INŻYNIERIA OPROGRAMOWANIA

wykład 6:
PROJEKTOWANIE

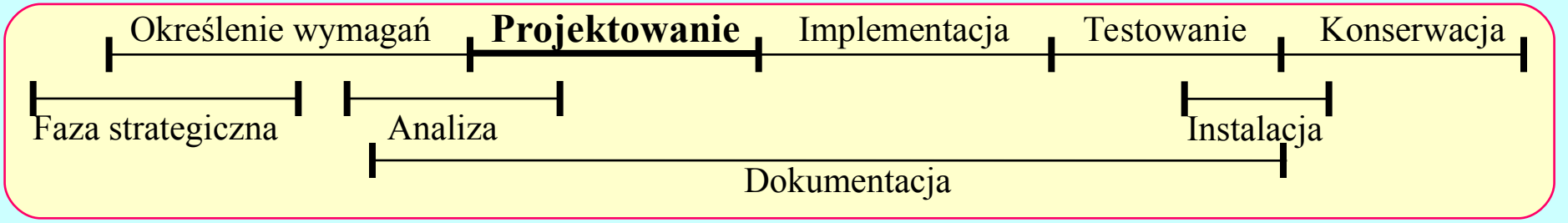
dr inż. Leszek Grocholski

Zakład Inżynierii Oprogramowania
Instytut Informatyki
Uniwersytet Wrocławski

Plan wykładu

1. Zadania wykonywane w fazie projektowania
2. Projektowanie składowych nie/związanych z dziedziną problemu
3. Projektowanie Interfejsu z użytkownikiem
4. Projektowanie składowej zarządzania danymi
5. Optymalizacja projektu
6. Dostosowanie do ograniczeń i możliwości środowiska implementacji
7. Określenie fizycznej struktury systemu
8. Graficzny opis sprzętowej konfiguracji systemu
9. Poprawność i jakość projektu
10. Wymagania niefunkcjonalne dla fazy projektowania
11. Podstawowe rezultaty fazy projektowania

Projektowanie



Celem projektowania jest opracowanie szczegółowego opisu implementacji systemu. Tak szczegółowego aby na jego podstawie można było wykonać system.

W odróżnieniu od analizy, w projektowaniu dużą role odgrywa środowisko implementacji. Projektanci muszą więc posiadać dobrą znajomość języków, bibliotek i narzędzi stosowanych w trakcie implementacji.

Dążenie do tego, aby struktura projektu zachowała ogólną strukturę modelu stworzonego w poprzednich fazach (analizie).
Niewielkie zmiany w dziedzinie problemu powinny implikować niewielkie zmiany w projekcie.

Zadania wykonywane w fazie projektowania

1. Uszczegółowienie wyników analizy. Projekt musi być wystarczająco szczegółowy aby mógł być podstawą implementacji.
   Stopień szczegółowości zależy od poziomu zaawansowania programistów.

2. Projektowanie składowych dot. wymagań funkcjonalnych (dziedzina)

3. Projektowanie składowych systemów nie związanych z dziedziną problemu

4. Projektowanie rozwiązań zapewniających spełnienie innych wymagań niefunkcjonalnych

3. Optymalizacja systemu

4. Dostosowanie do ograniczeń i możliwości środowiska implementacji

5. Określenie logicznej fizycznej struktury systemu

6. Weryfikacja,wryfikacja zgodności z wymaganiami i założeniami

Projektowanie składowych systemu nie związanych z dziedziną problemu

Składowa dot. dziedziny problemu – to co system robi z perspektywy klienta. Projekt skonstruowany przez uszczegółowienie modelu opisuje składowe programu odpowiedzialne za realizację podstawowych zadań systemu.

