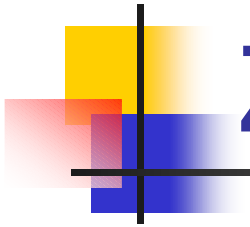
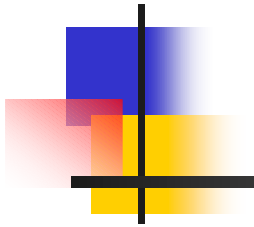
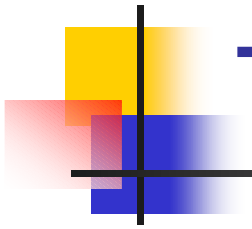


Zagadnienia

- Projektowanie struktury systemu
- Modele sterowania
- Dekompozycja modularna



Część I: Projektowanie architektury systemu



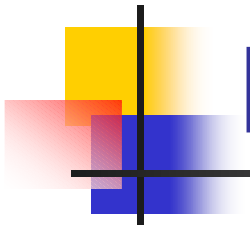
Tworzenie architektury

- Architekci systemowi stanowią interfejs pomiędzy klientem a wykonawcą systemu
- Złego projektu architektonicznego nie da się uratować dobrą konstrukcją – ta sama zasada stosuje się do oprogramowania
- Specjaliści zajmujący się projektowaniem architektury są odrębną klasą niż inżynierowie tworzący oprogramowanie

Proces projektowania architektury

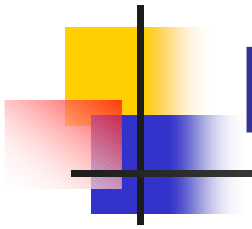
W procesie projektowania zwykle pojawiają się następujące, często przeplatające się etapy:

- Strukturyzacja systemu (dekompozycja strukturalna)
  - System podlega dekompozycji na kilka głównych podsystemów; określa się możliwą komunikację pomiędzy podsystemami
- Modelowanie sterowania
  - Tworzy się model relacji sterujących pomiędzy poszczególnymi częściami systemu
- Dekompozycja modularna
  - Wyszczególnione podsystemy ulegają dalszemu podziałowi na moduły



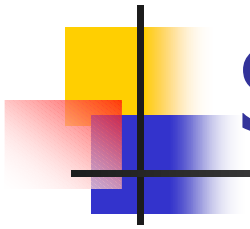
Podsystemy a moduły

- Rozróżnienie pomiędzy podsystemami a modułami nie jest ściśle zdefiniowane
- Podsystem stanowi odrębną całość. Funkcjonowanie podsystemu jest niezależne od serwisów dostarczanych przez pozostałe podsystemy
- Moduł jest składnikiem systemu dostarczającym funkcjonalności innym komponentom ale nie jest rozpatrywany jako odrębny system



Modele architektury

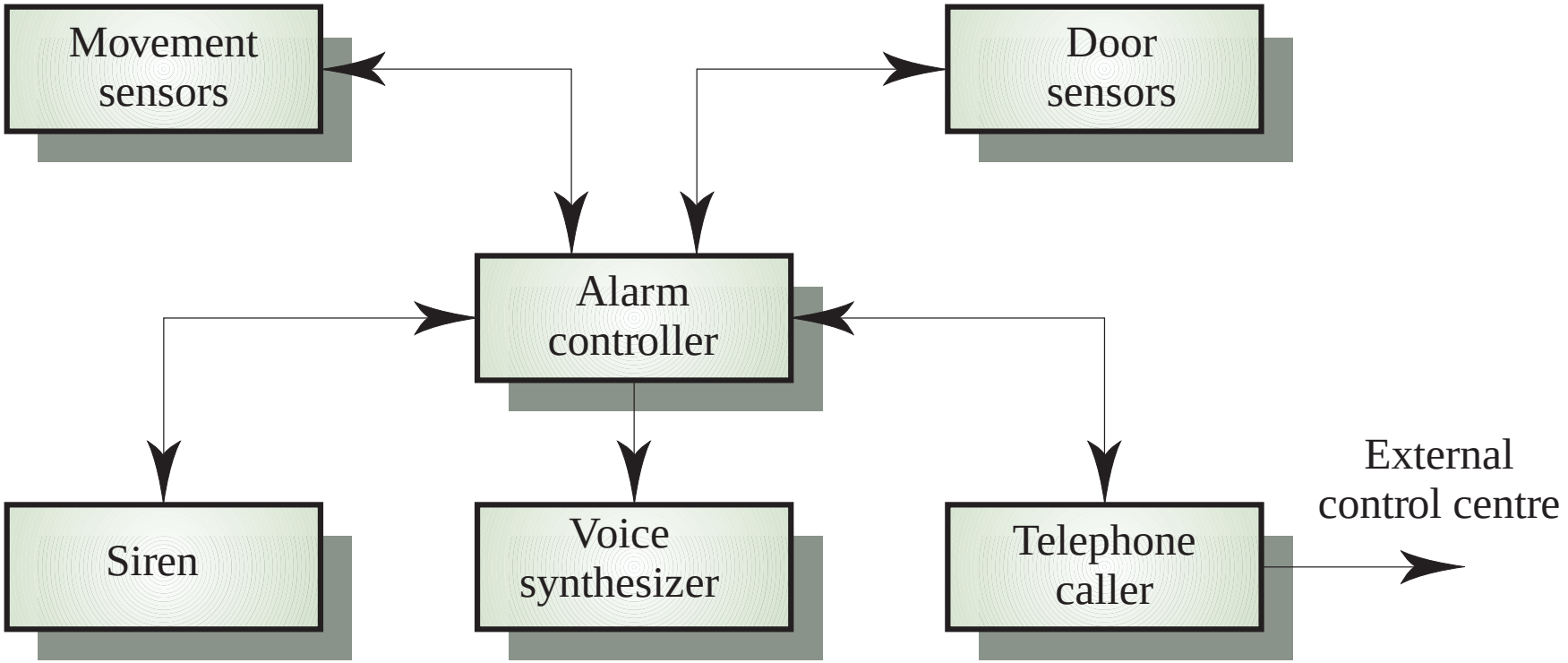
- Dekompozycja strukturalna, modularna oraz projekt sterowania opierają się o wybrany model bądź styl architektury
- Większość tworzonych systemów jest heterogenicznych
  - Poszczególne składowe systemu opierają się o inne modele
- Wybrany model architektury ma w oczywisty sposób wpływ na wydajność, niezawodność oraz pielęgnowalność systemu
  - Stąd wybór danego stylu może opierać się o wymagania niefunkcjonalne



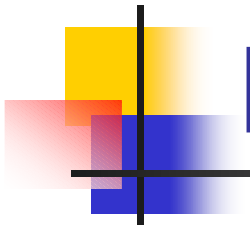
Strukturyzacja systemu

- Ma w zamierzeniu podzielić system na komunikujące się ze sobą podsystemy
- Projekt architektury zwykle przedstawia się w postaci schematu blokowego
  - Prezentuje on ogólny widok struktury systemu
- Bardziej szczegółowe modele dodatkowo mogą przedstawiać sposób współdzielenia danych przez podsystemy oraz ich interfejsy

