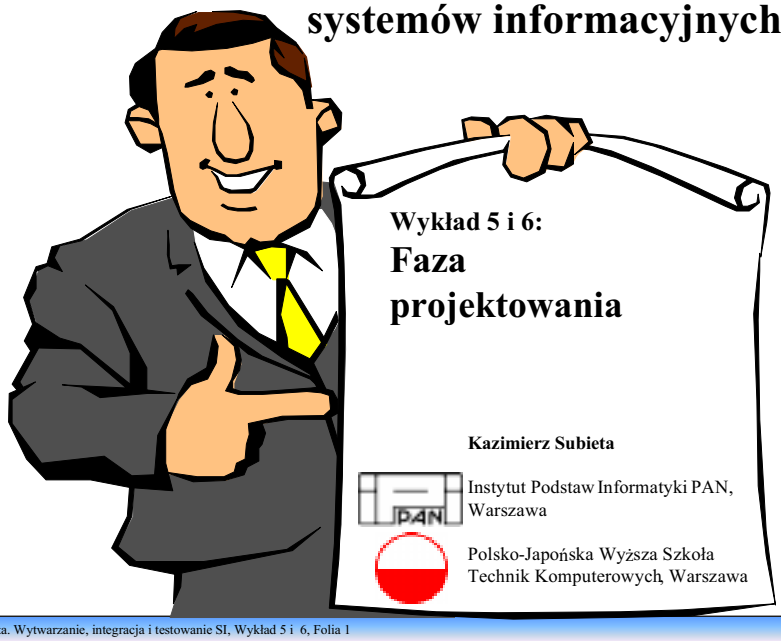
Wytwarzanie, integracja i testowanie systemów informacyjnych



Wykład 5 i 6:
Faza projektowania

Kazimierz Subieta

Instytut Podstaw Informatyki PAN, Warszawa

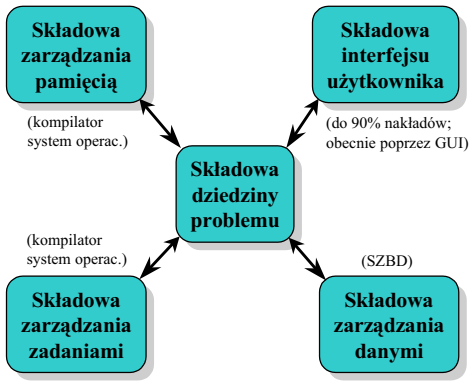
Polsko-Japońska Wyższa Szkoła Technik Komputerowych, Warszawa

Projektowanie składowych systemu nie związanych z dziedziną problemu

Projekt skonstruowany przez uszczegółowienie modelu opisuje składowe programu odpowiedzialne za realizację podstawowych zadań systemu.

Gotowe oprogramowanie musi się jednak składać z dodatkowych składowych:

- składowej interfejsu uzytkownika
- składowej zarządzania danymi (przechowywanie trwałych danych)
- składowej zarządzania pamięcią operacyjną
- składowej zarządzania zadaniami (podział czasu procesora)



RAD - *Rapid Application Development*

Szybkie rozwijanie aplikacji.

Terminem tym określa się narzędzia i techniki programowania umożliwiające szybką budowę prototypów lub gotowych aplikacji, z reguły oparte o programowanie wizyjne. Termin RAD występuje niekiedy jako synonim języków/środowisk czwartej generacji (4GL). Przykładami narzędzi RAD są: Borland Delphi RAD Pack, IBM VisualAge (for Cobol, Java, C++, Smalltalk), Microsoft Access Developer's Toolkit, Microsoft Visual FoxPro Professional, PowerBuilder Desktop, Power++ i wiele innych.

Łatwa realizacja pewnych funkcji systemu poprzez tworzenie bezpośredniego połączenia pomiędzy składowymi interfejsu użytkownika (dialogami, raportami) z elementami zarządzania danymi w bazie danych (przeważnie relacyjnej).

Składowa dziedziny problemu w najmniejszym stopniu poddaje się automatyzacji. Niekiedy inne ograniczenia lub nietypowość wykluczają możliwość zastosowania narzędzi RAD.

Projektowanie składowej interfejsu użytkownika

W ostatnich latach nastąpił gwałtowny rozwój narzędzi graficznych służących do tego celu: MS Windows, Object Windows, MS Foundation Class.

Systemy zarządzania interfejsem użytkownika: Zapp Factory, Visual Basic.

