explanatory comment should, first of all, clearly explain its tasks and purpose of its using. It is important to comment functions arguments passed and returned with mentioning values format and limits on values expected, as this is one of the biggest sources of bugs [4]. Thus, this kind of comments written before sub, routines greatly helps reader to gain a deep understanding of the following code.

## 2. COMMENT FORMATS

**2.1. Block comments**

Block comments are generally found within functions, methods, data structures and algorithms. Block comments have two main purposes:

- Commenting out code
- Writing long comments

In C, C++ and Java languages block comments begin with special separator "/*" and are terminated with "*/", as shown in the example below. A common used practice is to start block comments by a blank line to separate them from the rest of the code.

```
/* * Here is a block comment. */
```

The other technic of highlighting block comments is enclosing them into a rectangle box of stars and dashes.

The example of boxed comment is:

```
/**********************
 Comment in a box!!
**********************/
```

The initial "/*" could be followed by other characters such as "=", "_" or "-".

**2.2 Single-line comments**.
Single-line comment is a short comment which appears on a single line intented of the code that follows. As well as in the case of block comments, there is highly recommended to separate them from the rest of code by at least one blank line.
The example of single-line comment in Java code is:

```
if (condition) {
   /* Handle the condition. */
   ...
}
```

**2.3. Trailing comments**
Trailing comments are very short comments, which describe the action or use of a single line of code. They usually begin (and end) on the same line as the code they describe. For separating trailing comment from code it is common practice to tab it out. The comment should be far away from the code.
For example:

```
SelectSides( Players );          /* choose partners and positions    */
```

## 3. PROS AND CONS OF COMMENTING CODE

**3.1. Why comments are not always good.**
Comment can be very helpful if they are placed in correct place in right time. But frequently they just do mess or clarify the code, which we can understand without them. We need comments for clarification our motives (why we write our code that way) or even for warning about something. Nature of comments arises from low expressiveness of our programming languages. The best comment is the one that we don't need at all. That means that we can chose a good name for variable or function, decide to write an extra line that make our code more readable and understandable.
Another case is evolving our code. Chunks of it can be moved in another place or deleted or even rewrited. In this case we can face outdated misleading comments. Of course, programmers can maintain these comments, but it takes a lot of time and it`s better to change a piece of code and deal with comment only if we really need it in this place.
When we decide to write a comment, at first we need to think about people who will read it. Secondly we should do our best with this commenting. Time spending for writing a good comment will save a lot of time of our code readers.
Sometimes we can see redundant comments. They do nothing except amassing lines in our code. It means that there is no need to comment every single function or variable we declared. When we see a + b we already know what it does and there is no need to comment it. Usually this type of comments just makes it difficult to read the code.
Now we can see that comments are always helpful. There are a lot of cases when comments are not needed at all or when they just make code less readable.

**3.1.1. Journal comments**
Some programmers add a kind of "historical" comments containing their names, time or date and changes made every time they edit the code. For example:

```
// method name: pityTheFoo (included for the sake of redundancy)
// created: Feb 18, 2009 11:33PM
// Author: Bob
// Revisions: Sue (2/19/2009) - Lengthened monkey's arms
//            Bob (2/20/2009) - Solved drooling issue
 void pityTheFoo() {
    ...
}
```

There were some reasons to make log comments long, long ago, when there wasn`t a source control system making it automatically for us. Nowadays it is more likely to use one of the source control system and just fill the check-in comment boxes on our commits [5].

**3.1.2. Noise comments**
Sometimes comments are obvious and provide no new information. For example, each string of comments from code below can be discarded without loss of understandability:

```
/** The name. */
private String name;
/** The version. */
private String version;
/** The licenceName. */
private String licenceName;
/** The version. */
private String info;
```

Such code out commenting normally should be used in two cases: in code examples which serves teaching the concept of programming language, or in the case when programming language isn`t easily human readable (LikeAssembly).

**3.2. Good comments**
There are cases when we can't avoid using comments such as corporal rules or copyright. But we will not consider them now. A good example of good comment is to do comment. This type of comments can appear near the functions which will implement in the future (or not). It`s just a list of tasks that programmers want to do in future. Another example of good comments are warning comments. This type of comments warn other programmers about consequences of using code such as vulnerability or time of exe-

cution. Comments that describes our intents (why we decided to solve this problem this way or choose this data type etc.) or clarify our cod (when we really need it) are examples of the good comments too.

**3.2.1. Legal comments**
These comments should not be duplicated of contract or legal tome. Legal comments can include copyrights, refers to standard licenses, authorship. They also may refer to external documents. This type of comments should be included inck at the beginning of source file.

**3.2.2. Explanation of intent**
This type of comments explains why we have decided to use this implementation. It allows a developer to understand what is purpose of ou code. Also, it reduces situations where our intents aren't clear at a glance.

**3.2.3. Clarifications**
Sometimes the best way to describe developer explain to our code is write about it in readable form. For this reason, clarification comments may be used. This type of comments helps us to describe our obscure functions, returning values, non-obvious behavior etc.

**3.2.4. TO DO comments**
It is not a bad idea to include some "to do" lists in our code. It can be done with this type of comments. They can be connected to functions or pieces of code that we want to implement in future. Todo comments show developers that this function does nothing except reminding.

**3.3. Alternative comments**
As we mentioned above the one of the worst practice is out commenting code. Obvious, annoying, trashy, redundant comments lead to incomprehensible hard to maintain code. Having considered what comments' strengths and weaknesses are, we will treat how they could be replaced by other tools.

**3.3.1. Identifiers as comments**
Consider the example demonstrating how a typical comment can be encoded in an identifier:

Before:
```
++i;/* record another match of this expression */
```
After:
```
++number_of_expression_matches;
```

Huge part of source codes comments could be replaced by carefully and thoughtfully named variables, functions, methods and classes names. If a comment is intended to explain a complex expression, the expression should be split into understandable subexpressions using extract variable. If a comment explains a section of code, this section can be turned into a separate method via extract method.
However, identifiers could be completely misleading, if the programmer isn`t attentive when modifying code. This is the same problem which appears also when comments aren`t updated respectively to code, modifications. So, when we refactor code, we should be vigilant to change both comments and identifiers.

**3.3.2. Replacement comments with assertions**
From time to time it is reasonable to refactor a comment into an assertion. For example:

Before:
```
// value must not be negative
 public double squareRootOf(int value) {
...square root algorithm...
}
```
After:
```
public double squareRootOf(int value) {
Assert.isTrue(value >= 0);
  ...square root algorithm...
}
```

The benefits of this solutions are: supporting better testing, making debugging easier, serving as understandable comment about preconditions [6]. However, assertions slow down our code and may make a program incorrect when they are used improperly. So, assertions have some advantages as they are enforced as code and form programmable safeguards, but then also have all the disadvantages of code: expression of abstraction can be verbose and non-trivial

## 4. DOCUMETING SYSTEMS

There are several documenting systems available for various programming languages. These systems deal with the „Explanatory" type of comment. They create documentation out of comments from the code. Let's demonstrate these systems via Javadoc. This is the Java API contained in the JDK. It uses comments with specific tag /** to generate HTML pages with descriptions of all classes, interfaces, constructors etc. It also generates a tree with class hierarchy. For more details we can use documentation provided with JDK. PHP and C# also have documentation system, but the latter uses XML instead of HTML. These systems require from developers to maintain not only the code, but comments too.

## CONCLUSIONS

When reading tricky code, there is nothing more helpful than well-written comment. At the same time, there is often nothing harder than writing a well-placed, brief and clear comment.
On the one hand, plain English is always easier to read than code. Comments can explain things that couldn`t be easy expressed in programming language, besides, they don`t affect program execution speed. Writing good comments

discipline programmer`s mind. Comments are shorter than the code they document and much easier to skim-read.
On the other hand, they reduce the readability of well-written code, in addition they are less precise that the code they document. Sometimes using a lot of comments encourage bad code, take up screen space and time to read. By the way, programmers often refactor code, but don`t update comments, which provides to a high risk of spending hours tacking up a bug, because you trusted a non-reliable comment.
Thus, programmers should use comments carefully, for preference when it is impossible to make a code self-explanatory. Before writing a comment, it is recommended practice to try to increase code expressiveness by introduce an explaining variable, extract a method, use more descriptive identifier, or replace a comment with assertion.
The questions "To write or not to write?", "How many?", "How detailed comments to write?" is still hotly debated one.