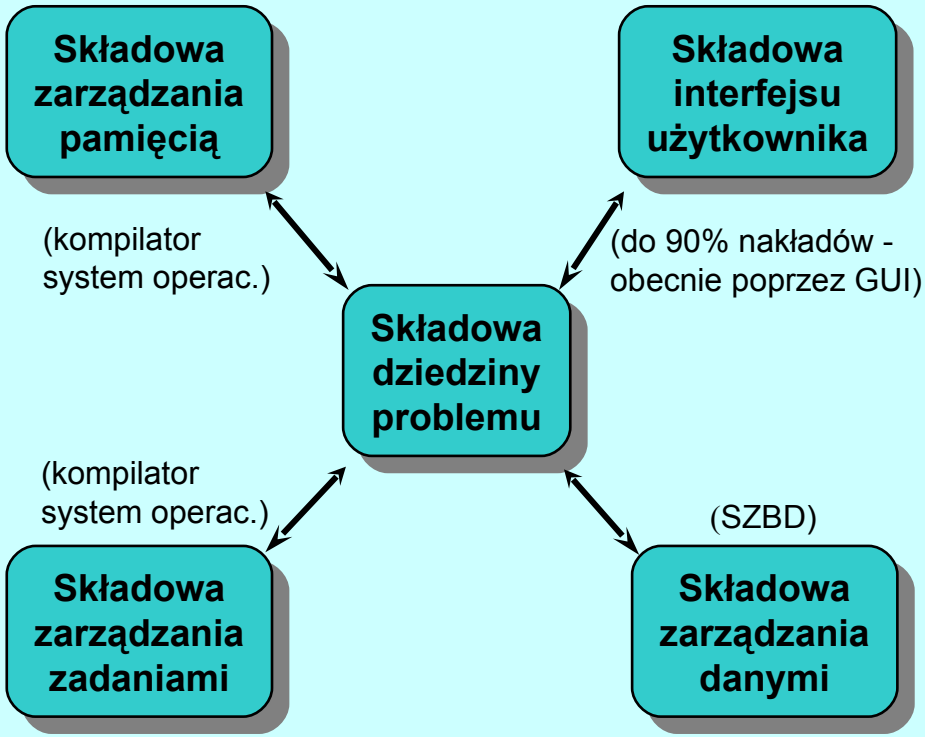
Gotowe oprogramowanie musi się jednak składać z dodatkowych składowych:

- składowej interfejsu użytkownika

- 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: Microsoft Visual Studio, PowerBuilder Desktop

Oraz przeróżne framwork'i.

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 interfejsu użytkownika

W ostatnich latach nastąpił gwałtowny rozwój narzędzi graficznych służących do tego celu różnych frameworków.

- 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.

Przykład: komendy „unixowe”.

Uwaga: Często szybszy od niż interfejs pełnoekranowy (okienkowy).

W pełnoekranowym środowisku okienkowym

Tworzenie ma sens dla dużych systemów.

Wygodny dla początkujących i średnio zaawansowanych użytkowników

Wprowadzanie i wyprowadzanie danych

Wprowadzanie (IN) przez użytkownika:

- Podawanie parametrów poleceń w przypadku systemów z linią komend
- Wprowadzanie danych w odpowiedzi na zaproszenie systemu
- Wprowadzanie danych w dialogach – formatki dialogowe

Wyprowadzanie (OUT) przez system:

- Wyświetlanie informacji w formatkach dialogowych
- Wyświetlanie i/lub wydruki raportów (analizy, planowanie…)
- Graficzna prezentacja danych (bardzo użyteczne)

Prototyp interfejsu użytkownika może powstać już w fazie określenia wymagań.
Pomaga uzyskać odpowiedz na pytanie: czy tak właśnie chce pracować użytkownik
Systemy zarządzania interfejsem użytkownika pozwalają na wygodną budowę prototypów oraz wykorzystanie prototypu w końcowej implementacji.

Zasady projektowania interfejsu użytkownika (1)

Wytyczne projektowania interfejsu użytkownika – np. 10 heurystyk Nielsena

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 wyglądać 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.

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

3. 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, sekund do przewidywanego zakończenia, „termometru”, itd.).

Zasady projektowania interfejsu użytkownika (2)

4. 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.

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

6. 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)

7. 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.

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

Reguła Millera 7 +/- 2:
Człowiek może się jednocześnie skupić na 5 - 9 elementach.
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.

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

- Wysoka efektywność i stabilność
- Bezpieczeństwo i prywatność danych, spójność i integralność przetwarzania
- Automatyczne sprawdzanie 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)

Spójność: zgodność danych z rzeczywistością lub z oczekiwaniami użytkownika.

Integralność: poprawność danych w sensie ich organizacji i budowy.

Wady relacyjnych baz danych

-Konieczność przeprowadzenie nietrywialnych odwzorowań przy przejściu z modelu pojęciowego (np. obiektowego w UML) na strukturę relacyjną.
→ Hibernate

- Ustalony format krotki powodujący trudności przy polach zmiennej długości.

- Trudności (niesystematyczność) reprezentacji dużych wartości (grafiki, plików tekstowych, itd.)