Strukturyzacja: przykład projektu architektury

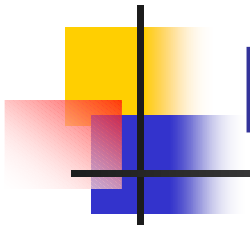


(C) 1995 Ian Somerville



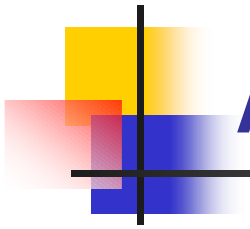
Model z repozytorium danych

- Podystemy muszą mieć możliwość wymiany danych. Można to zapewnić na dwa sposoby
  - Współdzielone dane są przechowywane w centralnej bazie danych, udostępnionej dla wszystkich podsystemów
  - Każdy z podystemów posiada swoją własną bazę danych. Dane są jawnie przekazywane do innych podsystemów
- W przypadku projektowania systemu który zawiera duże ilości współdzielonych danych, najczęściej używa się modelu z repozytorium danych
  - Np. C&C (command and control) systems



Model z repozytorium - cechy

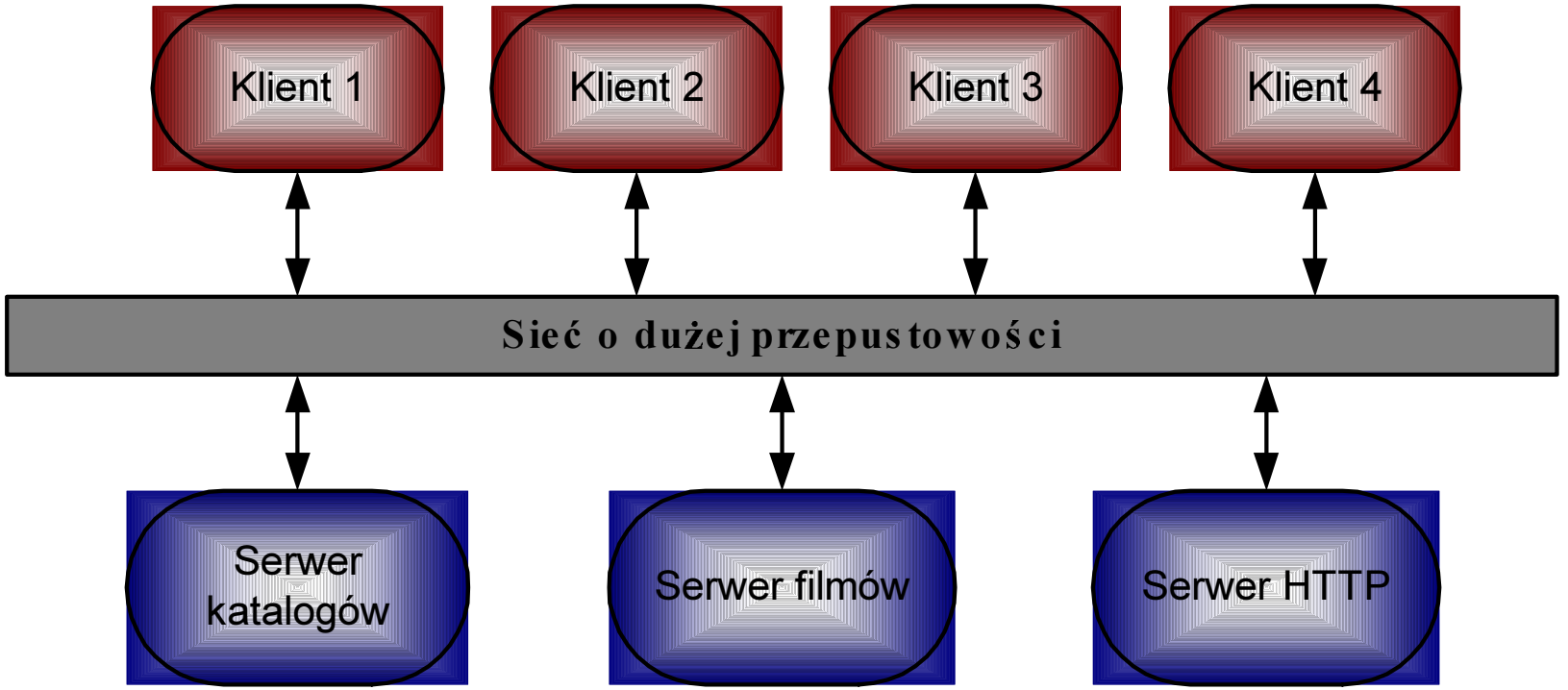
+ Wydajny sposób współdzielenia dużej ilości danych
+ Podystemy nie muszą nic wiedzieć o tym w jaki sposób dane są wytwarzane
+ Centralna polityka bezpieczeństwa, kopii zapasowych
- Podsystemy muszą uzgodnić model danych w repozytorium – kompromis
- Zmiana modelu danych – uciążliwa i kosztowna
- Trudności z efektywną dystrybucją – spójność i redundantność danych

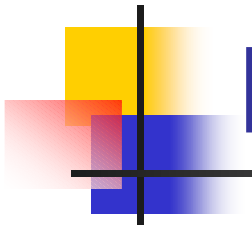


Architektura klient-serwer

- Model rozproszonego systemu; demonstruje rozproszenie danych oraz przetwarzania w obrębie sieci
- Zespół serwerów dostarczających określone usługi takie jak drukowanie, zarządzanie danymi, itp.
- Zbiór klientów żądających poszczególnych usług
- Sieć umożliwiająca klientom dostęp do usług oferowanych przez serwery

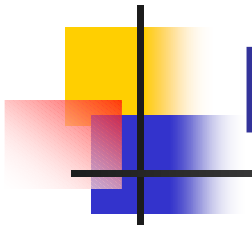
Przykład: biblioteka filmów





Klient-serwer - cechy

+ Rozproszenie danych w naturalny sposób
+ Efektywne wykorzystanie systemów sieciowych. Może umożliwić użycie tańszego sprzętu
+ Prostota i łatwość dodawania nowych serwerów bądź rozbudowy istniejących
- Brak współdzielonego modelu danych; podsystemy stosują różne reprezentacje danych. Wymiana danych (a więc i konwersja) może obniżać wydajność
- Nadmiarowe zarządzanie każdym z serwerów z osobna
- Brak centralnego rejestru nazw i usług – mogą się pojawić trudności z identyfikacją dostępnych usług

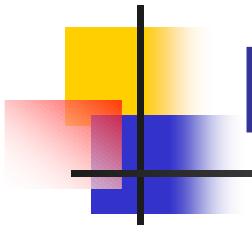


Model warstwowy

- Funkcjonalność podzielona pomiędzy warstwy
  - Każda z warst zbudowana w oparciu o serwisy oferowany przez niższą warstwę
- Wysoki poziom abstrakcji (niska współzależność):
  - Warsta wyższa nie zna szczegółów budowy warstwy niższej
  - Warstwa niższa może w ogóle nie wiedzieć o istnieniu warstw wyższych
- Przykład: model OSI dla protokołów sieciowych (Zimmerman, 1980)