- Interaktywne projektowanie dialogów, okien, menu, map bitowych, ikon oraz pasków narzędziowych z wykorzystaniem bogatego zestawu gotowych elementów
- Definiowanie reakcji systemu na zajście pewnych zdarzeń, tj. akcji podejmowanych przez użytkownika (np. wybór z menu).
- Symulacja pracy interfejsu.
- Generowanie kodu, często z możliwością wyboru jednego z wielu środowisk docelowych.

Organizacja interakcji z użytkownikiem

Realizacja komunikacji z użytkownikiem:

- **Za pomocą linii komend**

  Dla niewielkich systemów.
  Dla prototypów.
  Dla zaawansowanych użytkowników.
  Często szybszy od niż interfejs pełnoekranowy.

- **W pełnoekranowym środowisku okienkowym**

  Tworzenie ma sens dla dużych systemów.
  Wygodny dla początkujących i średnio zaawansowanych użytkowników

Komunikacja użytkownika z systemem polega na:

- Wydawaniu przez użytkownika poleceń.
- Przepływie danych pomiędzy użytkownikiem a systemem i odwrotnie.

Typowe sposoby wydawania przez użytkownika poleceń systemowi

- **Wpisywanie poleceń za pomocą linii komend.**
- **Wybór opcji z menu.**
- **Wciśnięcie odpowiedniej kombinacji klawiszy (skrótu).**
- **Korzystanie z ikon w paskach narzędziowych.**
- **Wybór przycisku w dialogu.**
- **Korzystanie z nawigacji kursorem myszy i przycisków myszy.**

Wprowadzanie i wyprowadzanie danych

Wprowadzanie przez użytkownika:

Podawanie parametrów poleceń w przypadku systemów z linią komend

Wprowadzanie danych w odpowiedzi na zaproszenie systemu

Wprowadzanie danych w dialogach

Wyprowadzanie przezsystem:

Wyświetlanie informacji w dialogach.

Wyświetlanie i/lub wydruki raportów.

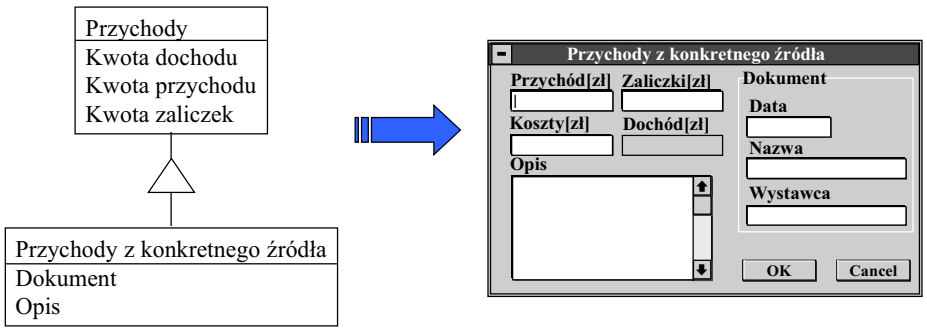
Graficzna prezentacja danych.

Prototyp interfejsu użytkownika może powstać już w fazie określenia wymagań. Systemy zarządzania interfejsem użytkownika pozwalają na wygodną budowę prototypów oraz wykorzystanie prototypu w końcowej implementacji.

Przykład okna dialogowego

Dialog:
- Przepływ danych pomiędzy użytkownikiem a systemem
- Parametry komunikatów wysyłanych przez użytkownika
- Metody i procesy, które zgodnie ze specyfikacją służą do edycji obiektów, encji lub zbiorników danych

Edycja obiektu “Przychody z konkretnego źródła”:



Zasady projektowania interfejsu użytkownika (1)

- **Spójność**. Wygląd oraz obsługa interfejsu powinna być podobna w momencie korzystania z różnych funkcji. Poszczególne programy tworzące system powinny mieć zbliżony interfejs, podobnie powinna wyglądac praca z rozmaitymi dialogami, podobnie powinny być interpretowane operacje wykonywane przy pomocy myszy. Proste reguły:
  - Umieszczanie etykiet zawsze nad lub obok pól edycyjnych
  - Umieszczanie typowych pól OK i Anuluj zawsze od dołu lub od prawej.
  - Spójne tłumaczenie nazw angielskich, spójne oznaczenia pól.

- **Skróty dla doświadczonych użytkowników**. Możliwość zastąpienia komend w paskach narzędziowych przez kombinację klawiszy.