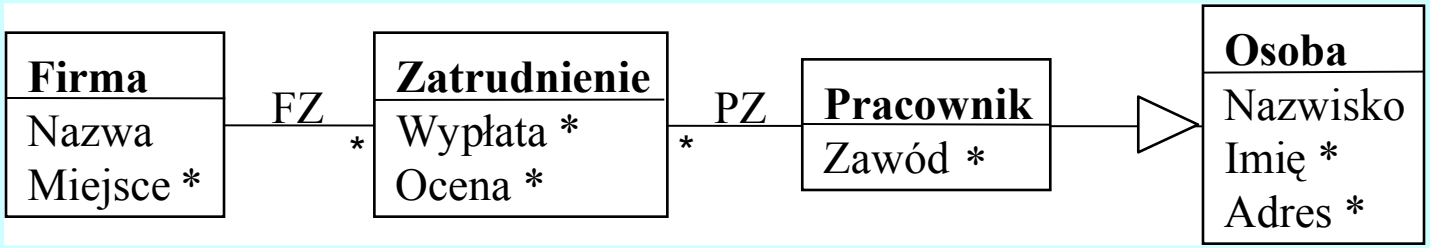
- W niektórych sytuacjach - duże narzuty na czas przetwarzania

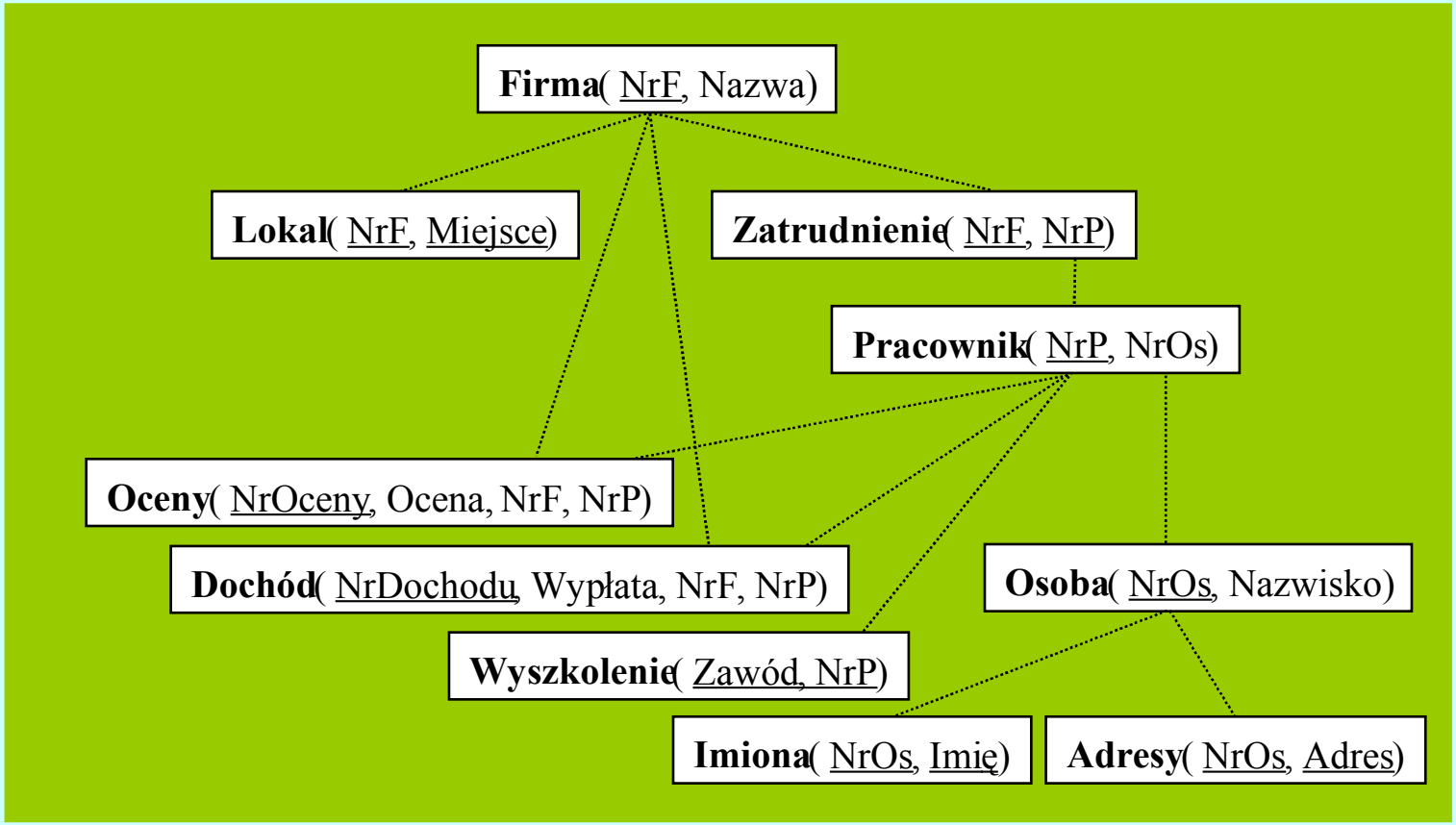
- Niedopasowanie interfejsu dostępu do bazy danych (SQL) do języka programowania (np. C), określana jako “niezgodność impedancji”.