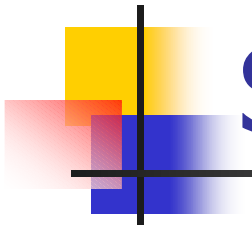
Model warstwowy - cechy

+ Wielokrotne użycie: ta sama niższa warstwa może być wykorzystywana przez różne wyższe warstwy
+ Przenośność: możliwość alternatywnych implementacji warstw niższych
+ Rozbudowywalność: możliwość rzobudowy/modyfikacji funkcjonalności warstw wyższych
- Narzut na wydajność: przekazywanie parametrów poprzez kolejne warstwy. Szczególnie uwidacznia się to w przypadku częściowego duplikowania funkcjonalności poszczególnych warstw – np. konstrukcja pakietu



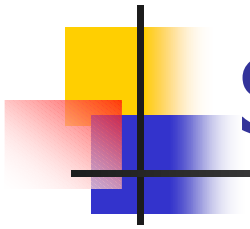
Modele sterowania

- Dotyczą przepływu sterowania pomiędzy podystemami. NIE są to modele dekompozycji systemu
- Sterowanie scentralizowane
  - Jeden wybrany podsystem jest odpowiedzialny za sterowanie, uruchamia i zatrzymuje pozostałe podsystemy
- Sterowanie oparte o zdarzenia (ang. event-based)
  - Każdy z podsystemów posiada funkcjonalność która reaguje na zdarzenia wygenerowane przez inne podsystemy bądź środowisko systemowe



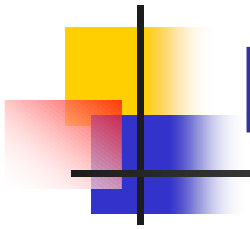
Sterowanie scentralizowane

- Wybrany podsystem steruje i zarządza wykonaniem pozostałych podsystemów
- Call-return model
  - Model procedur typu top-down. Sterowanie rozpoczyna się na szczycie hierarchii i podąża w dół. Stosuje się w systemach sekwencyjnych
- Manager model
  - Do stosowania w systemach współbieżnych. Jeden z komponentów systemu steruje wykonywaniem oraz synchronizacją pozostałych. Implementacja np. z użyciem instrukcji **case**



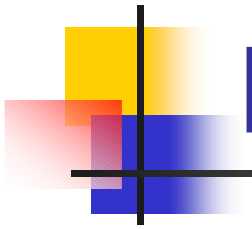
Systemy sterowane zdarzeniami

- Sterowaniem kierują tzw. zdarzenia – czas zaistnienia danego zdarzenia jest poza kontrolą podsystemu aktywowanego przez to zdarzenie
- Dwa podstawowe modele sterowania zdarzeniami
  - Modele z rozgłaszaniem (ang. broadcast models). Dane zdarzenie jest rozsyłane do wszystkich podsystemów. Każdy podsystem potrafiący obsłużyć to zdarzenie wykonuje określoną akcję
  - Modele sterowane przerwanami (ang. interrupt-driven models). Wykorzystywane w systemach czasu rzeczywistego gdzie kluczowa jest natychmiastowa reakcja na zdarzenia



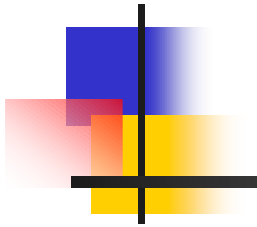
Dekompozycja modularna

- Jest to kolejny poziom strukturyzacji w którym podsystemy dzieli się na moduły
- Wyróżnia się dwa podstawowe modele dekompozycji
  - Model obiektowy w którym system ulega podziałowi na komunikujące się ze sobą obiekty
  - Model przeplywu danych w którym system jest dzielony na funkcjonalne moduły przetwarzające dane wejściowe na wyjściowe

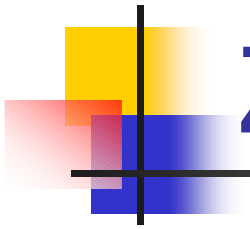


Podsumowanie

- Architekt systemowy jest odpowiedzialny za utworzenie modelu strukturalnego, modelu sterowania oraz modelu dekompozycji podstystemów (modularnej)
- W przypadku dużych systemów rzadko są one dekomponowane przy użyciu pojedynczego modelu
- Rozróżniamy następujące modele dekompozycji systemu: modele z repozytorium danych, modele typu klient serwer oraz modele warstwowe
- Modele sterowania to: modele z centralnym sterowaniem oraz modele sterowane zdarzeniami
- Modele dekompozycji modularnej to: modele obiektowe oraz modele przepływu danych

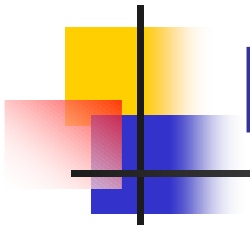


Część II:
Projektowanie funkcjonalne



Zagadnienia

- Reprezentacja projektu systemu jako zestawu funkcji współdzielących globalny stan systemu
- Notacje wykorzystywane w projektowaniu funkcjonalnym
- Projektowanie funkcjonalne na przykładzie
- Projekt przepływu danych
- Dekompozycja strukturalna

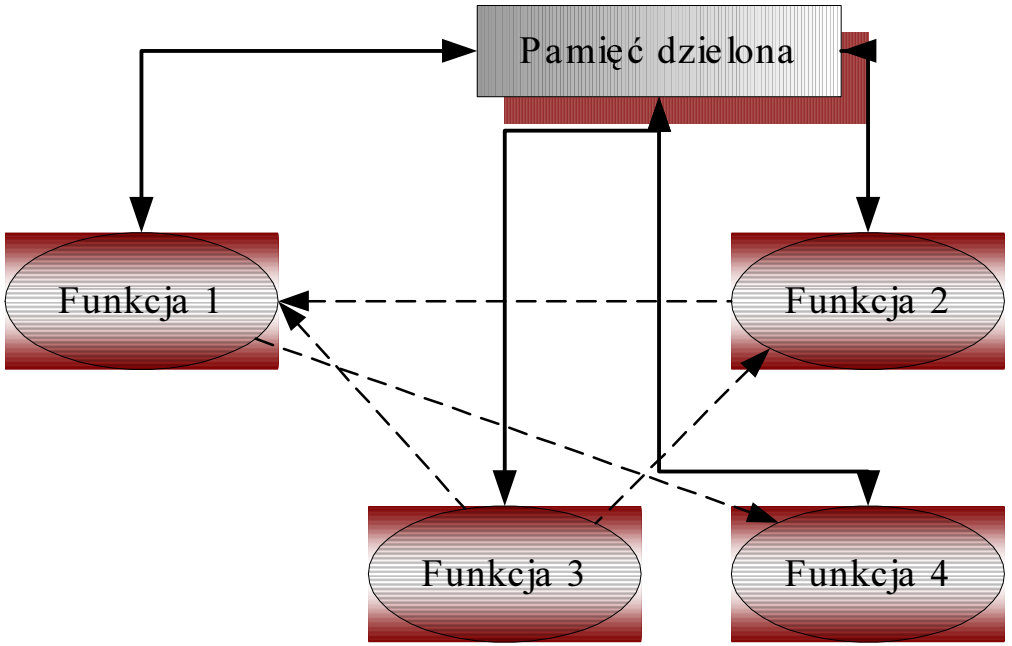


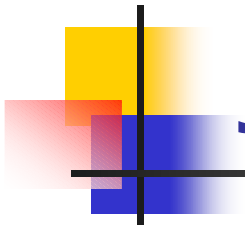
Projektowanie funkcjonalne

- (ang. function-oriented design) jest wykorzystywane nieformalnie od samego początku powstawania programów – stąd istnieje wielka mnogość systemów utworzonych z użyciem tego podejścia
- Jest bezpośrednio wspierane przez większość istniejących języków programowania
- Większość metod projektowych opiera się o podejście funkcjonalne
- Jest dostępnych wiele narzędzi CASE ułatwiających projektowanie w oparciu o tą metodologię