- **Potwierdzenie przyjęcia zlecenia użytkownika**. Realizacja niektórych zleceń może trwać długo. W takich sytuacjach należy potwierdzić przyjęcie zlecenie, aby użytkownik nie był zdezorientowany odnośnie tego co się dzieje. Dla długich akcji - wykonywanie sporadycznych akcji na ekranie (np. wyświetlanie sekund trwania lub sekund do przewidywanego zakończenia dla uświadomienia użytkownikowi, że “coś się dzieje”.

Zasady projektowania interfejsu użytkownika (2)

- **Prosta obsługa błędów**. Jeżeli użytkownik wprowadzi błędne dane, to po sygnale błędu system powinien automatycznie przejść do kontynuowania przez niego pracy z poprzednimi poprawnymi wartościami.

- **Odwoływanie akcji**(undo). W najprostszym przypadku jest to możliwość cofnięcia ostatnio wykonanej operacji. Jeszcze lepiej jeżeli sytem pozwala cofnąć się dowolnie daleko w tył.

- **Wrażenie kontroli nad systemem** Użytkownicy nie lubią, kiedy system sam robi coś, czego użytkownik nie zainicjował, lub kiedy akcja systemu nie daje się przerwać. System nie powinien inicjować długich akcji (np. składowania) nie informując użytkownika co w tej chwili robi oraz powinien szybko reagować na sygnały przerwania akcji (Esc, Ctrl+C, Break,...)

Zasady projektowania interfejsu użytkownika (3)

- **Nieobciążanie pamięci krótkotrwałej użytkownika**. Użytkownik może zapomnieć o tym po co i z jakimi danymi uruchomił dialog. System powinien wyświetlać stale te informacje, które są niezbędne do tego, aby użytkownik wiedział, co aktualnie się dzieje i w którym miejscu interfejsu się znajduje.

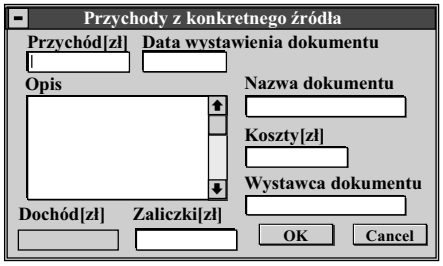
- **Grupowanie powiązanych operacji**. Jeżeli zadanie nie da się zamknąć w prostym dialogu lub oknie, wówczas trzeba je rozbic na szereg powiązanych dialogów. Użytkownik powinien być prowadzony przez ten szereg, z możliwością łatwego powrotu do wcześniejszych akcji.

Reguła 7 +/- 2.
Człowiek może się jednocześnie skupić na 5 - 9 elementach.
Ta reguła powinna być uwzględniana przy projektowaniu interfejsu użytkownika. Dotyczy to liczby opcji menu, podmenu, pól w dialogu, itd. Ograniczenie to można przełamać poprzez grupowanie w wyraźnie wydzielone grupy zestawów semantycznie powiązanych ze sobą elementów.

Dwa funkcjonalnie równoważne dialogi

Wewnętrzne grupowanie pól:

Bez wewnętrznego grupowania pól:



Projektowanie składowej zarządzania danymi

Trwałe dane mogą być przechowane w:

• pliku
• w bazie danych (relacyjnej, obiektowej, lub innej).

Poszczególne elementy danych- zestawy obiektów lub krotek- mogą być przechowywane w następującej postaci:

• w jednej relacji lub pliku
• w odrębnym pliku dla każdego rodzaju obiektów lub krotek

Sprowadzenie danych do pamięci operacyjnej oraz zapisanie do trwałej pamięci może być:

• na bieżąco, kiedy program zażąda dostępu i kiedy następuje zapełnienie bufora
• na zlecenie użytkownika

Zalety baz danych (relacyjnych i obiektowych)

- Wysoka efektywność i stabilność
- Bezpieczeństwo i prywatność danych, spójność i integralność przetwarzania
- Automatyczne sprawadzanie warunków integralności danych
- Wielodostęp, przetwarzanie transakcji
- Rozszerzalność (zarówno dodawanie danych jak i dodawanie ich rodzajów)
- Możliwość geograficznego rozproszenia danych
- Możliwość kaskadowego usuwania powiązanych danych
- Dostęp poprzez języki zapytań (SQL, OQL)

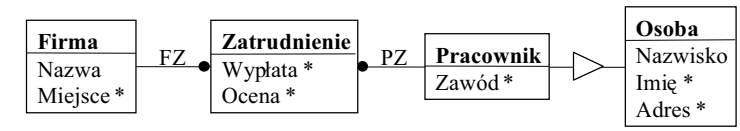
Integralność: poprawność danych w sensie ich organizacji i budowy.