- Brak możliwości rozszerzalności typów (zagnieżdżania danych)

- Brak systematycznego podejścia do informacji proceduralnej (metod)

Niezgodność modelu obiektowego i relacyjnego





Optymalizacja projektu (1)

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

- 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 (lokalnych)
- Umieszczanie zagnieżdżonego kodu zamiast wywoływania procedur.
- Dobór typów o minimalnej, niezbędnej wartości.
- Optymalizacja zapytań w SQL.

Uwaga:
Wielu specjalistów jest przeciwna sztuczkom optymalizacyjnym: zyski są bardzo małe (o ile w ogóle są) w stosunku do zwiększenia nieczytelności kodu.

Optymalizacja projektu (2)

o 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 20% kodu jest wykonywane przez 80% czasu.

- **Zaprogramowanie "wąskich gardeł" w języku niższego poziomu**, np. 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. zmniejszenie liczby poziomów)

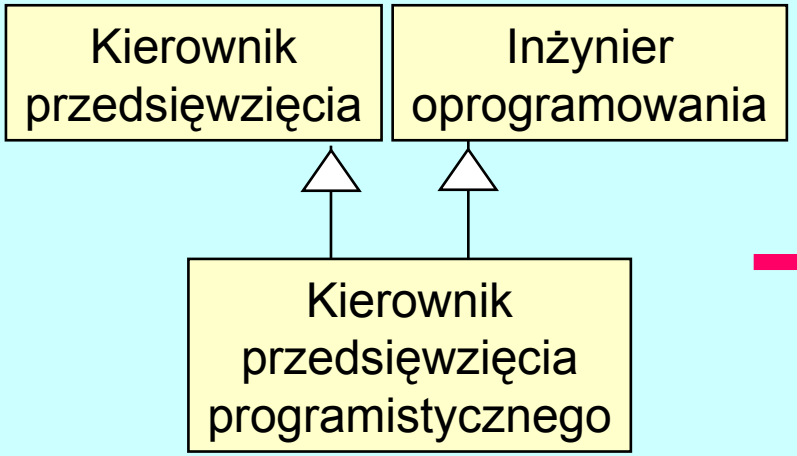
- Optymalizacja zapytań SQL

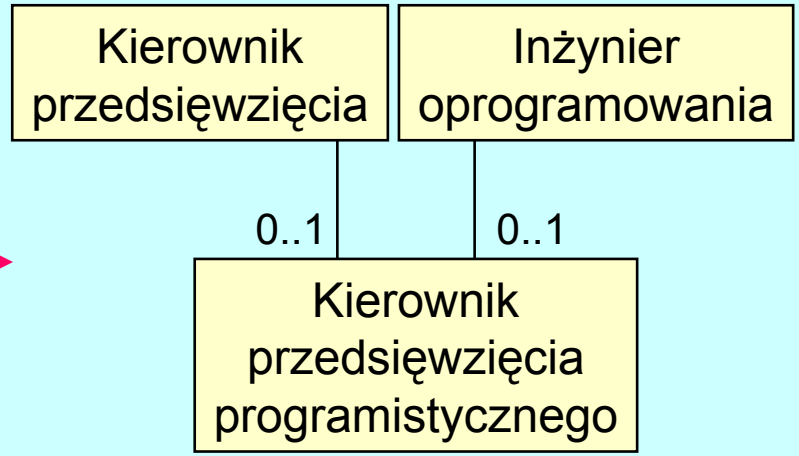
Dostosowanie do ograniczeń i możliwości środowiska implementacji

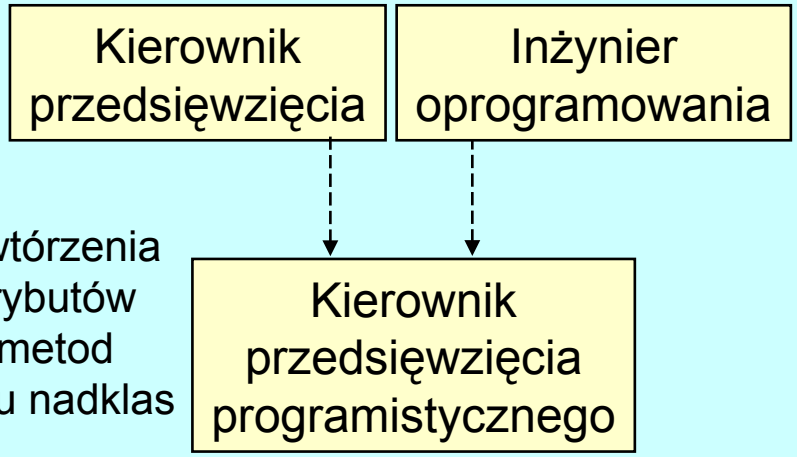
Projektant może zetknąć się z wieloma ograniczeniami implementacyjnymi, np:

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

Przykład: obejście braku wielodziedziczenia







Powtórzenia atrybutów i metod z obu nadklas

Na ogół, „obejście” braku pewnej cechy modelu pojęciowego w przyjętym środowisku implementacyjnym jest związane z zasadniczymi wadami (jakkolwiek firmy komputerowe na wszystkie sposoby próbują te wady zbagatelizować).