SOURCES

[1] Sun Microsystems Inc, Java code conventions, 1997
[2] David Straker, C Style: Standards and Guidelines, 1991
[3] Bernhard Spuida, The fine Art of Commenting, 2002, Tech Notes, general Series
[4] Brian W. Kernighan, P. J. Plauger, The elements of programming style, 1978, McGraw-Hill Book Company,
[5] Robert C. Martin, Clean Code. A Handbook of Agile Software Craftsmanship, 2009, Prentice Hall,
[6] Dori Reuveni, Kevin Bourrillion, Code Health: to comment or not to comment, 2017, blog post

# NewConnect

**Marcin Zdanowicz** [1]

**STRESZCZENIE:** Giełda Papierów Wartościowych w Warszawie (GPW) jest spółką akcyjną powołaną do życia w 1991 roku przez podmioty reprezentujące Skarb Państwa, w tym ministra przekształceń własnościowych, oraz ministra finansów, które podpisały akt założycielski Giełdy Papierów Wartościowych w Warszawie. System obrotu na GPW bazuje na kojarzeniu zleceń kupna i sprzedaży instrumentów finansowych, a realizacja transakcji ma miejsce podczas sesji giełdowych. W obrębie grupy kapitałowej obrót realizowany jest na rynkach: Główny Rynek GPW (obrót akcjami i innymi instrumentami udziałowymi i instrumentami pochodnymi), NewConnect (alternatywny system obrotu, nieregulowany – obrót akcjami i innymi instrumentami o specyfice udziałowej — dotyczy małych i średnich spółek), Catalyst, Treasury BondSpot Poland. W 2007 roku Giełda Papierów Wartościowych w Warszawie uruchomiła rynek NewConnect. Jest to rynek organizowany, prowadzony przez giełdę poza rynkiem regulowanym, w formule systemu obrotu. Zadaniem tego rynku jest finansowanie rozwoju małych i średnich firm o wysokim potencjale wzrostu (na przykład: z sektora elektronicznych mediów), krótko działających na rynku, o względnie niskiej przewidywalnej kapitalizacji, poszukujących relatywnie niedużego kapitału. Inwestycje, z którymi wiąże się oczekiwanie ponadprzeciętnych zysków, równocześnie odznaczają się podwyższonym ryzykiem. Na NewConnect obowiązują bardziej liberalne wymogi formalne i informacyjne w stosunku do rynku giełdowego. Wymogi stawiane spółkom zgłaszającym zamiar debiutu na NewConnect są niewielkie w porównaniu z Głównym Rynkiem. NewConnect umożliwia przedsiębiorstwom na względnie szybkie pozyskanie nowego, dodatkowego koniecznego kapitału. Debiut na NewConnect możliwy jest poprzez emisję prywatną i emisję publiczną.

**SŁOWA KLUCZOWE:** NewConnect, Giełda Papierów Wartościowych w Warszawie S.A., alternatywny system obrotu, rynek akcji, spółka akcyjna, małe i średnie przedsiębiorstwa, sektor nowych technologii, sektor innowacyjny

**ABSTRACT:** The Warsaw Stock Exchange (WSE) is a joint-stock company established in 1991 by entities representing the State Treasury, including the minister of ownership transformation and the minister of finance, which signed the founding act of the Warsaw Stock Exchange. The trading system of the WSE is based on matching buy and sell orders for financial instruments, and the transaction takes place during stock exchange sessions. Within the capital group, the turnover is carried out on the following markets: Main Market Of the WSE (trading in shares and other equity instruments and derivatives), NewConnect (alternative trading system, unregulated trading – trading in shares and other instruments with share specifics – for small and medium companies), Catalyst, Treasury Bondspot Poland. In 2007 the Warsaw Stock Exchange launched the NewConnect market. It is an organized market run by the stock exchange outside the regulated market in the formula of an alternative trading system. The task of this market is to finance the development of small and medium-sized enterprises with high growth potential (for example: from the electronic media sector), short- lived on the market, with a relatively low predictable capitalization, looking for relatively small capital. Investments, which involve the expectation of above-average profits, are simultaneously characterized by increased risk. On NewConnect, more liberal form and information requirements apply to the stock market. The requirements for companies submitting their debut on NewConnect are small compared to the Main Market. NewConnect enables companies to acquire new, additional and necessary capital relatively quickly. The debut on NewConnect is possible through private emission and public broadcast.

**KEYWORDS:** NewConnect, The Warsaw Stock Exchange, Alternative Trading System, market financing, joint stock company, small and medium-sized enterprises, high-tech sector, innovative sector

## 1. Wstępne informacje

Alternatywny system obrotu (ASO) oznacza platformę służącą finansowaniu rozwoju małych i średnich przedsiębiorstw o wysokim potencjale wzrostu. System jest adresowany głównie do przedsiębiorstw, które

1. Wrocławska Wyższa Szkoła Informatyki Stosowanej, ul. Księdza Marcina Lutra 4, 54-239 Wrocław E-mail: marcinzdanowicz@02.pl

działają krótko biznesie lub dopiero rozwijają działalność [1]. Przedmiotem obrotu w ASO mogą być zdematerializowane akcje, prawa do akcji (PDA), prawa poboru, kwity depozytowe, oraz inne udziałowe papiery wartościowe emitowane na podstawie odpowiednich przepisów prawa polskiego lub obcego wprowadzone do tego obrotu. Alternatywny system obrotu jest prowadzony przez Giełdę Papierów Wartościowych w Warszawie (Warszawska Giełda Papierów Wartościowych jest jedną z największych giełd w Europie Środkowo-Centralnej) i ma status rynku nieregulowanego, Platforma jest wspólnym projektem Giełdy i Ministerstwa Gospodarki, W maju 2007 roku Komisja Nadzoru Finansowego zatwierdziła regulamin alternatywnego systemu obrotu — nowego rynku giełdowego, który zaczął obowiązywać w czerwcu 2007 roku. Alternatywny system obrotu nosi nazwę NewConnect — rynek akcji Giełdy Papierów Wartościowych. Alternatywny system obrotu jest przewidziany dla podmiotów zamierzających pozyskać kapitał na rynku publicznym, lub w drodze zamkniętej emisji skierowanej do określonych inwestorów. Platforma jest przeznaczona dla spółek o małej i średniej kapitalizacji. Funkcją Alternatywnego system obrotu jest zbudowanie alternatywnej wobec kredytu możliwości finansowania dla małych, innowacyjnych podmiotów o znacznych perspektywach rozwoju i krótkiej historii funkcjonowania, czyli tzw. start-upów. ASO to szansa dla małych spółek, którym trudno byłoby wejść na klasyczny parkiet [2]. Alternatywny system obrotu można polecić przede wszystkim przedsiębiorstwom, które posiadają dobry biznes plan, korzystają z doradców, ale borykają się z brakiem środków finansowych. Nowy rynek ma pozwolić takim firmom na pozyskanie pieniędzy na finansowanie innowacyjnych działań i rozwój technologiczny, a w perspektywie pomóc zwiększyć rentowność i poprawić wyniki finansowe. Gdy przedsiębiorstwo urośnie, jego właściciele będą mogli wnioskować o wprowadzenie akcji do obrotu na rynku podstawowym Giełdy Papierów Wartościowych w Warszawie. Korzyściami wejścia na nowo powstały rynek dla spółki są prostsze procedury i niższe wymogi formalne w porównaniu z rynkiem regulowanym (rynki podstawowy i równoległy GPW), szybszy dostęp do kapitału. Upublicznienie spółki to również profity dla właścicieli firmy takie jak rynkowa wycena akcji, oraz pozyskanie, jeżeli zakłada to strategia, inwestora strategicznego.

## 2. Funkcjonowanie NewConnect

W dniu 30 sierpnia 2007 roku ruszył prowadzony przez Giełdę Papierów Wartościowych w Warszawie S.A. rynek finansujący działalność i funkcjonowanie małych i średnich firm o szerokich możliwościach wzrostu (NewConnect).
Jest to ważne narzędzie GPW dla inwestorów akceptujących wysokie ryzyko w zamian za możliwe wysokie zwroty z inwestycji. NewConnect posiada status rynku zorganizowanego, ale prowadzony jest przez GPW poza rynkiem regulowanym w formule alternatywnego systemu obrotu. To propozycja dla przedsiębiorstw z różnych sektorów, w tym funkcjonujących w sferze innowacyjnych technologii. NewConnect został zbudowany z myślą o prężnych przedsiębiorstwach, którym zastrzyk finansowy pozwoli wykorzystać ich innowacyjny potencjał. NewConnect stanowi rynek dla spółek z wizją rozwoju, o znacznej dynamice wzrostu, poszukujących kapitału w wysokości od kilkuset tysięcy do kilkudziesięciu milionów złotych, reprezentujących branże innowacyjne, bazujące głównie na aktywach niematerialnych (na przykład: IT – technologia informacyjna, elektroniczne środki masowego przekazu, telekomunikacja, ochrona środowiska, energia alternatywna, nowoczesne usługi, biotechnologie). NewConnect utożsamiany jest z nowoczesnym rynkiem finansującym dynamiczne firmy, mniejszymi niż na rynku regulowanym wymaganiami formalnymi (dopuszczeniowe oraz wynikające z obowiązków informacyjnych), relatywnie niskimi kosztami debiutu i notowań, znakomitą ekspozycją przedsiębiorstwa, prestiżem i renomą organizatora rynku, promocją i rozpoznawalnością firmy. NewConnect to rynek akcji bazujący na alternatywnym systemie obrotu prowadzonym przez Giełdę Papierów Wartościowych w Warszawie SA. Rynek NewConnect opiera swoje działanie i funkcjonowanie na infrastrukturze technologicznej i technicznej GPW [4].