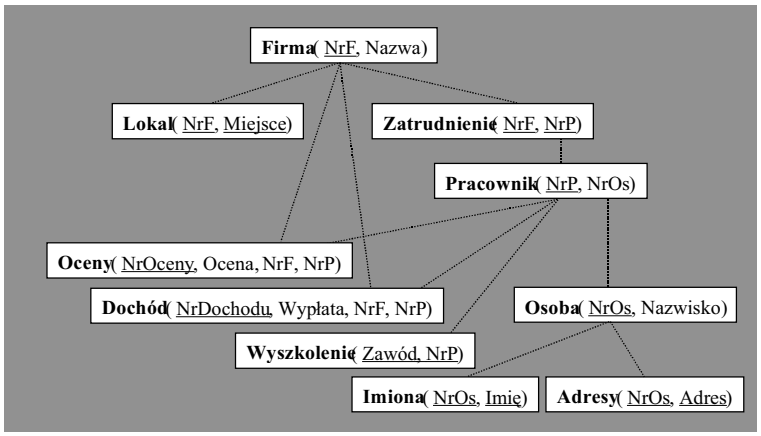
Spójność: zgodność danych z rzeczywistością lub z oczekiwaniami użytkownika.

Wady relacyjnych baz danych

- Konieczność przeprowadzenie nietrywialnych odwzorowań przy przejściu z modelu pojęciowego (np. w OMT) na strukturę relacyjną.
- Ustalony format krotki powodujący trudności przy polach zmiennej długości.
- Trudności (niesystematyczność) reprezentacji dużych wartości (grafiki, plików tekstowych, itd.)
- W niektórych sytuacjach - duże narzuty na czas przetwarzania
- Niedopasowanie interfejsu dostępu do bazy danych (SQL) do języka programowania (np. C), określana jako “niezgodność impedancji”.
- Brak możliwości rozszerzalności typów (zagnieżdżania danych)
- Brak systematycznego podejścia do informacji proceduralnej (metod)

Niezgodność modelu pojęciowego i relacyjnego





Optymalizacja projektu (1)

Bezpośrednia implementacja projektu może prowadzić do systemu o zbyt niskiej efektywności.

- Wykonanie pewnych funkcji jest zbyt wolne
- Struktury danych mogą wymagać zbyt dużej pamięci operacyjnej i masowej

Optymalizacja może być dokonana:

- Na poziomie projektu
- Na poziomie implementacji

Sposoby stosowane na etapie implementacji

- Stosowanie zmiennych statycznych zamiast dynamicznych i lokalnych.
- Umieszczanie zagnieżdżonego kodu zamiast wywoływania procedur.
- Dobór typów o minimalnej, niezbędnej wartości.
- Umieszczanie róznych danych w tych samych zmiennych

Wielu specjalistów jest przeciwna tego rodzaju sztuczkom optymalizacyjnym, ponieważ zyski są bardzo małe w stosunku do zwiększenia stopnia nieczytelności kodu programów.

Optymalizacja projektu (2)

Co może przynieść zasadnicze zyski optymalizacyjne?

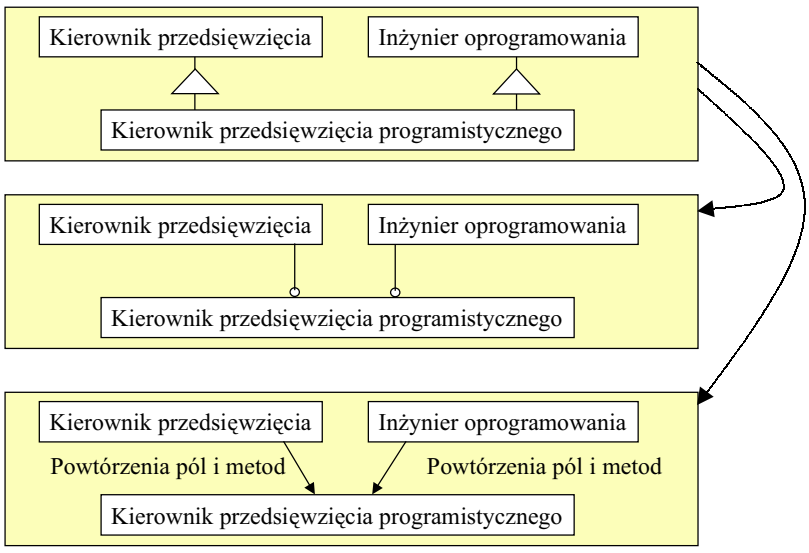
- **Zmiana algorytmu przetwarzania** Np. zmiana algorytmu sortującego poprzez wprowadzenie pośredniego pliku zawierającego tylko klucze i wskaźniki do sortowanych obiektów może przynieść nawet 100-krotny zysk.
- **Wyłowienie "wąskich gardeł"** w przetwarzaniu i optymalizacja tych wąskich gardeł poprzez starannie rozpracowane procedury. Znane jest twierdzenie, że 10% kodu jest wykonywane przez 90% czasu.
- **Zaprogramowanie "waskich gardeł" w języku niższego poziomu**, np. w assemblerze dla programów w C, lub w C dla programów w 4GL.
- **Denormalizacja relacyjnej bazy danych**, łączenie dwóch lub więcej tablic w jedną.
- **Stosowanie indeksów, tablic wskaźników i innych struktur pomocniczych**
- **Analiza mechanizmów buforowania danych** w pamięci operacyjnej i ewentualna zmiana tego mechanizmu (np. zmniekszenie liczby poziomów)