Określenie fizycznej struktury systemu

Obejmuje:

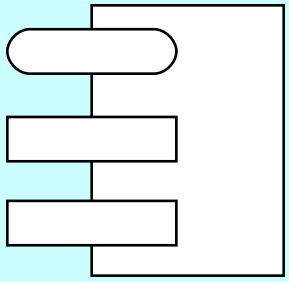
- Określenie struktury kodu źródłowego, tj. wyróżnienie plików źródłowych, zależności pomiędzy nimi oraz rozmieszczenie składowych projektu w plikach źródłowych.

- Podział systemu na poszczególne aplikacje.

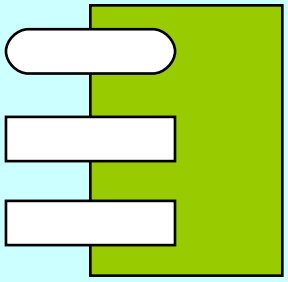
- Fizyczne rozmieszczenie danych i aplikacji na stacjach roboczych i serwerach.

Oznaczenia (Booch)

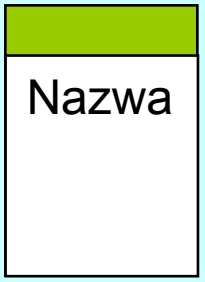
Deklaracja:



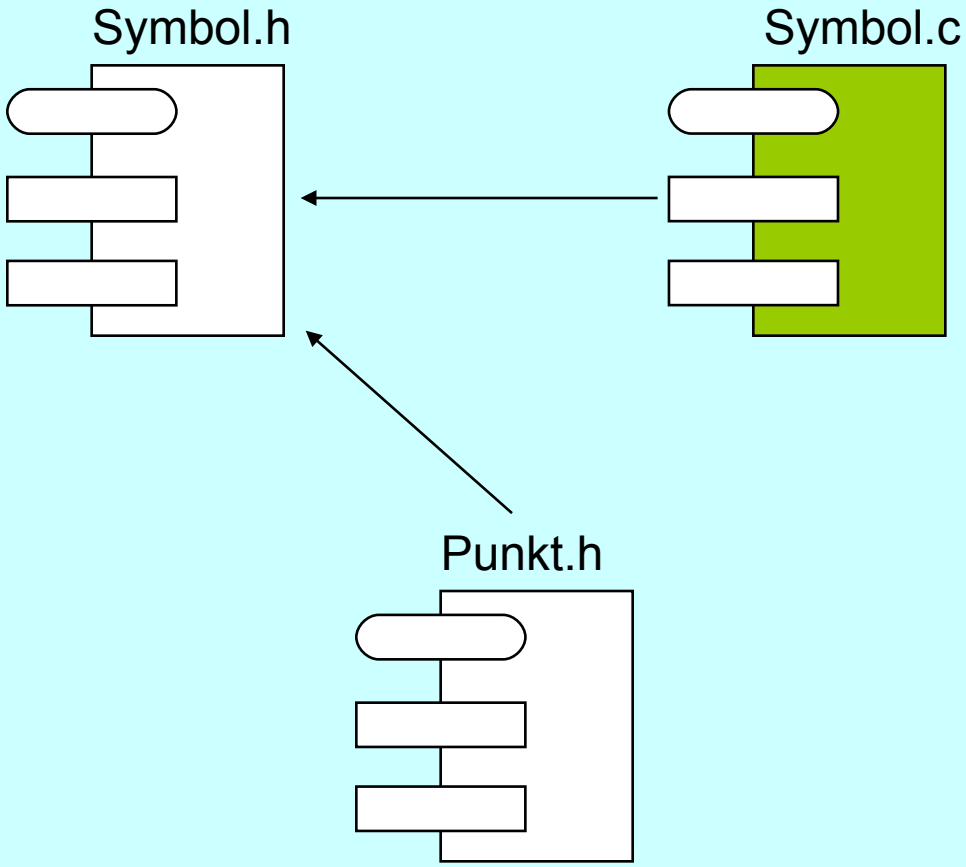
Definicja:



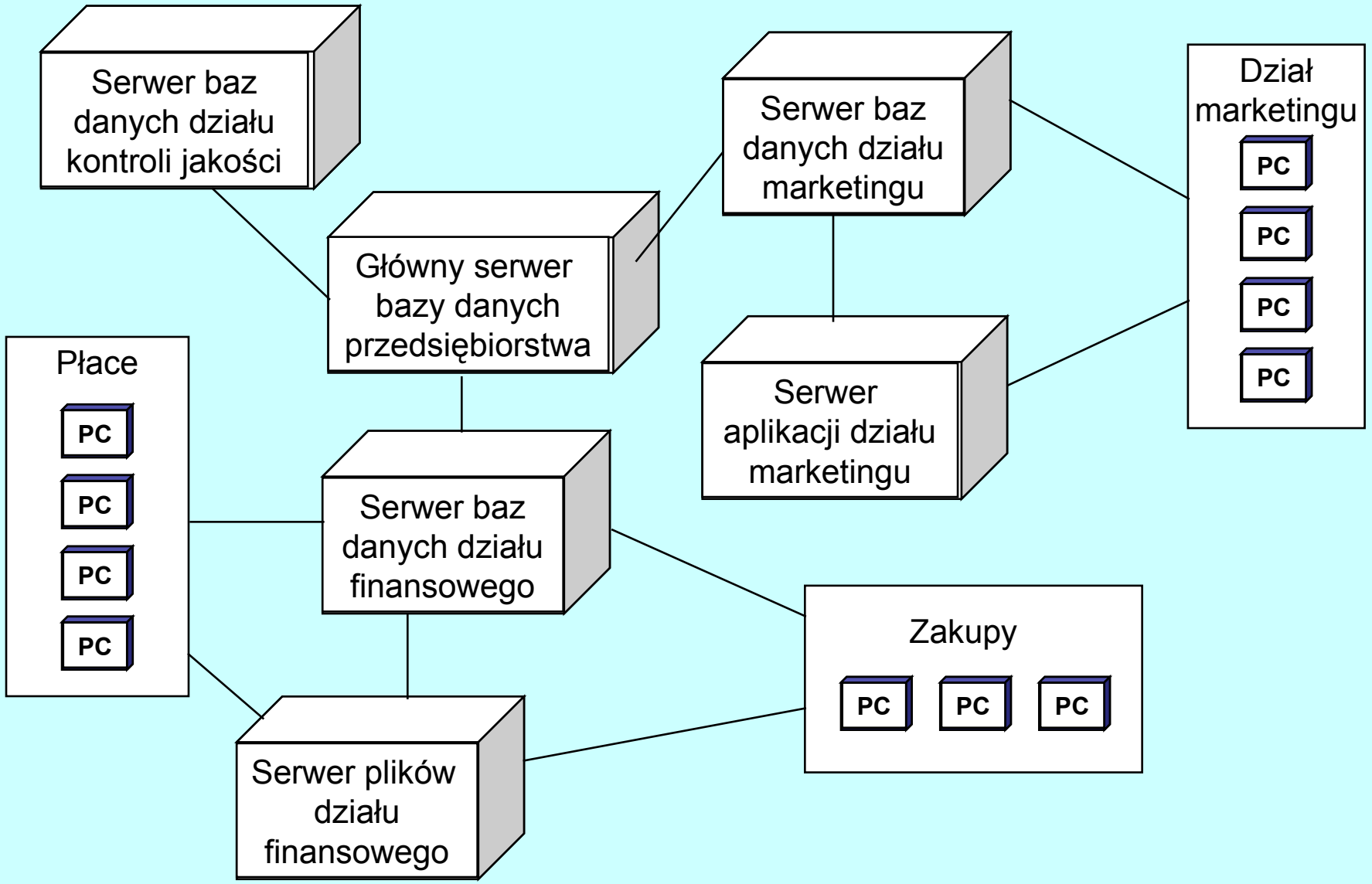
Moduł główny:



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

Np. Kompletność projektu KLAS oznacza, 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

Jakość projektu

Metody projektowe i stosowane notacje są w dużym stopniu nieformalne, 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 dużym 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.
Istotne staje się kryterium podziału projektu na części.
W zależności od tego kryterium, możliwe jest wiele rodzajów spójności.

Kryteria podziału projektu (i rodzaje spójności):

Podział przypadkowy. Podział na moduły (części) wynika wyłącznie z tego, że całość jest za duża (utrudnia wydruk, edycję, itd)

Podział logiczny. Poszczególne składowe wykonują podobne funkcje, np. obsługa błędów, wykonywanie podobnych obliczeń.