Strategia projektowania funkcjonalnego

- Podział systemu na współpracujące funkcje
- Stan systemu jest scentralizowany i współdzielony przez funkcje
- Lokalne informacje o stanie – tylko na czas wykonywania danej funkcji



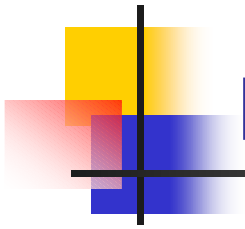


Jakie systemy projektować?

- Niektóre systemy są ze swojej natury funkcjonalne
- Są to systemy utrzymujące minimalną ilość informacji o stanie, np. w przypadku wykonywania niezależnych akcji – gdzie wynik działania danej akcji nie zależy od poprzednich wyników
- Przykład: systemy transakcyjne. Kolejne transakcje są od siebie niezależne

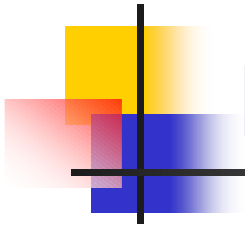
Projekt oprogr. bankomatu

```
loop
    loop
    Wyświetl_komunikat (" Wprowadź kartę") ;
        exit when Karta_wprowadzona ;
    end loop ;
    Numer_konta := Odczyt_z_karty ;
    Pobierz_dane_konta (PIN, Stan_konta, Gostępna_gotówka) ;
    if Validacja_ok (PIN) then
    loop
        Wyświetl_komunikat_z_operacjami ;
        case Operacja is
            when Tylko_gotówka => Wydaj_gotówkę (Dostępna gotówka, Żądana_ilość) ;
            when Wydruk_stanu_konta => Drukuj_stan_konta (Stan_konta) ;
        end case ;
        Zwróć_kartę ;
        Wyświetl_komunikat ("Odbierz kartę lub wciśnij klawisz kontyuacji") ;
        exit when Karta_wyjęta ;
    end loop ;
    Aktualizuj_stan_konta (Numer_konta,  Żądana ilość) ;
    else
        Zatrzymaj_kartę ;
    end if ;
end loop ;
```



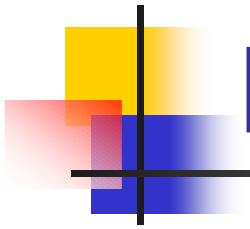
Projektowanie funkcjonalne a obiektowe

- Dla wielu typów aplikacji projektowanie obiektowe daje w efekcie bardziej niezawodny i pielęgnowalny system
- Pewne typy aplikacji utrzymują małą ilość informacji o stanie – w takich przypadkach dobrze sprawdza się projektowanie funkcjonalne
- Standardy, metody oraz narzędzia CASE dot. projektowania funkcjonalnego są mocno ugruntowane
- Istnieje konieczność pielęgnacji ogromnej liczby istniejących systemów – wiele z nich będzie działać jeszcze przez dziesięciolecia!



Proces projektowania funkcjonalnego

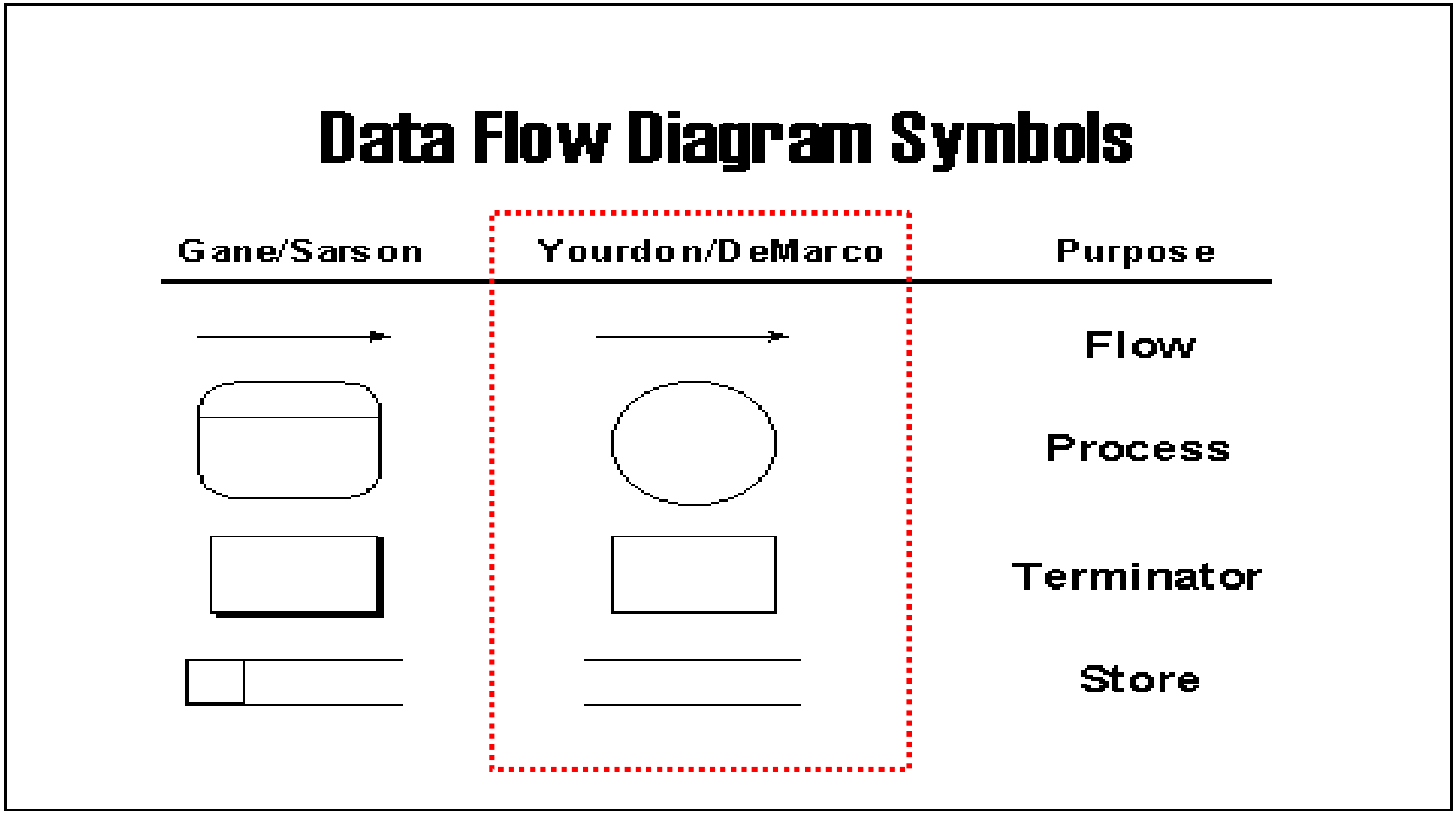
- Projekt przepływu danych
  - Modeluje przetwarzanie danych w obrębie systemu, wykorzystywane są tu diagramy przepływu danych (ang. data-flow diagrams)
- Dekompozycja strukturalna
  - Modeluje sposób w jaki poszczególne funkcje dzielą się na funkcje niższego sposobu. Sposób reprezentacji – diagramy strukturalne
- Projekt szczegółowy
  - Opisuje szczegółowo poszczególne składowe projektu oraz ich interfejsy. Informacje zwykle są zapisywane w słowniku danych a struktury sterowania w projekcie wyraża się za pomocą PDL-a