Dostosowanie do ograniczeń i możliwośći środowiska implementacji

Projektant może zetknąć się z wieloma ograniczeniami implementacyjnymi:

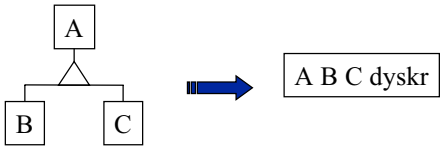
- Brak dziedziczenia wielokrotnego.
- Brak dziedziczenia.
- Brak metod wirtualnych (przesłaniania).
- Brak złożonych atrybutów
- Brak typów multimedialnych

Przykład: jak obejść brak wielo-dziedziczenia

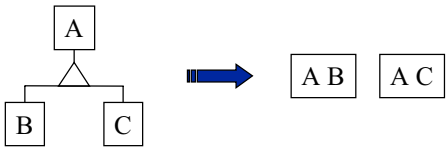


Obejście braku dziedziczenia

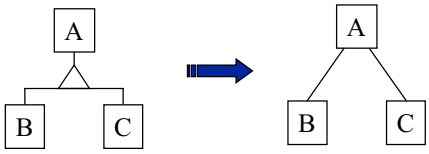
Generalizacja ==> Nadklasa.
Zamiana całej generalizacji (nad- i pod- klasy) na pojedynczą klasę poprzez dodanie atrybutów podklas do atrybutów nadklasy oraz dodanie dodatkowego atrybutu-dyskryminatora wariantu.



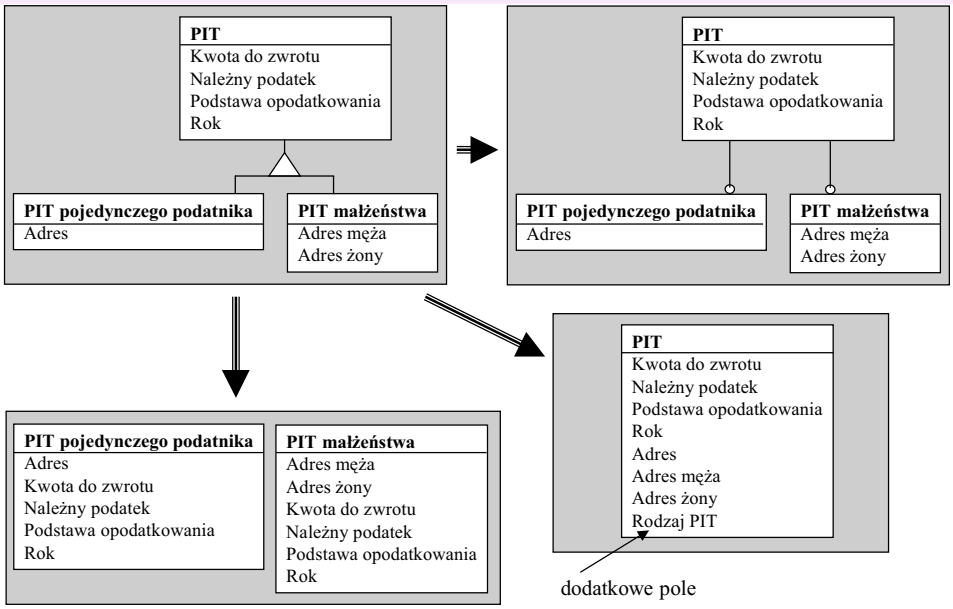
Generalizacja ==> Podklasy.
Usunięcie nadklasy i pozostawienie podklas z propagacją atrybutów nadklasy i dublowaniem powiązań nadklasy dla każdej podklasy



Generalizacja ==> Powiązania.
Zamiana generalizacji na zestaw powiązań łączących nadklasę ze wszystkimi podklasami.