Podział czasowy. Składowe są uruchamiane w podobnym czasie, np. podczas startu lub zakończenia pracy systemu.

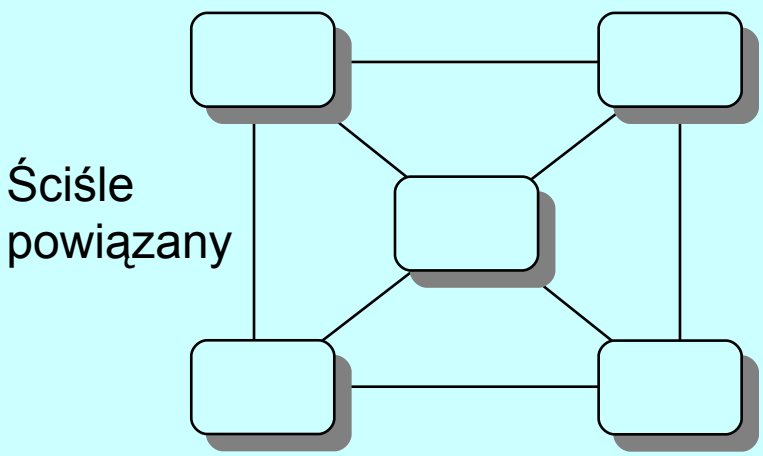
Podział proceduralny (sekwencyjny). Składowe są kolejno uruchamiane. Dane wyjściowe jednej składowej stanowią wejście innej

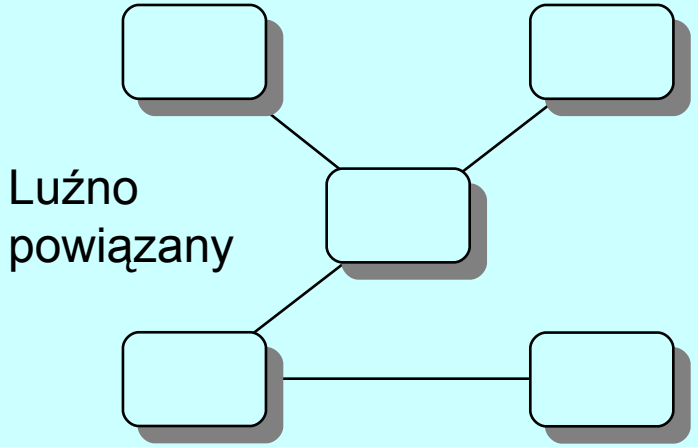
Podział komunikacyjny. Składowe działają na tym samym zbiorze danych wejściowych i wspólnie produkują zestaw danych wyjściowych

Podział funkcjonalny. Wszystkie składowe są niezbędne dla realizacji jednej tej samej funkcji.

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 (kohezja).

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 odzwierciedlać strukturę problemu. Ścisły związek projektu z rzeczywistością.

- Spójność oraz stopień powiązania składowych

- Zrozumiałe nazewnictwo

- Czytelna i pełna specyfikacja

- Odpowiednia złożoność składowych na danym poziomie abstrakcji

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

Wymagania niefunkcjonalne dla fazy projektowania

- Wymagania odnośnie wydajności
- 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 odnośnie dokumentacji
- Wymagania odnośnie bezpieczeństwa
- Wymagania odnośnie przenaszalności
- Wymagania odnośnie jakości
    - wybór metod projektowania
    - decyzje dotyczące ponownego użycia
    - wybór narzędzi
    - wybór metod oceny projektu przez ciała zewnętrzne

- Wymagania odnośnie niezawodności
- Wymagania odnośnie podatności na pielęgnację (*maintenance*)
- Wymagania odnośnie odporności na awarie

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 m. 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 (projektowanie struktur logicznych).

- 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 (*reverse engineering*), dla odtworzenia projektu na podstawie istniejącego kodu.

Zawartość dokumentu projektowego

Celem (**Dokładnego**) **Dokumentu Detalicznego Projektu** (**DDP**) jest szczegółowy opis rozwiązania problemu określonego w dokumencie wymagań na oprogramowanie. DDP musi uwzględniać wszystkie wymagania. Powinien być wystarczająco detaliczny aby umożliwić implementację i pielęgnację kodu.

Styl DDP powinien być systematyczny i rygorystyczny. Język i diagramy użyte w DDP powinny być klarowne. Dokument powinien być łatwo modyfikowalny.

Struktura DDP powinna odpowiadać strukturze projektu oprogramowania. Język powinien być wspólny dla całego dokumentu. Wszystkie użyte terminy powinny być zdefiniowane i użyte w zdefiniowanym znaczeniu.