Rynek NewConnect został zorganizowany w myślą o spółkach:

1. w początkowej fazie rozwoju:
2. poszukujących kapitału od kilkuset tysięcy złotych do kilkudziesięciu milionów złotych;
3. reprezentujących sektory o znacznym udziale w aktywach wartości niematerialnych;
4. o dużej dynamice rozwoju;
5. pragnących w przyszłości zadebiutować na rynku regulowanym.

Debiut na NewConnect możliwy jest za pomocą emisji prywatnej (tzw. private placement), czyli oferty zamkniętej adresowanej do co najwyżej 99 podmiotów — inwestorów instytucjonalnych i prywatnych. Niezależnie od wielkości emisji dopuszczenie do obrotu odbywa się na podstawie krótkiego i prostego dokumentu informacyjnego opracowanego przez Autoryzowanego Doradcę we współpracy z emitentem i zaakceptowanego wyłącznie przez Autoryzowanego Doradcę. Drugą z dróg zadebiutowania na NewConnect jest emisja publiczna. Przedsiębiorstwo decydujące się na tę formę emisji podlega takim samym procedurom dopuszczeniowym, jakie wykorzystywane są dla rynku regulowanego, wraz z koniecznością przygotowania prospektu emisyjnego i zatwierdzenia go przez Komisję Nadzoru Finansowego. Dla ofert wynoszących co najwyżej 2,5 mln zł rolę dokumentu dopuszczeniowego może pełnić memorandum informacyjne, które podlega kontroli Komisji Nadzoru Finansowego [6].

Harmonogram wejścia na rynek NewConnect w systemie private placement i systemie oferty publicznej można przedstawić w sposób następujący [7]:

**I. Oferta prywatna** (do maksymalnie 99 inwestorów)

1. decyzja właścicieli spółki o wejściu na NewConnect (NC);
2. przekształcenie firmy w spółkę akcyjną (jeżeli działa w innej formie prawnej);
3. uchwała Walnego Zgromadzenia Akcjonariuszy (WZA) o wprowadzeniu akcji do obrotu na NC w drodze oferty prywatnej;
4. wybór Autoryzowanego Doradcy;
5. sporządzenie dokumentu informacyjnego;
6. zatwierdzenie dokumentu informacyjnego przez Autoryzowanego Doradcę;
7. rejestracja akcji w Krajowym Depozycie Papierów Wartościowych (KDPW);
8. złożenie do Zarządu Giełdy Papierów Wartościowych wniosku o wprowadzenie akcji do obrotu na rynku NewConnect, załączając zatwierdzony przez Autoryzowanego Doradcę dokument informacyjny;
9. GPW zatwierdza dokument informacyjny i decyduje o wprowadzeniu akcji do obrotu;
10. pierwsze notowanie na NewConnect;

**II. Oferta publiczna** (do 100 i więcej inwestorów):

1. decyzja właścicieli spółki o wejściu na NewConnect;
2. przekształcenie firmy w spółkę akcyjną (jeżeli działa w innej formie prawnej);
3. uchwała Walnego Zgromadzenia Akcjonariuszy (WZA) o wprowadzeniu akcji do obrotu na NC w drodze oferty publicznej i ewentualnej nowej emisji akcji;
4. budowa zespołu doradców, wybór domu maklerskiego:
5. przygotowanie publicznego dokumentu informacyjnego (prospekt lub memorandum informacyjne);
6. zatwierdzenie dokumentu informacyjnego przez Komisją Nadzoru Finansowego (lub inny europejski organ nadzoru) – 20 dni roboczych;
7. rejestracja akcji w Krajowym Depozycie Papierów Wartościowych (KDPW);
8. złożenie do Zarządu Giełdy Papierów Wartościowych wniosku o wprowadzenie akcji do obrotu na rynku NewConnect, załączając zatwierdzony przez Autoryzowanego Doradcę dokument informacyjny;
9. GPW zatwierdza dokument informacyjny i decyduje o wprowadzeniu akcji do obrotu;
10. pierwsze notowanie na NewConnect.

Jeżeli spółka zdecyduje się na wprowadzenie akcji poprzez ofertę prywatną możliwość, a później gotowość spółki do wprowadzenia na NewConnect ocenia niezależna firma doradcza pełniąca rolę tzw. Autoryzowanego Doradcy (z którym spółka zawiera umowę o współpracy). Jej zadaniem jest wspieranie spółki w procesie przygotowań do debiutu oraz kooperacja przez co najmniej rok funkcjonowania na NewConnect. Rolę autoryzowanego doradcy mogą pełnić przedsiębiorstwa inwestycyjne, kancelarie prawne, firmy audytorskie firmy doradztwa finansowego. O tym, kto ma status Autoryzowanego Doradcy, decyduje Giełda Papierów Wartościowych (prowadzi ona także ich spis). Autoryzowany Doradca dla spółki sporządza dokument informacyjny lub sprawdza, czy został on przygotowany zgodnie z wymogami obowiązującymi na NewConnect. Dokument ten (opracowany w języku polskim lub angielskim) powinien zawierać autentyczne, rzetelne i całościowe dane o emitencie i emisji. Powinien on się składać ze wstępu (nazwa spółki, rodzaj akcji, spis treści),
oraz z sześciu rozdziałów zawierających kolejno: czynniki ryzyka, dane o osobach odpowiedzialnych za informacje w dokumencie informacyjnym, dane o akcjach wprowadzonych do obrotu, dane o emitencie (między innymi: historię spółki, informację o jej kapitałach, produktach, usługach i strategii), sprawozdania finansowe, oraz załączniki (odpis z Krajowego Rejestru Sądowego, statut). Autoryzowany Doradca bierze odpowiedzialność za jakość przygotowanego dokumentu, składając oświadczenie, że zamieszczone w nim informacje są prawdziwe. W sytuacji gdy wejście na NewConnect spółki poprzedza prywatna emisja akcji, doradca pomaga jej przeprowadzić emisję (organizuje spotkania z inwestorami, prowadzi plasowanie). Może także sam zakupić akcje spółki. Po wprowadzeniu do obrotu na NewConnect doradza spółce (co najmniej przez rok) i pomaga jej w wypełnianiu obowiązków informacyjnych [8].
Spółka wchodząca na rynek NewConnect musi się liczyć z kosztami. Są one jednak mniejsze niż przy wchodzeniu na Główny Rynek GPW. Pierwszy wydatek może się wiązać z przekształceniem spółki w spółkę akcyjną. Wszystkie sprawozdania finansowe przed publikacją w dokumencie informacyjnym muszą zostać poddane odpłatnemu audytowi. Następny koszt wiąże się z umową z Autoryzowanym Doradcą umowa jest zawierana na co najmniej rok i przez cały ten czas spółka ponosi koszty związane z tą umową. Kolejne koszty to te związane z nową emisją akcji. Szczególnie znaczne mogą być koszty audytu sprawozdań finansowych oraz sporządzenia dokumentu informacyjnego. Spółka ponosi również koszty związane z uczestnictwem w Krajowym Depozycie Papierów Wartościowych. Umowa z animatorem rynku to następny koszt, który należy wziąć pod uwagę. Umowa jest zawierana na dwa lata i przez ten okres należy liczyć się z kosztami. Koszty notowań przedstawione są w załączniku do regulaminu alternatywnego systemu obrotu. Dokument ten oprócz stawek ukazuje terminy i sposób pobierania opłat od uczestników rynku [10].
Notowania na rynku NewConnect odbywają się w systemie kierowanym zleceniami z udziałem animatora rynku lub cenami z udziałem market makera (wyboru systemu dokonuje spółka). Funkcję animatora rynku i market makera może sprawować dom maklerski. Ich zadaniem jest głównie podtrzymywanie płynności obrotu. Animator ryn-

ku zobowiązuje się do dokonywania na własny rachunek na rynku kierowanym zleceniami czynności mających na celu wspomaganie płynności obrotu instrumentami finansowymi określonego emitenta (bierze udział w handlu, aby go ożywić). Z kolei market maker zobowiązuje się do składania na rynku kierowanym cenami ofert kupna i sprzedaży. Obaj funkcjonują na zasadach ukazanych przez organizatora Alternatywnego Systemu Obrotu. czyli Giełdę Papierów Wartościowych w Warszawie [11].

Ważnym indeksem charakteryzującym rynek NewConnect jest NClndex wprowadzony 30 sierpnia 2007 roku z ustaloną pierwszą wartością 100 pkt. Sposób jego liczenia zmieniał się wraz ze wzrostem liczby notowanych spółek, obejmując wszystkie podmioty wprowadzone do obrotu na tym rynku. W czerwcu 2013 roku Giełda Papierów Wartościowych wprowadziła zmianę w sposobie liczenia głównego indeksu NClndex, rozpoczęła prezentować nowy indeks: NClndex30, pokazujący 30 najbardziej płynnych spółek [12].