Przykład obejścia braku dziedziczenia

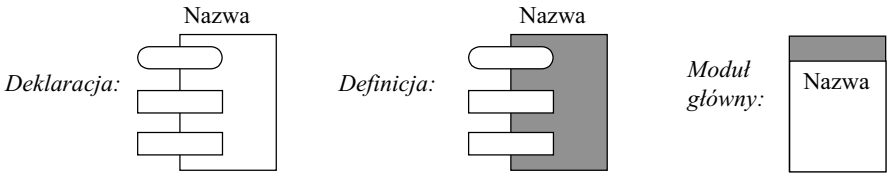


Określenie fizycznej struktury systemu

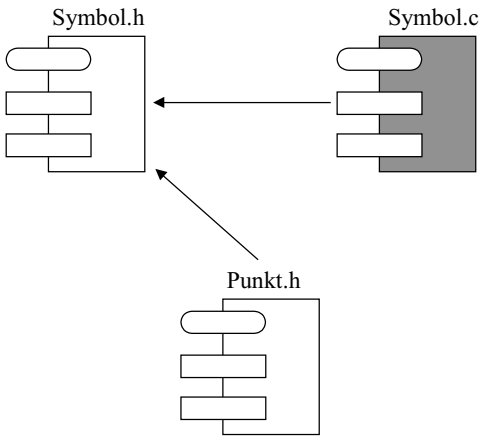
Obejmuje:

- Określenie struktury kodu źródłowego, tj. wyróżnienie plików źródłowych, zależności pomiędzy nimi oraz rozmieszczenie składowych projektu w plikach źródłowych.
- Podział systemu na poszczególne aplikacje.
- Fizyczne rozmieszczenie danych i aplikacji na stacjach roboczych i serwerach.

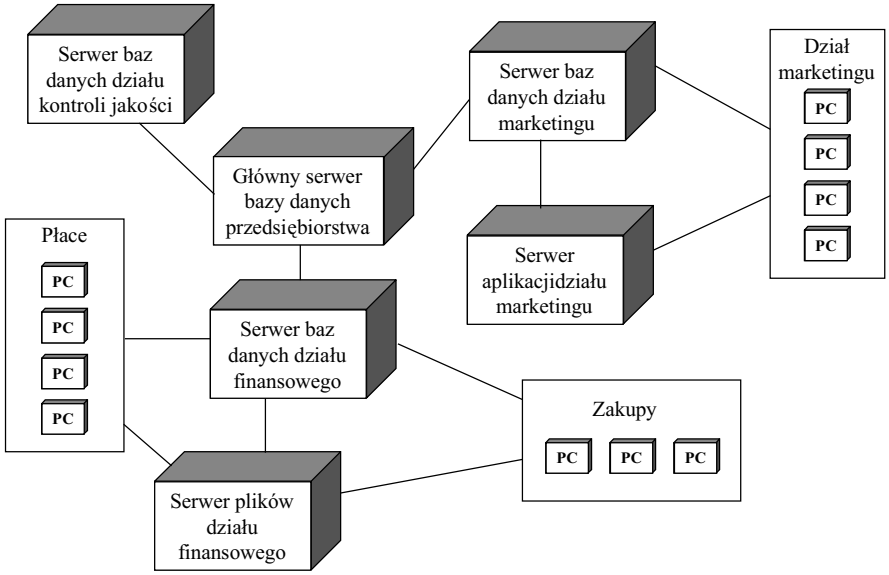
Oznaczenia (Booch)



Przykład zależności kompilacji dla C++



Graficzny opis sprzętowej konfiguracji systemu



Poprawność projektu

Poprawność oznacza, że opis projektu jest zgodny z zasadami posługiwania się notacjami. Nie gwarantuje, że projekt jest zgodny z wymaganiami użytkownika.

Poprawny projekt musi być

* kompletny
* niesprzeczny
* spójny
* zgodny z regułami składniowymi notacji

Kompletność projektu oznaczą, że zdefiniowane są:

* wszystkie klasy
* wszystkie pola (atrybuty)
* wszystkie metody
* wszystkie dane złożone i elementarne

a także że opisany jest sposób realizacji wszystkich wymagań funkcjonalnych.