Diagramy przepływu danych

- Opisują przetwarzanie danych wejściowych; zostają one przetworzone na dane wyjściowe w wyniku sekwencji przekształceń funkcjonalnych
- Nie opisują JAK wyglądają takie przekształcenia
- Stanowią integralną część wielu metod projektowych
- Są przetwarzane następnie na projekt sekwencyjny lub równoległy. W projekcie sekwencyjnym elementy przetwarzające dane to funkcje/procedury. W projekcie równoległym elementy te odpowiadają procesom bądź zadaniom.

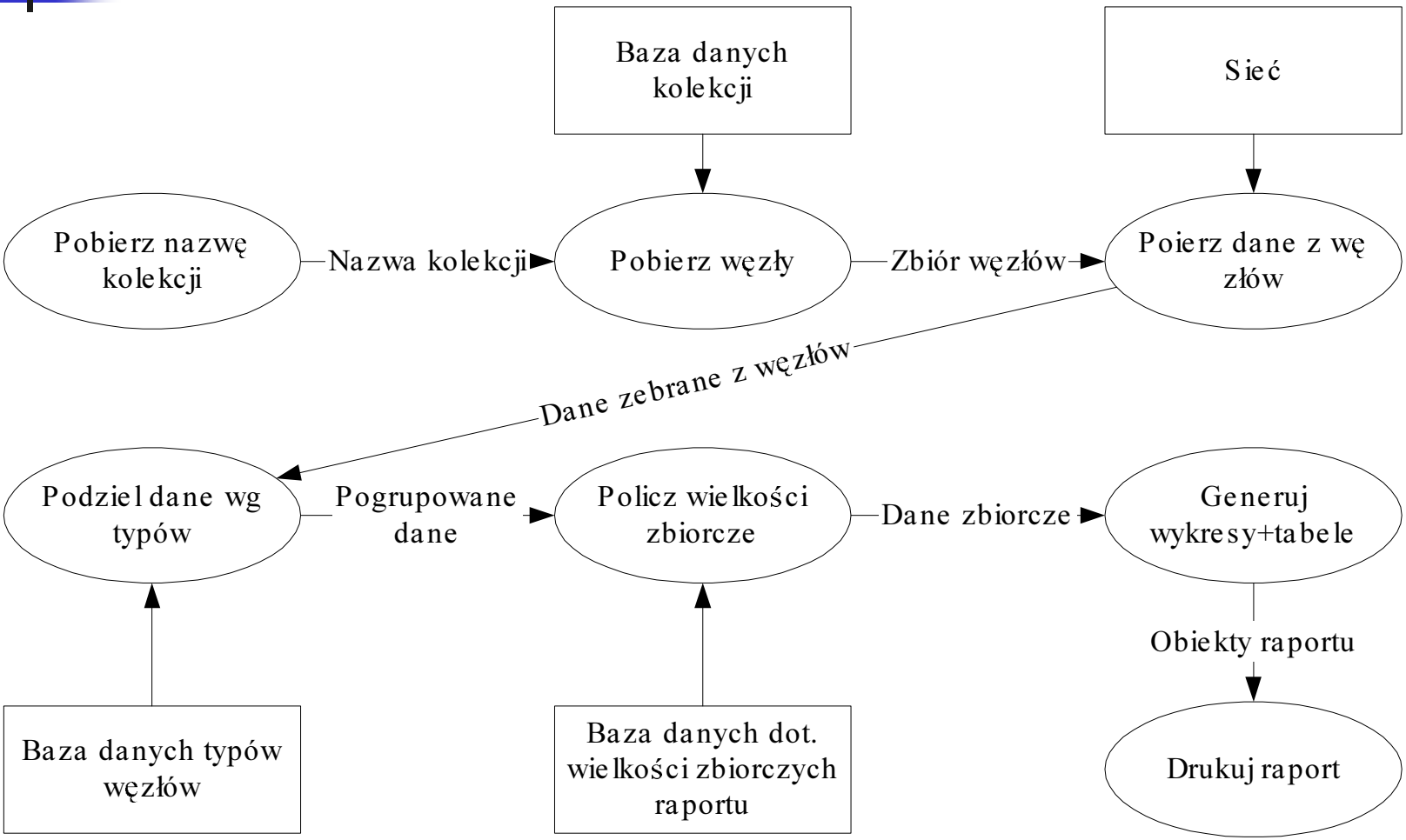
Notacje przepływu danych

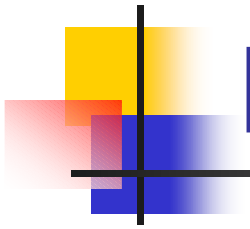


Notacje przepływu danych (c.d)

- Aby diagram przepływu danych był kompletny, poszczególne jego składniki muszą posiadać odpowiednie etykiety:
  - **Przepływ danych** – wyrażenie opisujące dane
  - **Terminator** – wyrażenie opisujące system, agenta, urządzenie, itp. skąd pochodzą dane wchodzące do systemu bądź gdzie wychodzą dane opuszczające system
  - **Magazyn danych** – wyrażenie opisujące plik, bazę danych bądź repozytorium przechowujące dane
  - **Proces** – wyrażenie opisujące operację wykonywaną na danych
- Procesy mogą być również etykietowane za pomocą nazw podsystemów bądź operacji wykonywanych na danych

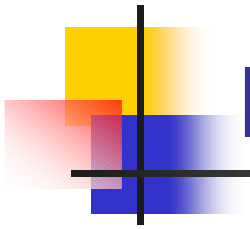
SNMP – raport aktywności węzłów





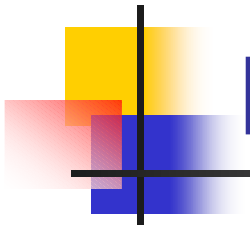
Dekompozycja strukturalna

- Celem dekompozycji strukturalnej jest utworzenie modelu pokazującego wywołania poszczególnych funkcji czyli dynamiczną strukturę
- NIE jest to statyczny podział na komponenty
- Celem dekompozycji jest utworzenie jednostek spójnych i luźno ze sobą połączonych
- W swojej istocie polega na konwersji diagramu przepływu danych na diagram strukturalny