W Tabeli 1 przedstawiono dziesięć spółek o najbardziej płynnych akcjach na dzień 28 czerwca 2013 roku, Największym wskaźnikiem udziału w obrotach oznaczała się spółka MEDICALG S A (10,91%). Na drugiej pozycji uplasowała się spółka ZWG S.A. (9,76%) a trzecią lokatę zajęła spółka PGSSOFT S.A. (8,07%).

| L.p. | Nazwa spółki | Wskaźnik udziału w obrotach (w %) |
|---|---|---|
| 1. | MEDICALG S.A. | 10,91 |
| 2. | ZWG S.A. | 9,76 |
| 3. | PGSSOFT S.A. | 8,07 |
| 4. | PLASMA S.A. | 7,96 |
| 5. | LUG S.A. | 6,52 |
| 6. | 11BIT S.A. | 6,18 |
| 7. | WINDMOBIL S.A. | 4,33 |
| 8. | CSY S.A. | 4,19 |
| 9. | PRYMUS S.A. | 3,88 |
| 10. | UBOATLINE S.A. | 3,43 |

Tabela 1. Udział 10 największych spółek w NClndex30 na dzień 28 czerwca 2013 (w %) Źródło: [13]

Rynek NewConnect jest rynkiem dużym, liczącym 413 spółek, których kapitalizacja wynosi około 9 mld zł (dane z 31 sierpnia 2012 roku). Rynek ten szybko rośnie, średnio raz w tygodniu pojawia się na nim kilka nowych spółek. W 2011 roku przybyły 172 nowe spółki. Na rynku NewConnect oprócz spółek polskich jest notowanych kilka spółek zagranicznych. Większość notowanych spółek to spółki małe. Spółki z NewConnect są dzielone według następujących sektorów: BDU (budownictwo i materiały budowlane), EEN (energia odnawialna), EHA (handel hurtowy i detaliczny z wykorzystaniem platform elektronicznych), HAN (tradycyjny handel hurtowy i detaliczny), IFT (informatyka i działalności pokrewne), INW (działalność brokerska, fundusze inwestycyjne, inwestycje finansowe), MDA (media tradycyjne i elektroniczne), NCH (budownictwo ogólne i inżynieria lądowa), REC (odzyskiwanie i utylizacja substancji lub materiałów), TEC (elektronika, przemysł elektromaszynowy), TLK (telekomunikacja), UFI (doradztwo finansowe, UNN (usługi inne niesklasyfikowane), WYP (restauracje, hotele, turystyka), ZDR (produkcja farmaceutyków, przychodnie medyczne). W 2011 roku przeciętnie na sesji realizowano około tys. transakcji. Rynek odznacza się małą płynnością, co powoduje dużą zmienność kursów. Kursy podatne są na manipulacje i przypadkowe wartości. W wolnym obrocie znajduje się około 25% akcji. Większość akcji znajduje się w rękach dużych inwestorów. Przeważnie nie są to inwestorzy instytucjonalni, a najczęściej są to spółki rodzinne lub założone przez zaprzyjaźnione osoby fizyczne. W wielu przypadkach inwestorzy ci gromadzą akcje uprzywilejowane co do głosu. Udział inwestorów indywidualnych na rynku NewConnect oceniany jest na 92%. Spółki z rynku NewConnect mają relatywnie wysokie współczynniki fundamentalne. Na koniec 2011 roku średni współczynnik CIZ (relacja kapitalizacji giełdową i sumy zysków netto za ostatnie cztery kwartały) wynosił około 57, a średni współczynnik CIWK (relacja kapitalizacji giełdowej i wartości kapitałów własnych na koniec ostatniego kwartału kalendarzowego) wynosił 6 [14].

W celu ucywilizowania stosunków między spółkami i inwestorami GPW przygotowała zbiór zasad właściwego postępowania spółek. Na szczególną uwagę zasługują dwa dokumenty: *Dobre praktyki spółek notowanych na NewConnect* oraz *Dobre praktyki autoryzowanych doradców NewConnect.* Stosowanie zasad ładu korporacyjnego (dobrych praktyk) nie jest obowiązkowe, ale obowiązkowe jest poinformowanie o ich przyjęciu inwestorów. Każda spółka musi przekazać w formie komunikatu czy będzie stosować reguły ładu korporacyjnego. Niektóre spółki przyjęły te zasady częściowo, informując, które zasady zostały wykluczone. *Dobre praktyki spółek notowanych na NewConnect* podkreślają wagę informacji przekazywanych inwestorom. Polecają korzystanie nie tylko z klasycznych metod, ale także z nowoczesnych technologii. Zalecane jest podawanie na własnej stronie internetowej konkretnych informacji takich jak główne informacje o spółce i jej działalności, opis działalności, opis branży, do której spółka należy i jej

pozycja na rynku, dokumenty korporacyjne, zarys planów strategicznych, struktura akcjonariatu oraz liczba akcji w wolnym obrocie, raporty bieżące i okresowe, informacje o istotnych wydarzeniach korporacyjnych. *Dobre praktyki spółek notowanych na NewConnect* regulują również, co powinien zawierać raport półroczny spółki: bilans, rachunek zysków i strat, porównywalne dane za półrocze roku poprzedniego, charakterystykę czynników mających wpływ na wyniki finansowe, informacje o działalności spółki w zakresie badań i rozwoju, w szczególności o pozyskiwaniu licencji i patentów. *Dobre praktyki autoryzowanych doradców NewConnect* omawiają z kolei zasady prawidłowego postępowania autoryzowanych doradców [15]. Podsumowując, stwierdzić należy, że alternatywny rynek akcji jest potrzebnym segmentem polskiego rynku kapitałowego. Małym i średnim spółkom daje możliwość pozyskania kapitału finansującego ich potrzeby rozwojowe. Może być etapem przejściowym dla dynamicznie działających przedsiębiorstw o dobrej pozycji finansowej. Część spółek z rynku NewConnect przenosi się na Główny Rynek. Pomimo że uczestnictwo na Giełdzie Papierów Wartościowych w Warszawie wiąże się z bardziej restrykcyjnymi wymaganiami związanymi z polityką informacyjną i większymi kosztami wejścia na główny parkiet oraz bieżącej obsługi, to pojawienie się w gronie emitentów powinno być dla każdej firmy wyróżnieniem. Spółka staje się bardziej wiarygodnym partnerem biznesowym, a to może być docenione przez inwestorów poszukujących małych i perspektywicznych przedsiębiorstw.

## 3. Wnioski

Wzorem kilku giełd zagranicznych (rynki alternatywne o specyfice zbliżonej do NewConnect powstały między innymi na giełdzie w Londynie – AIM, Sztokholmie – OMX) w sierpniu 2007 roku na giełdzie warszawskiej utworzono nową platformę obrotu akcjami noszącą nazwę NewConnect. Do obrotu na tej platformie są dopuszczone akcje spółek mniejszych rozmiarów, niespełniające standardów obowiązujących na Głównym Rynku. Celem Zarządu Giełdy było rozszerzenie zakresu działalności giełdy poprzez dopuszczenie na nią dynamicznych firm mniejszych rozmiarów. Dla mniejszych spółek wejście na NewConnect ma istotne pozytywne skutki, między innymi: zwiększa prestiż, dzięki czemu ułatwia sprzedaż produktów przedsiębiorstwa; zostaje zwiększona rozpoznawalność firmy; obecność na NewConnect ułatwia pozyskanie klientów. NewConnect jest rynkiem alternatywnym, czyli nie obowiązuje na nim wiele przepisów normujących rynek regulowany. Organizację tego rynku ukazuje Regulamin alternatywnego systemu obrotu określany przez władze giełdy i zatwierdzany przez Komisję Nadzoru Finansowego. Wejście na NewConnect wymaga spełnienia określonych warunków. Są to posiadanie statusu spółki akcyjnej, nieograniczona zbywalność akcji, korzystanie z pomocy autoryzowanego doradcy i animatora rynku (lub market makera), przygotowanie dokumentu dopuszczeniowego, Istotną rolę na rynku NewConnect pełnią wymogi informacyjne stawiane wprowadzanym na ten rynek spółkom. W praktyce można mówić o dwóch sposobach dopuszczenia spółki do debiutu na NewConnect: poprzez emisję niepubliczną (prywatną) oraz poprzez emisję publiczną akcji. Analogicznie jak na Głównym Rynku spółki notowane na NewConnect są zobligowane do publikowania raportów okresowych i raportów bieżących. Spółki notowane na NewConnect to w większości firmy niewielkich rozmiarów, o dużych ambicjach, zamierzające szybko zdobyć kapitały. Część spółek notowanych na NewConnect na tyle wzmocniła się kapitałowo, że zaczęła spełniać kryteria obowiązujące na Głównym Rynku giełdy, co otworzyło im drogę do przejścia na ten rynek. Powyższa sytuacja może być odbierana jako istotny sukces NewConnect.