Spójność projektu oznacza semantyczną zgodność wszystkich informacji zawartych na poszczególnych diagramach i w specyfikacji.

Poprawność diagramów klas i stanów

Diagramy klas:

- Acykliczność związków generalizacji-specjalizacji
- Opcjonalność cyklicznych związków agregacji
- Brak klas nie powiązanych w żaden sposób z innymi klasami. Sytuacja taka może się jednak pojawić, jeżeli projekt dotyczy biblioteki klas, a nie całej aplikacji.
- Umieszczenie w specyfikacji sygnatur metod informacji o parametrach wejściowych, wyjściowych i specyfikacji wyniku

Diagramy stanów:

- Brak stanów (oprócz początkowego), do których nie ma przejścia.
- Brak stanów (oprócz końcowego), z których nie ma wyjścia.
- Jednoznaczność wyjść ze stanów pod wpływem określonych zdarzeń/warunków

Poprawność diagramów przepływu danych

- Brak przepływu danych pomiędzy zbiornikami danych
- Brak przepływu danych pomiędzy interfejsem zewnętrznym i zbiornikiem danych
- Zgodność specyfikacji przepływów danych wpływających i wypływających do poszczególnych procesów posiadających diagramy bardziej szczegółowe z przepływami zaznaczonymi na digramach bardziej szczegółowych.
- Umieszczenie w specyfikacji procesów informacji o odczycie i ustawianiu danych zgodnych z przepływami wpływającymi i wypływającymi z poszczególnych procesów.

Jakość projektu

Metody projektowei stosowane notacjesą w dużym stopniu nieformalnę zaś ich użycie silnie zależy od rodzaju przedsięwzięcia programistycznego

Jest więc dość trudno ocenić jakość projektu w sensie jego adekwatności do procesu konstruowania oprogramowania i stopnia późniejszej satysfakcji użytkowników: stopień spełnienia wymagań, niezawodność, efektywność, łatwość konserwacji i ergonomiczność.

Pod terminem *jakość* rozumie się bardziej szczegółowe kryteria

* spójność
* stopień powiązania składowych
* przejrzystość

Istotne jest spełnienie kryteriów formalnych jakości, które w duzym stopniu rzutują na efektywną jakość, chociaż w żadnym stopniu o niej nie przesądzają. Spełnienie formalnych kryteriów jakości jest warunkiem efektywnej jakości. Nie spełnienie tych kryteriów na ogół dyskwalifikuje efektywną jakość.

Spójność

Spójność opisuje na ile poszczególne części projektu pasują do siebie.

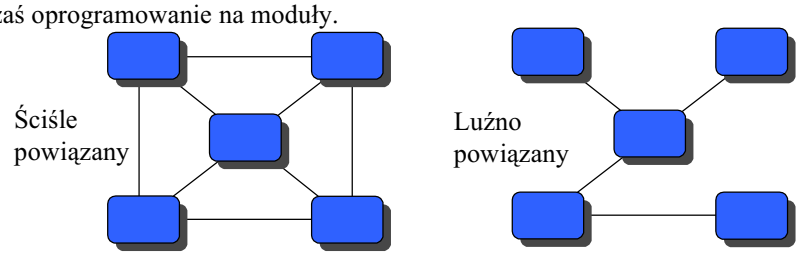
Poziomy spójności (*Constantine i Yourdon*):

Spójność przypadkowa. Podział na moduły (części) wynika wyłącznie z tego, że całość jest za duża (utrudnia wydruk, edycję, itd)
Spójność logiczna. Poszczególne składowe wykonują podobne funkcje, np. obsługa błędów, wykonywanie podbnych obliczeń.
Spójność czasowa. Składowe są uruchamiane w podobnym czasie, np. podczas startu lub zakończenia pracy systemu.
Spójność proceduralna. Składowe są kolejno uruchamiane.
Spójność komunikacyjna. Składowe działają na tym samym zbiorze danych wejściowych i wspólnie produkują zestaw danych wyjściowych
Spójność sekwencyjna. Dane wyjściowe jednej składowej stanowią wejście innej.
Spójność funkcjonalna. Wszystkie składowe są niezbędne dla realizacji jednej tej samej funkcji.

Definicje tych poziomów nie są precyzyjne.

Stopień powiązania składowych

W dobrym projekcie powinno dążyć się do tego, aby stopień powiązania pomiędzy jego składowymi był minimalny. To kryterium określa podział projektu na części zaś oprogramowanie na moduły.