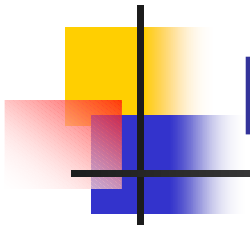
Dekompozycja strukturalna - notacja

- Funkcja – w postaci prostokąta
- Hierarchia wywołań – połączenia pomiędzy funkcjami
  - Wejścia i wyjścia (implementowane bądź poprzez przekazywanie parametrów bądź przez pamięć dzieloną) – strzałki
  - Strzałka dochodząca do funkcji reprezentuje jej dane wejściowe, wychodząca – dane wyjściowe
- Magazyny danych – zaokrąglone prostokąty



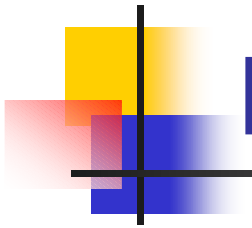
Dekompozycja – wskazówki (1/2)

- Konwersja DFD na diagram strukturalny
  - Nie jest czynnością mechaniczną
  - Ale istnieje szereg ogólnych wytycznych ułatwiających stworzenie rozsądnego projektu
- W przypadku aplikacji biznesowych wystarczy jeśli diagram strukturalny najogólniejszego poziomu zawiera 4 funkcje:
  - Odczyt danych wejściowych (z ew. walidacją poprawności)
  - Przetworzenie danych
  - Aktualizacja współdzielonych danych systemowych
  - Generacja danych wyjściowych (z ew. formatowaniem, drukowaniem, zapisem)
- Funkcje sterujące/synchronizujące powinny być umieszczone możliwie blisko wierzchołka hierarchii funkcji



Dekompozycja – wskazówki (2/2)

- Celem procesu projektowania jest identyfikacja luźno powiązanych i spójnych funkcji. Każda z zidentyfikowanych funkcji powinna wykonywać tylko jedną logicznie wyodrębnioną czynność
- Intuicyjna wskazówka dotycząca szczegółowości: każdy węzeł w diagramie strukturalnym powinien posiadać od dwóch do siedmiu podwęzłów
  - Tylko jeden podwęzeł sugeruje zwykle że dany węzeł jest zbyt mało spójny, tzn. nie ma wystarczająco jasno odseparowanej funkcjonalności
  - Zbyt dużo podwęzłów oznacza wprowadzenie zbyt dużego stopnia szczegółowości na danym etapie projektu