Zasady wizualizacji diagramów:

- wyróżnienie ważnych informacji,
- wyrównanie użytych oznaczeń,
- diagramy powinny być czytane od lewej do prawej oraz z góry do dołu,
- podobne pozycje powinny być zorganizowane w jeden wiersz, w tym samym stylu,
- symetria wizualna powinna odzwierciedlać symetrię funkcjonalną,
- należy unikać przecinających się linii i nakładających się oznaczeń i rysunków,
- należy unikać nadmiernego zagęszczenia diagramów.

Modyfikowalność, ewolucja, odpowiedzialność

Modyfikowalność dokumentu. Tekst, diagramy, wykresy, itd. powinny być zapisane w formie, którą można łatwo zmodyfikować. Należy kontrolować nieprzewidywalne efekty zmian, np. lokalnych zmian elementów, które są powtórzone w wielu miejscach dokumentu i powiązane logicznie.

Ewolucja dokumentu. DDP powinien podlegać rygorystycznej kontroli, szczególnie jeżeli jest tworzony przez zespół ludzi. Powinna być zapewniona formalna identyfikacja dokumentów, ich wersji oraz ich zmian. Wersje powinny być opatrzone unikalnym numerem identyfikacyjnym i datą ostatniej zmiany. Powinno istnieć centralne miejsce, w którym będzie przechowywana ostatnia wersja.

Odpowiedzialność za dokument. Powinna być jednoznacznie zdefiniowana. Z reguły, odpowiedzialność ponosi osoba rozwijająca dane oprogramowanie. Może ona oddelegować swoje uprawnienia do innych osób dla realizacji konkretnych celów związanych z tworzeniem dokumentu.

Medium dokumentu Należy przyjąć, że wzorcowa wersja dokumentu będzie w postaci elektronicznej, w dobrze zabezpieczonym miejscu. Wszelkie inne wersje, w tym wersje papierowe, są pochodną jednej, wzorcowej wersji.

Dalsze zalecenia odnośnie DDP (Dokumentu Detalicznego Projektu)

- DDP jest centralnym miejscem, w którym zgromadzone są wszystkie informacje odnośnie budowy i działania oprogramowania.

- DDP powinien być zorganizowany w taki sam sposób, w jaki zorganizowane jest oprogramowanie.

- DDP powinien być kompletny, odzwierciedlający wszystkie wymagania zawarte w Specyfikacji wymagań.

- Materiał, który nie mieści się w podanej zawartości dokumentu, powinien być załączony jako dodatek.

- Nie należy zmieniać numeracji punktów. Jeżeli jakiś punkt nie jest zapełniony, wówczas należy pozostawić jego tytuł, zaś poniżej zaznaczyć "Nie dotyczy."

Zawartość DDP (1)

Informacja organizacyjna

a - Streszczenie
b - Spis treści
c - Formularz statusu dokumentu
d - Zapis zmian w stosunku do ostatniej wersji

CZĘŚĆ 1 - OPIS OGÓLNY

1. **WPROWADZENIE**

Opisuje cel i zakres, określa użyte terminy, listę referencji oraz krótko omawia dokument.

1.1. Cel Opisuje cel DDP oraz specyfikuje przewidywany rodzaj jego czytelnika.

1.2. Zakres

Identyfikuje produkt programistyczny będący przedmiotem dokumentu, objaśnia co oprogramowanie robi (i ewentualnie czego nie robi) oraz określa korzyści, założenia i cele. Opis ten powinien być spójny z dokumentem nadrzędnym, o ile taki istnieje.

1.3. Definicje, akronimy, skróty

1.4. Odsyłacze

1.5. Krótkie omówienie

2. **STANDARDY PROJEKTU, KONWENCJE, PROCEDURY**

2.1. Standardy projektowe

2.2. Standardy dokumentacyjne

2.3. Konwencje nazwowe

2.4. Standardy programistyczne

2.5. Narzędzia rozwijania oprogramowania

Zawartość DDP (2)

CZĘŚĆ II - SPECYFIKACJA KOMPONENTÓW

n [IDENTYFIKATOR KOMPONENTU]
n.1. Typ
n.2. Cel
n.3. Funkcja
n.4. Komponenty podporządkowane
n.5. Zależności
n.6. Interfejsy
n.7. Zasoby
n.8. Odsyłacze
n.9. Przetwarzanie
n.10. Dane

Dodatek A. Wydruki kodu źródłowego
Dodatek B. Macierz zależności pomiędzy zbiorem wymagań i zbiorem komponentów oprogramowania
Dodatek C. Uzasadnienie spełnienia wymagań niefunkcjonalnych

INŻYNIERIA OPROGRAMOWANIA

Dziękuję za uwagę