Co to są "powiązania pomiędzy składowymi"?

- Korzystanie przez procesy/moduły z tych samych danych
- Przepływy danych pomiędzy procesami/modułami
- Związki pomiędzy klasami
- Przepływy komunikatów
- Dziedziczenie

Stopień powiązań można oceniać przy pomocy miar liczbowych.

Przejrzystość

Dobry projekt powinien być przejrzysty czyli czytelny, łatwo zrozumiały. Na przejrzystość wpływają następujące czynniki:

- **Odzwierciedlenie rzeczywistości**. Składowe i ich związki pojawiające się w projekcie powinny odzwieciedlać strukture problemu. Ścisły związek projektu z rzeczywistością.
- **Spójność oraz stopień powiązania składowych**. Omówione kryteria także rzutują na zrozumiałość projektu.
- **Zrozumiałe nazewnictwo**
- **Czytelna i pełna specyfikacja**
- **Odpowiednia złożoność składowych.**

Na uwagę zasługuje dziedziczenie oraz przypisanie metod do klas jako czynnik przejrzystości projektu. Pozwala to znacznie uprościć i zdekomponować problem.

Niefunkcjonalne wymagania na fazę projektowania

Wymagania odnośnie wydajności (*performance*)
Wymagania odnośnie interfejsu (protokoły, formaty plików, ...)
Wymagania operacyjne (aspekty ergonomiczne, języki, pomoce)
Wymagania zasobów (ilość procesorów, pojemność dysków, ...)
Wymagania w zakresie weryfikacji (sposoby przeprowadzenia)
Wymagania w zakresie akceptacji i testowania
Wymagania odnosnie dokumentacji
Wymagania odnośnie bezpieczeństwa (*security*)
Wymagania odnośnie przenaszalności
Wymagania odnośnie jakości

- wybór metod projektowania
- decyzje dotyczące ponowneog użycia
- wybór narzędzi
- wybór metod oceny projektu przez ciała zewnętrzne

Wymagania odnośnie niezawodności
Wymagania odnośnie podatności na zarządzanie (*maintenance*)
Wymagania odnośnie odporności na awarie (*safety*)

Kluczowe czynniki sukcesu fazy projektowania

- **Wysoka jakość modelu projektowego**
- **Dobra znajomość środowiska implementacji**
- **Zachowanie przyjętych standardów**, np. konsekwentne stosowanie notacji i formularzy.
- **Sprawdzenie poprawności projektu** w ramach zespołu projektowego
- **Optymalizacja projektu** we właściwym zakresie. Powinna być ograniczona do istotnych, krytycznych miejsc
- **Poddanie projektu ocenie przez niezależne ciało** oceniające jego jakość pod względem formalnym i merytorycznym.

Podstawowe rezultaty fazy projektowania

- Poprawiony dokument opisujący wymagania
- Poprawiony model
- Uszczegółowiona specyfikacja projektu zawarta w słowniku danych
- Dokument opisujący stworzony projekt składający się z (dla obiektowych):
    - diagramu klas
    - diagramów interakcji obiektów
    - diagramów stanów
    - innych diagramów, np. diagramów modułów, konfiguracji
    - zestawień zawierających:
        - definicje klas
        - definicje atrybutów
        - definicje danych złożonych i elementarnych
        - definicje metod
- Zasoby interfejsu użytkownika, np. menu, dialogi
- Projekt bazy danych
- Projekt fizycznej struktury systemu
- Poprawiony plan testów
- Harmonogram fazy implementacji

Narzędzia CASE w fazie projektowania

Tradycyjnie stosuje się Lower-CASE.
Zastosowania podobne jak w fazie analizy:

- Edytor notacji graficznych
- Narzędzia edycji słownika danych
- Generatory raportów
- Generatory dokumentacji technicznej
- Narzędzia sprawdzania jakości projektu

Narzędzia CASE powinny wspomagać proces uszczegóławiania wyników analizy. Powinny np. automatycznie dodawać atrybuty realizujące związki pomiędzy klasami. Powinny ułatwiać dostosowanie projektu do środowiska implementacji.

Powinna istnieć możliwość automatycznej transformacji z modelu obiektów na schemat relacyjnej bazy danych.

Niektóre narzędzia CASE umożliwiają projektowanie interfejsu użytkownika.

Narzędzia inżynierii odwrotnej (reinżynierii, *reengineering*), dla odtworzenia projektu na podstawie istniejącego kodu.