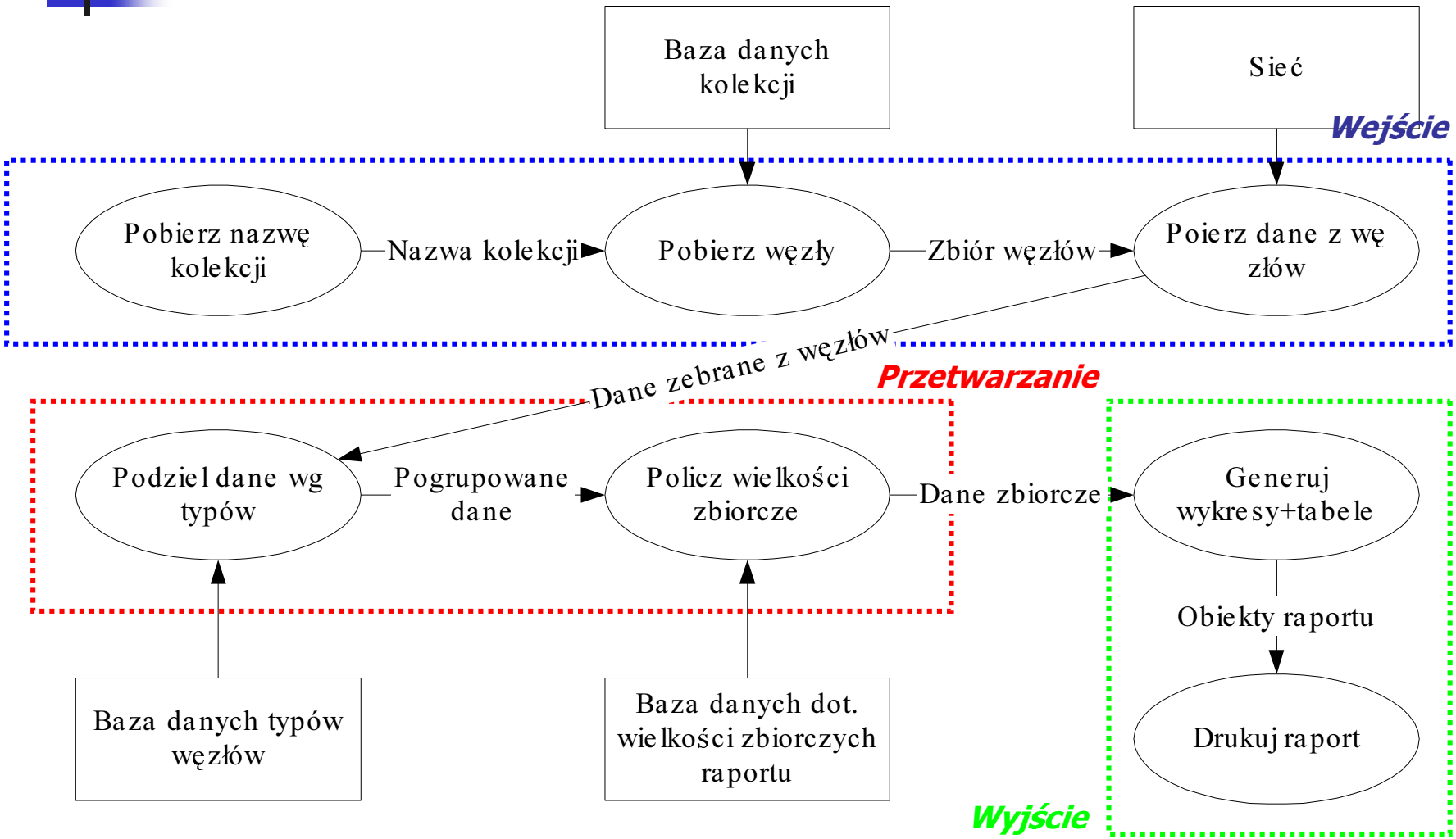


Kolejne etapy procesu

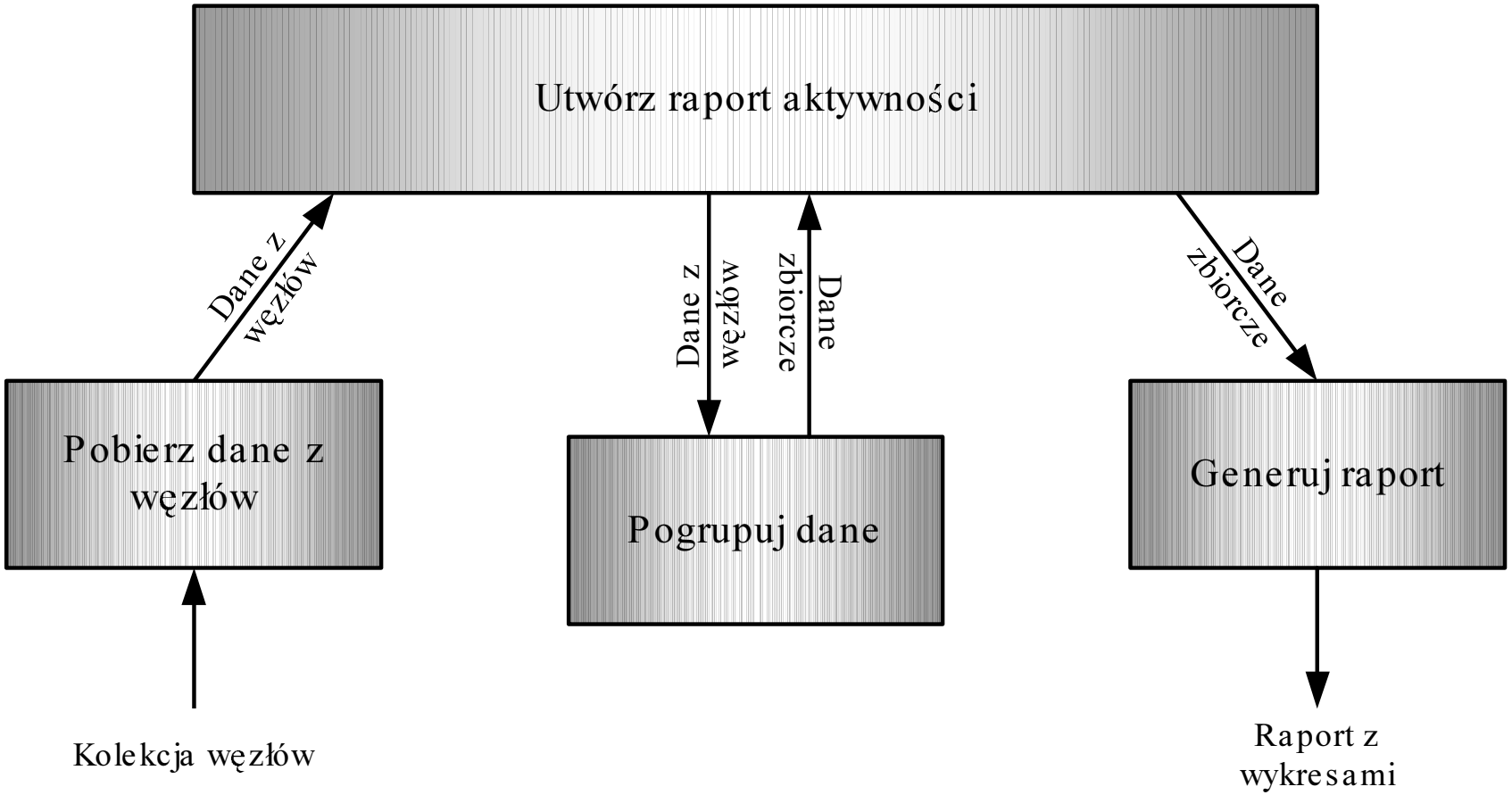
Proces konwersji diagramu przepływu danych na diagram strukturalny

- Identyfikacja transformacji przetwarzających
  - Transformacje odpowiedzialne za przetwarzanie danych; nie powiązane z obsługą we/wy. Powinny być zgrupowane w jedną funkcję na najbardziej ogólnym poziomie diagramu strukturalnego
- Identyfikacja transformacji wejściowych
  - Dotyczą one odczytu, walidacji oraz formatowania danych wejściowych. Grupuje się je jako funkcję wejściową
- Identyfikacja transformacji wyjściowych
  - Dotyczą one formatowania oraz zapiasu danych wyjściowych. Grupuje się je jako funkcję wyjściową.