## Literatura:

[1] S. Antkiewicz, M. Kalinowski (red.), I*nnowacje finansowe*, Warszawa 2008, s. 93.
[2] *Bankowość inwestycyjna, inwestorzy, banki i firmy inwestycyjne na rynku finansowym*, red. P. Niedziółka, Warszawa 2015. s. 99.
[3] D. Kordela, *NewConnect – rynek giełdowy dla małych i średnich przedsiębiorstw: systematyka, organizacja, perspektywy rozwoju*, Warszawa 2013, s. 83.
[4] Z. Dobosiewicz, *Giełda: zasady działania, inwestorzy, rynki giełdowe*, Warszawa 2013, s. 177.
[5] W. Popczyk, *Przedsiębiorstwa rodzinne w otoczeniu globalnym: analiza porównawcza ekspansji międzynarodowej firm rodzinnych i nierodzinnych z rynku NewConnect*, Łódź 2013, s. 167.
[6] R. Pastusiak, *Przedsiębiorstwo na rynku kapitałowym: operacje giełdowe rynku publicznego i niepublicznego*, Warszawa 2010, s. 377.
[7] M. Adamska, *Droga na giełdę: jak przygotować spółkę do emisji publicznej*, Warszawa 2008, s. 462.
[8] W. Dębski, *Rynek finansowy i jego mechanizmy: podstawy teorii i praktyki,* Warszawa 2014, s. 176.
[9] T. Maliński, *Giełda papierów wartościowych dla bystrzaków*, Gliwice 2015, s. 29.
[10] *Źródła finansowania działalności a sprawność przedsiębiorstw działających w Polsce*, red. D. Ostrowska, Warszawa 2014, s. 225
[11] M. A. Kaber, *Współczesne instytucje rynku finansowego*, Białystok 2014, s. 79.
[12] *Uwarunkowania rynkowe rozwoju mikro, małych i średnich przedsiębiorstw. Mikrofirma*, red. A. Bielawska, G. Rosa, Szczecin 2011, s. 203.
[13] B. Kołosowska, *Finansowanie sektora małych i średnich przedsiębiorstw ze źródeł pozabankowych*, Warszawa 2013, s. 139.
[14] A. Jagielnicki, *NewConnect: nowa szansa na duże zyski*, Gliwice 2013, s. 53, 73.
[15] P. Zygmanowski, *Determinanty rozwoju rynku akcji*

# Sztuczne sieci neuronowe ANN. Sieci Kohonena

*Artificial neural networks (ANN). Kohonen networks*

**Paweł Iljaszewicz**[1]

**STRESZCZENIE:** Artykuł omawia sztuczne sieci neuronowe (*ang. ANN- Artificial neural networks*). Jedną z odmian są sieci Kohonena zwane Mapą Samoorganizującą (*ang. SOM – Self Organizing Map*) realizują one proces uczenia się sieci neuronowych samodzielnie tzn. rozpoznają relacje występujące w skupieniach poprzez wykrycie wewnętrznej struktury i kategoryzują je w procesie samouczenia. SOM służy do uformowania odwzorowania z przestrzeni wielowymiarowej do przestrzeni jednowymiarowej lub dwuwymiarowej. Główną cechą SOM jest to, że tworzy on nieliniową projekcję wielowymiarową kolektora danych na regularnej, niskowymiarowej (zwykle 2D) sieci. Na wyświetlaczu klastrowanie przestrzeni danych, jak również relacje metryczno-topologiczne elementów danych, są wyraźnie widoczne. Jeśli elementy danych są wektorami, składniki, których są zmiennymi z określone znaczenie, takie jak deskryptory danych statystycznych lub pomiary, które opisują proces, siatka SOM może być wykorzystana, jako podstawa, na której może znajdować się każda zmienna wyświetlane osobno przy użyciu kodowania na poziomie szarości lub pseudo koloru. Ten rodzaj projekcji został uznany za bardzo przydatny do zrozumienia wzajemnych zależności między zmiennymi, a także strukturami zbioru danych.

**SŁOWA KLUCZOWE:** Sieci Kohonena, sieci neuronowe, Mapa samoorganizująca (SOM), WEBSOM

**SUMMARY:** The article discusses artificial neural networks (ANN). One of the varieties is the Kohonen network, called the Self Organizing Map (SOM), that perform the learning process of neural networks independently, i.e. they recognize relationships occurring in clusters by detecting an internal structure and categorizing them in the process of self-learning. SOM is used to form mapping from a multidimensional space to a one-dimensional or two-dimensional space. The main feature of SOM is that it creates a non-linear multi-dimensional projection of a data collector on a regular, low-dimensional (usually 2D) network. On the display, data space clustering as well as metric-topological relations of data elements are clearly visible. If the data elements are vectors, the components of which are variables with defined meanings, such as statistical data descriptors or measurements that describe the process, the SOM grid can be used as a basis on which each variable can be displayed separately using gray or pseudo-color coding. This type of projection has been found to be very useful for understanding the interrelationships between variables as well as data set structures.

**KEYWORDS:** Kohonen networks, artificial neural networks (ANN), Self Organizing Map SOM, WEBSOM.

## 1. Wprowadzenie

Architektury sieci i procesy sygnałowe używane do modelowych układów nerwowych można podzielić na trzy kategorie, każda jest oparta na innej filozofii. Sieci wyprzedzające [1] (*ang. Feedforward networks*) przekształcają zbiory sygnałów wejściowych w zestawy sygnałów wyjściowych. Pożądana transformacja wejścia-wyjścia jest zwykle określany przez zewnętrzną, nadzorowaną regulację parametrów systemu. W sieciach sprzężeń zwrotnych [2] (*ang. feedback network*) informacja wejściowa określa początkowy stan działania układu sprzężenia zwrotnego, a po przejściu przez stan asymptotyczny finalny stan jest identyfikowany, jako wynik obliczeń.
W trzeciej kategorii sąsiednie komórki w sieci neuronowej współdziałają poprzez wzajemne oddziaływania boczne i rozwijają się adaptacyjnie w określone detektory o różnych wzorach sygnału. W tej kategorii nazywa się uczenie się konkurencyjne, bez nadzoru ani samoorganizujące się. Sieć Kohenena inaczej samoorganizująca się mapa SOM, należy do kategorii uczenia się konkurencyjnego.
Jest to sztuczna sieć neuronowa przypominająca arkusz, której komórki są specjalnie dostrojone do różnych wzorów sygnałów wejściowych lub klas wzorców przez nienadzorowany proces uczenia się. Tylko w wersji podstawowej jedna cela lub lokalna grupa komórek na raz daje aktywną odpowiedź na bieżące dane wejściowe. Taki sposób działania nazywamy uczeniem się konkurencyjne, bez

1. Instytut Technologiczno-Przyrodniczy, Falenty, al. Hrabska 3, 05-090 Raszyn Email: p.iljaszewicz@itp.edu.pl

nadzoru, albo samoorganizującym się. Model ten został przedstawiony przez fińskiego naukowca T. Kohonena. Algorytm samouczenia odpowiednio modyfikuje wagowe współczynniki w neuronach składających się na mapę topologiczną, aby zbliżyć je do skupień na podstawie danych uczących [3]. Mapa samoorganizująca się miała być realną alternatywą dla bardziej tradycyjnych architektur sieci neuronowych. Można zapytać, jak "neuron" może być mapą. Jego analityczny opis został już opracowany, bardziej w technicznym niż biologicznym kierunku. Ale osiągane wyniki uczenia się wydają się bardzo naturalne, co najmniej wskazując, że same procesy adaptacyjne w pracy na mapie mogą być podobne do spotykanych w mózgu. Może więc istnieć wystarczające uzasadnienie dla kalibrowania tych map "neuronowych sieci" w takim samym znaczeniu, jak ich tradycyjni odpowiedniki.

Mapy samoorganizujące się lub systemy składające się z kilku modułów map, zostały użyte do zadań podobnych do tych, jakie są stosowane przez tradycyjne sieci neuronowe: rozpoznawanie wzorów, robotyka, sterowanie procesem i nawet przetwarzanie informacji semantycznych. Przestrzenna segregacja różnych odpowiedzi i ich organizacja w topologicznie powiązane podzbiory powodują wysoki stopień wydajności w typowych operacjach sieci neuronowych.

Kluczowym wynalazkiem Kohonena było wprowadzenie modelu systemu, który składa się z co najmniej dwóch oddziałujących ze sobą podsystemów o różnym charakterze. Jednym z tych podsystemów jest konkurencyjna sieć neuronowa, która implementuje funkcję zwycięzcy-bierze-wszystko (*ang. winner-take-all function*), ale jest też inny podsystem, który jest kontrolowany przez sieć neuronową i który modyfikuje lokalną synaptyczną plastyczność neuronów w uczeniu się. Nauka jest ograniczona przestrzennie do lokalnego sąsiedztwa najbardziej aktywnych neuronów. Podsystem kontroli plastyczności mógłby opierać się na niespecyficznych interakcjach neuronalnych, ale prawdopodobnie jest to efekt kontroli chemicznej. Tylko dzięki rozdzieleniu przekazywania sygnału neuronowego i kontroli plastyczności stało się możliwe wdrożenie skutecznego i niezawodnego systemu samoorganizującego się.

SOM jest algorytmem służącym do wizualizacji i interpretacji dużych zestawów danych wysokomiarowych. Typowe aplikacje to wizualizacja stanów procesu lub wyników finansowych poprzez przedstawienie centralnych zależności w obrębie danych na mapie.

Mapa składa się z regularnej siatki jednostek przetwarzania "neuronów". Model jakiejś wielowymiarowej obserwacji to wynikowy wektor składający się z funkcji, który jest powiązany z każdą jednostką. Mapa stara się reprezentować wszystkie dostępne obserwacje z optymalną dokładnością przy użyciu ograniczonego zestawu modeli. W tym samym czasie modele stają się uporządkowane na siatce, tak że podobne modele są blisko siebie, a modele odmienne od siebie oddalone.

Niemniej jednak, zasada SOM może być również wyrażona matematycznie w czystej abstrakcyjnej formie, bez odniesienia do jakiegokolwiek leżącego u podstaw neuronowego lub innych składników.

## 2. Matematyczne określenie algorytmu Kohena

Rozważmy pierwsze elementy danych, które są wektorami w n-wymiarowej przestrzeni euklidesowej.

$$x(t) = [\xi 1(t), \xi 2(t), \ldots, \xi n(t)]$$

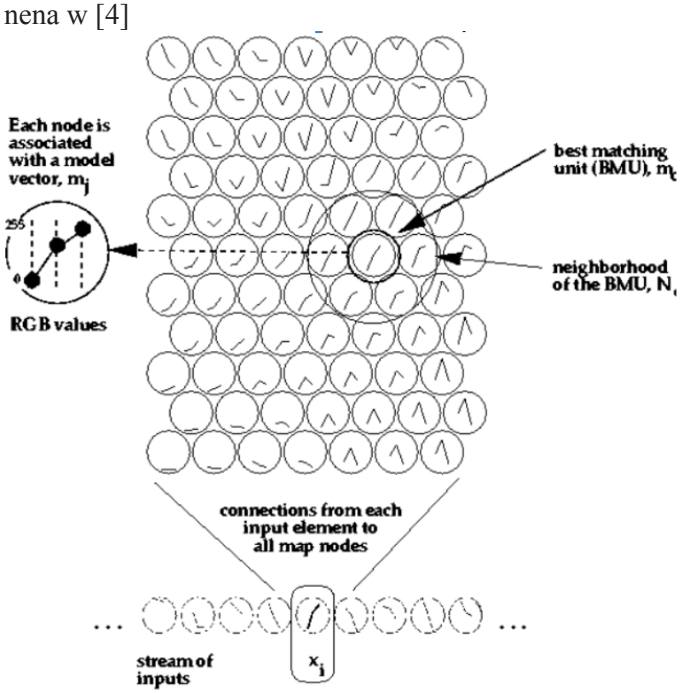
Gdzie t jest indeksem pozycji danych w danej sekwencji. Niech tym modelem będzie i :,

$$m_i(t) = [\mu_{i1}(t), \mu_{i2}(t), \ldots, \mu_{in}(t)],$$

gdzie teraz *t* oznacza indeks w sekwencji, w której generowane są modele. Ta sekwencja jest zdefiniowana, jako proces typu wygładzania, w którym nowa wartość $m_i(t+1)$ jest obliczana iteracyjnie ze starej wartości $m_i(t)$ i nowej pozycji danych $x(t)$, jako:

$$m_i(t+1) = m_i(t) + \alpha(t) h_{ci}(t)[x(t) - m_i(t)]$$

Gdzie $\alpha(t)$ jest czynnikiem skalarnym, który definiuje wielkość korekty; jego wartość maleje wraz ze wskaźnikiem kroku *t*. Indeks i odnosi się do modelu w trakcie przetwarzania, a *c* jest indeksem modelu, który ma najmniejszą odległość od $x(t)$ w przestrzeni euklidesowej sygnałowej. Czynnik $h_{ci}(t)$ to rodzaj wygładzającego jądra, zwany także funkcją sąsiedztwa. Funkcja ta jest równa 1, gdy $i=c$, a jego wartość maleje, gdy wzrasta odległość między wzorami $m_i$ i $m_c$ na siatce. Ponadto, szerokość przestrzenna jądra na siatce zmniejsza się wraz ze wskaźnikiem kroku *t*. Te funkcje wskaźnika kroku, które określają zbieżność, należy wybrać [2] bardzo delikatnie. Również inicjalizacja $m_i(1)$ jest problemem omówionym dokładnie przez Kohonena w [4]




Ryc. 1 Podstawowa architektura mapy samoorganizującej się (SOM). Wejście x jest w pełni połączone z tablicą węzłów map, która najczę-

2. Poruszono dokładnie w [3].

ściej jest dwuwymiarowa. Każdy węzeł mapy, zwizualizowany, jako krąg na siatce służy jako model mi lub do użycia innego terminu, prototypu klasy podobnych wejść. Diagramy liniowe wewnątrz kółek oznaczają trzy wartości RGB. Na przykład węzły w lewym dolnym rogu odpowiadają kolorom, które mają wysokie wartości wszystkich komponentów, tj. czerwony, zielony i niebieski (RGB), a zatem ten narożnik zawiera bardzo ciemne kolory[3]

Na rysunku 1 przedstawiono podstawową ideę w procesie uczenia algorytmu SOM. Dla każdego wektora wejściowego próbki *x(t)* zwycięzca i węzły w jego sąsiedztwie zmieniają się bliższe *x(t)* w wejściowej przestrzeni danych. Podczas procesu uczenia się poszczególne zmiany mogą być sprzeczne, ale końcowym wynikiem sieciowym w procesie jest to, że uporządkowane wartości dla $m_i(t)$ pojawią się na tablicy macierzy.
Jeśli liczba dostępnych próbek wejściowych jest ograniczona, próbki muszą być przedstawione w sposób powtarzalny algorytmowi SOM [5].
Chociaż powyższy algorytm został zastosowany z powodzeniem w wielu aplikacjach, okazało się, że schemat nazywany mapą wsadową (*ang. Batch Map*) daje zasadniczo podobne wyniki, ale o rząd wielkości szybsze. Podstawową ideą jest to, że dla każdego węzła *j* w siatce średnia $\bar{x}_j$ wszystkich elementów wejściowych *x(t)* jest utworzona, a $m_i$ jest najbliższym modelem. Następnie nowe modele są obliczane jako:

$$m_i = \frac{\sum_j n_j h_{ji} \bar{x}_j}{\sum_j n_j h_{ji}}$$

Gdzie $n_j$ to liczba elementów wejściowych odwzorowanych w węźle *j*, a indeks *j* działa nad węzłami w sąsiedztwie węzła *i*. W przypadku zaktualizowanego $m_i$, ten schemat jest powtarzany kilka razy, zawsze przy użyciu tej samej partii elementów danych wejściowych w celu ustalenia zaktualizowanego $\bar{x}_j$.
Matematyczna teoria SOM jest bardzo skomplikowana i tylko przypadek jednowymiarowy został całkowicie przeanalizowany [6]. W przypadku zmodyfikowanej reguły adaptacji, w której dopasowanie jest uśredniane na $h_{ci}$, SOM może pochodzić z konkretnej potencjalnej funkcji [7]. Najwyraźniej SOM należy do problemów „źle postawionych" w matematyce.

## 3. Praktyczne porady mające zastosowanie podczas realizacji SOM

Wskazane jest, aby zwracać uwagę przynajmniej na następujące zalecenia, aby uzyskane mapowania były stabilne, dobrze zorientowane i poprawne topologicznie. [3]

**Forma tablicy**: Sześciokątna siatka węzłów ma być preferowana do kontroli wzrokowej. Krawędzie macierzy powinny być raczej prostokątne niż kwadratowe, ponieważ "elastyczna sieć" utworzona z wektorów modelowych $m_i$ musi być zorientowana wraz z rozmieszczeniem wejścia *x* w przestrzeni sygnałowej. Z drugiej strony, ponieważ $m_i$ musi być w przybliżeniu równe rozkładowi *x*, pożądane jest ponadto, aby szerokość i wysokość tablicy odpowiadały co najmniej głównym wymiarom (na przykład główne elementy składowe) rozkładu *x*.

**Skalowanie komponentów wektorowych** jest bardzo subtelnym problemem. Można łatwo zauważyć, że orientacja lub uporządkowana "regresja" wektorów modelu w przestrzeni wejściowej zależy od skalowania komponentów (lub wymiarów) wektorów danych wejściowych. Zwłaszcza, jeśli elementy danych reprezentują zmienne różnego rodzaju i dlatego są podawane w różnych skalach, nie istnieje żadna prosta reguła określająca, jakiego rodzaju zmiana skali jest najlepsza przed wprowadzeniem danych treningowych do algorytmu uczenia. Można próbować heurystycznie uzasadnionych skalowań i sprawdzać, jakość otrzymanych map np. za pomocą mapowania Sammona [8] lub średniego błędu kwantyzacji. Często użyteczną strategią jest normalizacja wszystkich zmiennych wejściowych takich, że np. wszystkie ich wariancje stają się równe.