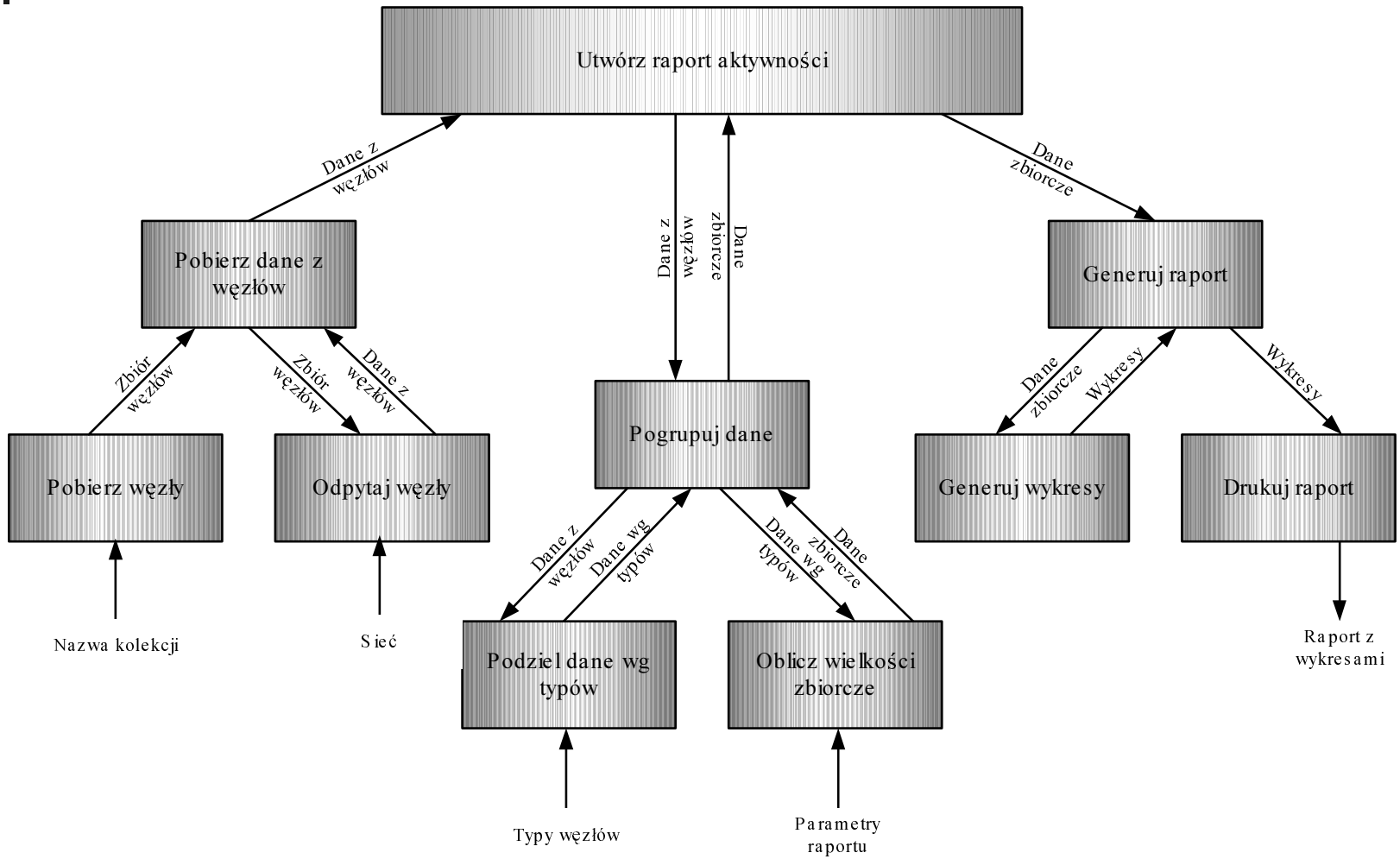
SNMP – raport aktywności węzłów



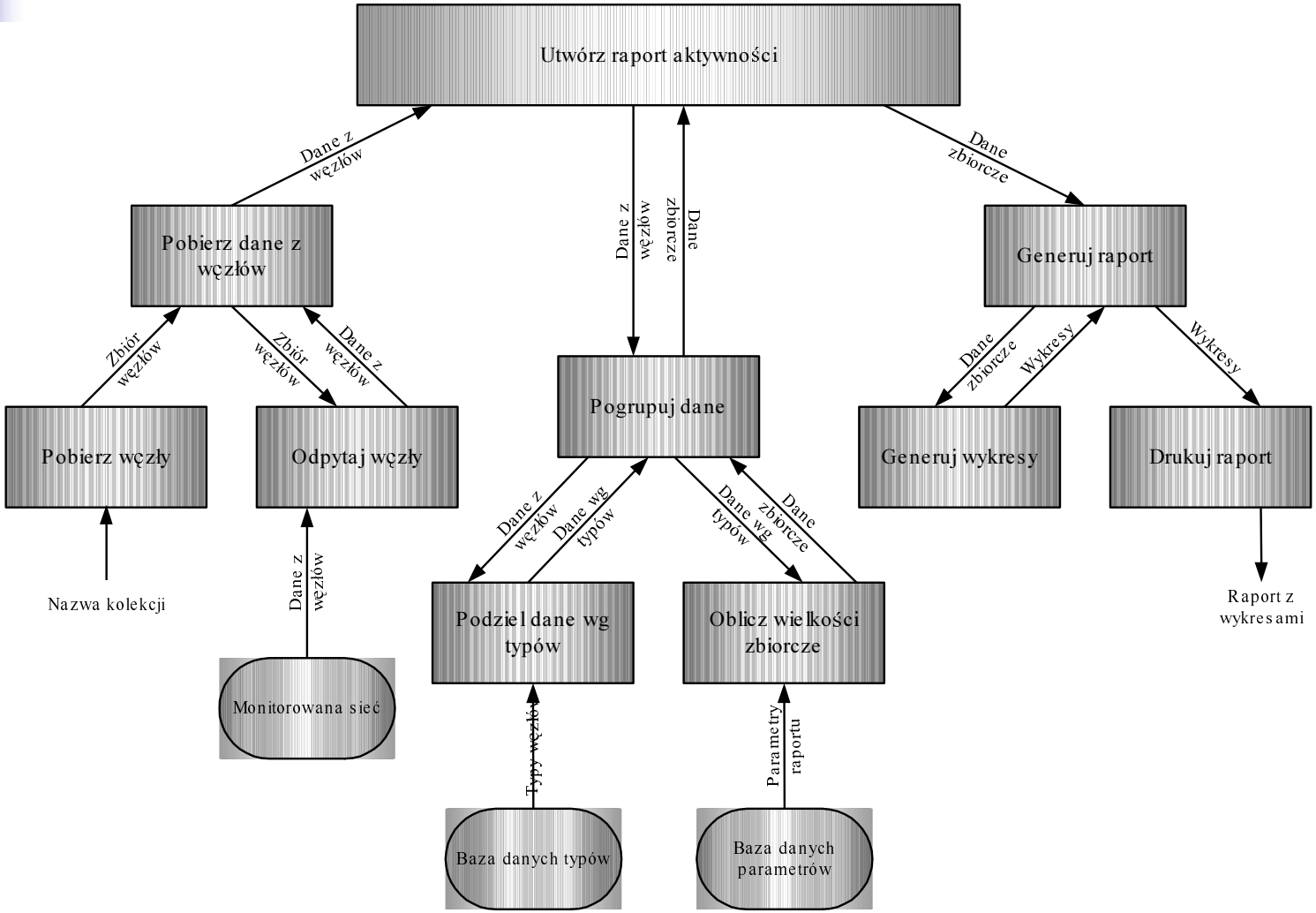
Początkowa postać diagramu

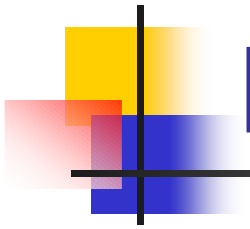


Poszerzony diagram strukturalny



Końcowa postać diagramu



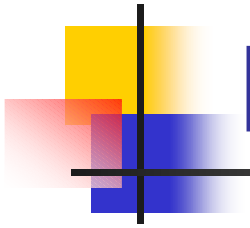


Projekt szczegółowy

- W jego wyniku dla każdej funkcji powinna powstać krótka specyfikacja projektowa (ang. minispec). Powinna ona opisywać przetwarzanie, dane wejściowe i wyjściowe
  - Na tym etapie często udaje się ujawnić błędy w początkowej dekompozycji oraz brakujące (pominięte) funkcje
- Specyfikacje funkcjonalne powinny być przechowywane w słowniku danych
  - Zmniejszenie ryzyka użycia już wykorzystywanych nazw
- Na podstawie specyfikacji można generować szczegółowe opisy projektowe
  - Wyrażone w PDL-u bądź w wybranym języku programowania

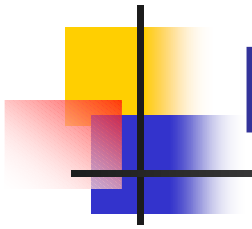
Pozycje w słowniku danych

Nazwa	Typ	Opis
Nazwa kolekcji	STRING	Nazwa zbioru poszczególnych węzłów w sieci
Odpytaj węzły	FUNCTION	**Wejście**: Lista węzłów **Funkcjonalność**: Dla każdego węzła z listy wysyła poprzez sieć zapytanie do lokalnego agenta SNMP o predefiniowane wartości **Wyjście**: Lista struktur zawierających węzły oraz zestaw uzyskanych wartości
Podziel dane wg typów	FUNCTION	**Wejście**: Lista struktur węzłow wraz z wynikami pomiarów **Funkcjonalność**: Dla każdego węzła z listy przypisuje jego typ zdefiniowany w bazie. Dla każdego typu gromadzi sumaryczne wyniki pomiarów **Wyjście**: Lista struktur węzłów z przypisanymi typami, lista typów wraz z sumarycznymi wartościami



Podsumowanie (1/2)

- Projektowanie funkcjonalne opiera się o identyfikację funkcji przetwarzających dane wejściowe na wyjściowe
- Wiele systemów biznesowych to systemy przetwarzania transakcyjnego które ze swojej natury są funkcjonalne
- Projektowanie funkcjonalne składa się z:
  - Identyfikacji transformacji danych
  - Dekompozycji funkcji na podfunkcje
  - Utworzenia szczegółowych opisów zidentyfikowanych podfunkcji



Podsumowanie (2/2)

- Diagramy DFD pozwalają na opis przepływu danych w obrębie całości systemu. Diagramy strukturalne przedstawiają dynamiczną hierarchię wywołań funkcji
- Diagramy DFD mogą być zaimplementowane wprost jako komunikujące się ze sobą procesy sekwencyjne.