**Nauka z niewielką liczbą dostępnych próbek treningowych**: W niektórych problemach, np. podczas tworzenia map dla przedmiotów zdefiniowanych przez skończony zestaw atrybutów, nie ma dużego zestawu próbek treningowych. Jednak dla dobrej dokładności statystycznej proces uczenia może wymagać dość dużej liczby kroków szkoleniowych, a jeśli liczba dostępnych próbek jest znacznie mniejsza, próbki mogą być użyte ponownie w treningu. Próbki mogą być stosowane cyklicznie lub w losowo permutowanej kolejności lub pobierane losowo z podstawowego zestawu przy użyciu minimalnych zasobów (nazywanego uczeniem się bootstrapu). W praktyce okazało się, że uporządkowane cykliczne nakładanie dostępnych próbek nie jest zauważalnie gorsze od innych, statystycznie rygorystycznych metod pobierania próbek.

**Jakość uczenia się**: różne procesy uczenia się będą uzyskiwane dla różnych wartości początkowych $m_i(1)$ i przy stosowaniu różnych sekwencji wektorów treningowych *x(t)* i różnych parametrów uczenia się. Oczywiście byłoby pożądane, aby powstała mapa zawsze była najmniej dwuznaczna.
Oczywiste jest, że musi istnieć jakaś optymalna mapa dla tych samych danych wejściowych. Porównując mapy o tej samej strukturze macierzy i definicji $h_{ci}$, średnia z || x-mc || (błąd kwantyzacji) w danych treningowych jest przydatnym wskaźnikiem wydajności. Dlatego można próbkować z większą liczbą (powiedzmy kilkadziesiąt) losowych ini-

cjalizacji $m_i(1)$ tak, by powstała mapa z najmniejszym błędem kwantyzacji. Zauważmy jednak, że nie miałoby sensu porównywać błędów kwantyzacji dla map o różnych strukturach lub $h_{ci}$, gdyż błąd jest minimalny, jeśli $h_{ci} = \delta_{ci}$ (delta Kroneckera). Dla pojedynczego jądra nie występuje już samoorganizacja, ponieważ algorytm zostanie zredukowany do klasycznej kwantyzacji wektora.

## 4. Implementacja algorytmu SOM

Analizę danych, tworzenie klastrów i wizualizację przez SOM najlepiej przeprowadzić przy użyciu komercyjnego oprogramowania z kodem pobranym z domeny publicznej. Używanie oprogramowania z własnym kodem nie jest zalecane, ponieważ istnieje wiele subtelnych aspektów, które należy wziąć pod uwagę i które wpływają na zbieżność i dokładność algorytmu. SOM_PAK to zestaw narzędzi do prawidłowego zastosowania SOM. Ten oryginalny pakiet programowy został stworzony przez zespół programistów SOM z Uniwersytetu Technicznego w Helsinkach, szczególnie w przypadku bardzo dużych problemów[3]. Więcej informacji o pakietach można znaleźć na stronie SOM_PAK Jussi Hynninena[4].
Poniżej wyszczególniono cztery szczegółowe zastosowania SOM, które stanowią podstawę do dalszej kontroli następujących działów:

### 4.1. SOM jest modelem określonych aspektów biologicznych sieci neuronowych

Być może najbardziej typowym pojęciem SOM jest uznanie go za sztuczny model sieci neuronowej mózgu, szczególnie eksperymentalnie odnalezionych "map" w korze mózgowej. Istnieje wiele dowodów neurofizjologicznych na poparcie idei, że SOM przechwytuje niektóre z podstawowych zasad przetwarzania mózgu. Niektóre wcześniejsze sztuczne modele sieci samoorganizujących się oparte na sieciach neuronowych zostały wcześniej zaprezentowane, np. przez Shun-ichi Amari [9] czy Gail Carpenter i Stephen Grossberg [10]; jednak zasada SOM okazała się najbardziej efektywna, by produkować mapy podobnie jak mózg. Związek pomiędzy SOM i jego odpowiednikami w neurodynamice jest opisany szczegółowo, np. w pracach Kohonena [11, 12].

### 4.2. SOM stanowi reprezentację nowego paradygmatu sztucznej inteligencji i modelowania kognitywnego,

SOM można postrzegać jako maszynowy model uczenia się bez nadzoru oraz jako adaptacyjny schemat reprezentacji wiedzy. W rozdziale 5 rozważany jest związek między SOM (zwłaszcza mapą kategorii słów) a sieciami semantycznymi. Tradycyjne formalizacje reprezentacji wiedzy - sieci semantyczne, systemy ramowe, logika orzecznicza stosowane jako pewne przykłady, są statyczne, a relacje referencyjne elementów są określane przez człowieka. Ponadto formalizmy te opierają się na milczącym założeniu, że relacja między językiem naturalnym a światem jest jeden do jednego: świat składa się z obiektów i relacji między obiektami, a te obiekty i związki mają bezpośrednią korespondencję z elementami języka. W odniesieniu do reprezentacji wiedzy i uczenia się, aspekty poznawcze i filozoficzne są bardzo istotne. Zagadnienia te zostaną omówione bardziej szczegółowo w rozdziale

### 4.3. SOM jest narzędziem do analizy statystycznej i wizualizacji

SOM jest obecnie często używane jako narzędzie statystyczne do analizy wielowymiarowej. SOM jest zarówno metodą rzutowania, która odwzorowuje przestrzenną przestrzeń danych o dużej przestrzeni na przestrzeń niskiego wymiaru, jak i metodą grupowania, aby podobne próbki danych były odwzorowywane na pobliskie neurony. Z metodologicznego i obliczeniowego punktu widzenia można rozważyć matematyczne i statystyczne właściwości algorytmu (na przykład złożoność czasu i przestrzeni, właściwości konwergencji), a także charakter danych wejściowych (sygnały, stałe wskaźniki statystyczne, ciągi symboli) i ich wstępne przetwarzanie. Istnieje wiele wariantów SOM, w których, na przykład, reguły adaptacji są różne, można zastosować różne miary odległości, a struktura połączeń międzysystemowych jest zmienna. SOM jest narzędziem do opracowywania złożonych aplikacji [3].

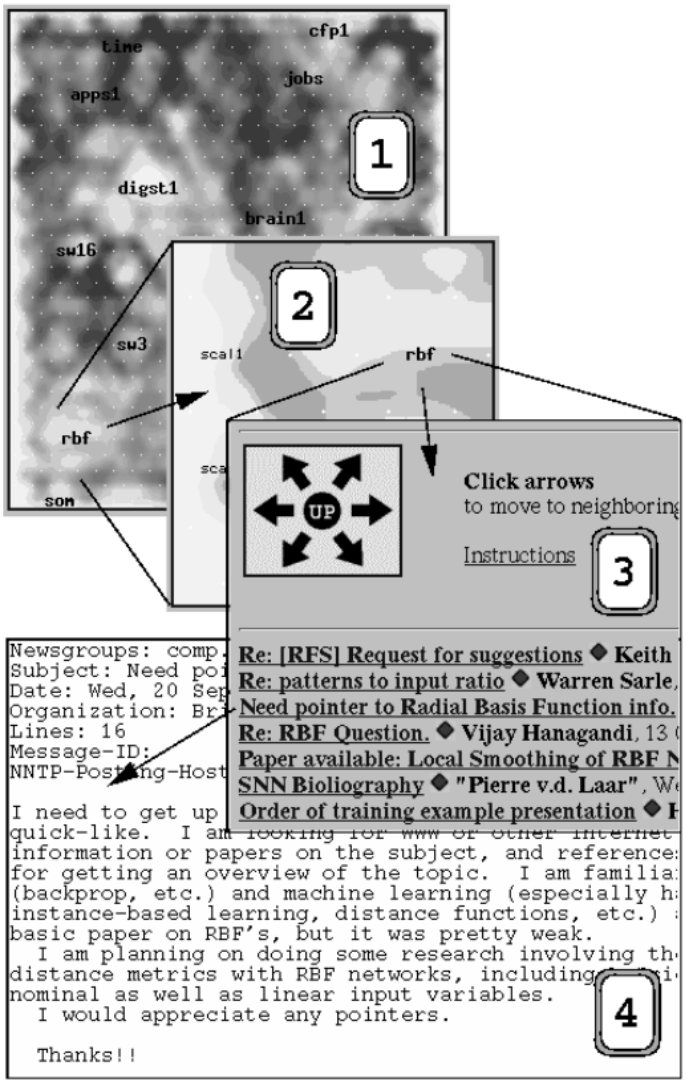
### 4.4. SOM jako narzędzie do opracowywania złożonych aplikacji

SOM jest szeroko stosowana jako metoda eksploracji danych i wizualizacji złożonych zbiorów danych. Obszary zastosowania obejmują na przykład przetwarzanie obrazu i rozpoznawanie mowy, sterowanie procesem, analizę ekonomiczną i diagnostykę w przemyśle i medycynie. Podsumowanie zastosowań inżynierii SOM podano w [5].
Niektóre aplikacje wymagają wydajnej budowy dużych map. Wyszukiwanie najlepiej pasującej jednostki jest zwykle najtrudniejszą operacją obliczeniową w SOM. Korzystając z SOM o strukturze drzewa, możliwe jest użycie wyszukiwania hierarchicznego dla najlepszego dopasowania [13]. W tej metodzie chodzi o zbudowanie hierarchii SOM i uczenie SOM na każdym poziomie przed przejściem do następnej warstwy. Kolejna metoda przyspieszenia, dzięki której zwycięzca może szybciej wyszukiwać, w oparciu o ideę Koikkalainena, jest przedstawiona w [4].
Większość aplikacji SOM używa danych liczbowych. Można również wykazać, że cechy statystyczne tekstu naturalnego także da się uznać za cechy numeryczne, które ułatwiają zastosowanie SOM w zadaniach NLP (programowanie neurolingwistyczne). Algorytm SOM znajduje zastosowanie w pojazdach autonomicznych [14].

3.Pakiet jest dostępny do pobrania pod adresem http://www.cis.hut.fi/research/som_lvq_pak.shtml
4.Strona dostępna pod adresem http://www.cis.hut.fi/~hynde/lvq/

## 5. Obliczenia i analiza wyników przy zastosowaniu metody WEBSOM

Za pomocą metody WEBSOM można organizować tekstowy zbiór dokumentów na graficznym wyświetlaczu mapy, który zapewnia przegląd kolekcji i ułatwia interaktywne przeglądanie. Interesujące dokumenty można znaleźć na mapie za pomocą wyszukiwania ukierunkowanego na treść. Każdy dokument jest zakodowany jako histogram kategorii słów, które są tworzone przez organizowanie algorytmu mapy (SOM) w oparciu o podobieństwa w kontekście słów. Zakodowane dokumenty są zorganizowane na innej samoorganizującej się mapie, mapie dokumentów, na której pobliskie lokalizacje zawierają podobne dokumenty. Szczególną uwagę poświęca się obliczaniu bardzo dużych map dokumentów, które są możliwe do obliczeń przez komputery ogólnego przeznaczenia, jeśli wymiarowość histogramów kategorii słów jest najpierw redukowana za pomocą metody losowego mapowania i jeśli w obliczeniach SOM stosuje się wydajne obliczeniowo algorytmy [15]. Na rysunku 2 przedstawiono podstawowe poziomy interfejsu WEBSOM.




Ryc. 2. Podstawowe poziomy interfejsu WEBSOM

Przedstawiono na nim: (1) całą mapę, (2) powiększoną mapę, (3) węzeł mapy i (4) widok dokumentu, pokazany w porządku rosnącej szczegółowości. Przenoszenie między poziomami lub do sąsiednich obszarów na tym samym poziomie odbywa się za pomocą kliknięć myszką na obrazkach lub na łączach dokumentów. Po znalezieniu interesującego obszaru na mapie eksploracja powiązanych dokumentów w sąsiednich obszarach jest prosta.

W WEBSOM każdy dokument jest reprezentowany na mapie dokumentu jako punkt w taki sposób, że wzajemna odległość między dwoma dowolnymi punktami reprezentacji odzwierciedla podobieństwo odpowiednich dwóch histogramów. Dlatego podobne dokumenty są mapowane blisko siebie na mapie dokumentu, podobnie jak książki na półkach dobrze zorganizowanej biblioteki. Metoda WEBSOM jest z łatwością stosowana do wszelkiego rodzaju gromadzenia dokumentów tekstowych, nawet jeśli nie dostarczono im słów kluczowych. Zorganizowano kolekcje aż 100 000 dokumentów na mapach o liczbie rzędu 10 000 węzłów.

Metoda jest szczególnie odpowiednia do zadań związanych z eksploracją, w których użytkownicy albo nie znają domeny bardzo dobrze, albo mają tylko ograniczone wyobrażenie o zawartości analizowanej bazy pełno-tekstowej. Dzięki WEBSOM dokumenty są porządkowane zgodnie z ich treścią. Mapy pomagają w eksploracji, dając ogólną wizualizację tego, jak wygląda przestrzeń informacyjna. [15]

## 6. Podsumowanie

Liczba potencjalnych zastosowań algorytmu Kohonena w przyszłych kierunkach jest duża, począwszy od badań nad implikacjami w lingwistyce i filozofii języka. Z drugiej strony, rozwój aplikacji uzasadnia dalsze rozwijanie metody SOM. Interesujące byłoby zbadanie związku między modelowaniem interpretacji języka naturalnego za pomocą SOM a niektórymi pokrewnymi teoriami. Interesujące punkty widzenia można znaleźć w językoznawstwie ogólnym, językoznawstwie kognitywnym, kognitywistyce i filozofii języka. Specyficzne powiązane teorie obejmują np. teorię prototypów, gramatykę kognitywną, teoria relewancji i konstruktywizm. Wykorzystanie SOM w przetwarzaniu w języku naturalnym można porównać z szerokim spektrum innych podejść statystycznych i opartych na teorii ciał. Istnieje kilka alternatyw kodowania informacji kontekstowych w tekstach dla mapy kategorii słów. Można oczywiście opracować inne sposoby gromadzenia informacji kontekstowych. Mogą one być oparte na przykład na kojarzeniu słów z wprowadzaniem innych modalności. Można opracować nowe aplikacje w takich obszarach jak tłumaczenie maszynowe i systemy współpracy.

W tłumaczeniu maszynowym można zastosować mapy kategorii słów, np. w selekcji leksykalnej, tj. znalezienie odpowiedniego słowa lub frazy do zastosowania w kontekście. Materiały tekstowe w językach innych niż angielski mogą być również przetwarzane. W wymienionych wyżej pracach różne formy słowne leksemów były kodowane oddzielnie. Takie podejście wydawało się działać dobrze, gdy rozważany jest angielski.

Jeśli podstawowa forma słowa może wystąpić w ponad 10 000 różnych fleksyjnych form wyrazów (takich jak czasowniki w języku fińskim), nowe problemy muszą zostać rozwiązane w inny sposób.

W metodzie WEBSOM związek pomiędzy charakterystyką danych wejściowych, wyborami wartości parametrów i możliwymi narzędziami do przetwarzania wstępnego wymaga dogłębnych badań. Jako konkretny przykład można rozważyć wpływ długości dokumentu na proces WEBSOM. W niektórych przypadkach długi dokument może być podzielony na krótsze części, które prawdopodobnie mają jaśniejszą tożsamość. W tradycyjnym wyszukiwaniu słów kluczowych, które wykorzystuje algebrę Boole'a do łączenia słów kluczowych, zaletą jest możliwość łączenia terminów na różne sposoby tak, aby użytkownik kontrolował proces. Podobną funkcjonalność można osiągnąć za pomocą metody WEBSOM, umożliwiając użytkownikowi łączenie wyników kolejnych wyszukiwań na mapie dokumentu. Przydatne byłoby również zbadanie zadowolenia użytkowników z oceny metody WEBSOM.

## Literatura:

[1] M. M. Nassand L. N. Cooper, "A theory for the development of feature detecting cells invisual cortex," Biot. Cybern.,vol. 19, pp. 1-18, 1975

[2] S. Athuraliya, S.H. Low, V.H. Li and Q. Yin, "REM: Active queue management," IEEE Network Mag., vol. 15, no. 3, pp. 48-53, 2001

[3] T. Kohonen, Self-Organizing Maps, Springer-Verlag, Berlin, Heidelberg, New York, 1995, XVI+362pp.

[4] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological Cybernetics, 43:59-69.

[5] Kohonen, T., Kaski, S. and Lappalainen, H. (1997). Self-organized formation of various invariant-feature filters in the adaptive-subspace SOM. Neural Computation, 9: 1321-1344.

[6] Fort, J.C. (2006). SOM's mathematics. Neural Networks, 19: 812–816.

[7] Heskes, T. (2001). Self-organizing maps, vector quantization, and mixture modeling. IEEE Transactions on Neural Networks, 12:1299-1305.

[8] Sammon, J. (1969). A nonlinear mapping for data structure analysis. IEEE Trans. Comp., 18(5):401-409.

[9] Amari, S.-I. (1967). A theory of adaptive pattern classifiers. IEEC, 16:299-307

[10] Carpenter, G. and Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. Computer Vision, Graphics, and Image Processing, 37:54-115.

[11]. Kohonen, T. (1992). An attempt to interpret the Self-Organizing Mapping physiologically. Report A16, Helsinki University of Technology, Laboratory of Computer and Information Science.

[12] Kohonen, T. (1993). Physiolocigal interpretation of the self-organizing map algorithm. Neural Networks, 6(7):895-905.

[13] Koikkalainen, P. (1994). Progress with the tree-structured self-organizing map. In Cohn, A. G., editor, Proceedings of ECAI'94, 11th European Conference on Artificial Intelligence, pages 211-215, New York. John Wiley & Sons.

[14] Autonomiczne pojazdy P Kardasz, O Lyubov, E Kardasz, Biuletyn Naukowy Wrocławskiej Wyższej Szkoły Informatyki Stosowanej 2017

[15] Krista Lagus, Timo Honkela, Samuel Kaski, and Teuvo Kohonen WEBSOM - A Status Report, Publications of the Finnish Artificial Intelligence Society, pp. 73-78. 1998