Politechnika Łódzka
Instytut Informatyki

INŻYNIERSKA PRACA DYPLOMOWA

MAPOWANIE OBIEKTOWO-RELACYJNE W JĘZYKU C++

Wydział: Fizyki Technicznej, Informatyki i Matematyki Stosowanej
Promotor: dr inż. Arkadiusz Tomczyk
Dyplomant: Marcin Maciaszczyk
Nr albumu: 165466
Kierunek: Informatyka
Specjalność: Inżynieria Oprogramowania i Analiza Danych

Łódź, 10 maja 2014r.

Instytut Informatyki
90-924 Łódź, ul. Wólczańska 215, budynek B9
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

Spis treści

Rozdział 1

Wstęp

1.1 Uzasadnienie wyboru tematu

Wraz z rozwojem informatyki proces tworzenia oprogramowania wymagał od informatyków co raz większej wiedzy w poszczególnych dziedzinach takich jak na przykład bazy danych czy sieci komputerowe. Ciężko jest być specjalistą w każdej z tych dziedzin, dlatego też podczas tworzenia oprogramowania programiści co raz częściej sięgają po różnego rodzaju narzędzia programistyczne, które mają za zadanie ułatwić im pracę.

Przykładem takich narzędzi są aplikacje szkieletowe wykorzystywane jako fundamenty dla tworzonych aplikacji czy też biblioteki udostępniające zestawy określonych funkcji. Oczywiście nie zawsze mamy możliwość skorzystania z wspominanych narzędzi, jednak jeśli taka istnieje warto wziąć to pod uwagę podczas fazy planowania tworzenia oprogramowania, ponieważ decydując się na korzystanie z nich jesteśmy w stanie zaoszczędzić sporo czasu, a także uniknąć wielu błędów związanych z nieznajomością danej dziedziny.

Wraz z kolegą z tego samego roku studiów – Sebastianem Florkiem, zdecydowaliśmy się w ramach pisamia pracy dyplomowej na wspólne stworzenie aplikacji szkieletowej, która będzie realizowała mapowanie obiektowo-relacyjne w języku C++. Wybór języka programowania związany był z dość dobrą jego znajomością, a także z pewnym doświadczeniem w programowaniu w tym języku nabytym w trakcie studiów. Mapowanie obiektowo-relacyjne jest to obecnie zagadnienie bardzo powszechne, szczególnie w językach programowania takich jak C# czy Java. Programiści C++ nie mają już tak dużego wyboru wśród dostępnych narzędzi służących do mapowania obiektowo-relacyjnego, co tłumaczy nasz wybór.

Podział naszej pracy zostanie opisany w dalszych rozdziałach, jednak na wstępie warto zaznaczyć, że tematem pracy Sebastiana jest generator opisu mapowania

obiektowo-relacyjnego, który jest wstępnym etapem pracy naszej aplikacji szkieletowej. Celem wspomnianego generatora jest wygenerowanie pliku projektu, a także plików nagłówkowych oraz klas na podstawie istniejącej już bazy danych. Moim zadaniem będzie samo mapowanie obiektowo-relacyjne, które będzie się opierało na wcześniej wygenerowanych plikach.

1.2 Problematyka i zakres pracy

Programowanie obiektowe jest obecnie jednym z najpopularniejszych paradygmatów programowania, a pojęcia takie jak klasa czy obiekt znane są wszystkim programistom. Podobnie jest z relacyjnym modelem organizacji baz danych i terminami takimi jak relacja czy krotka. Chcąc wykorzystać oba te podejścia w jednej aplikacji musimy zadbać o obustronną konwersję pomiędzy danymi z tabel relacyjnej bazy danych a obiektami aplikacjii. Tym właśnie zajmuje się mapowanie obiektowo-relacyjne, które wraz z tworzeniem aplikacji szkieletowych w języku C++ jest główną problematyką niniejszej pracy.

W tym momencie powstaje pytanie czy na prawdę warto korzystać z bibliotek i aplikacji szkieletowych służących do mapowania obiektowo relacyjnego? Odpowiedź nie jest jednoznaczna w wszystkich przypadkach, ale warto wymienić jego podstawowe wady i zalety, których głębsza analiza znajduje się w dalszej części pracy. Zacznijmy od zalet wykorzystania narzędzi ORM[1]:

- Znacznie zredukowana ilość pracy wymagana na oprogramowanie dostępu do bazy danych.
- Nieobowiązkowa znajomość języka SQL[2]. W celu tworzenia zapytań do bazy danych korzystamy z interfejsu udostępnianego przez daną bibliotekę czy też aplikację szkieletową.
- Uniezależnienie się od rodzaju systemu zarządzania bazą danych, możemy korzystać równie dobrze z MySQL, Oracle, PostgreSQL czy też Microsoft SQL Server niekoniecznie posiadając o nich rozbudowaną wiedzę.
- Aby skorzystać z transakcji, połączenia z bazą danych a także wieu innych funkcjonalności baz danych wystarczy zazwyczaj wywołać pojedyńczą metodę.
- Cały model danych przechowywany jest w jednym miejscu oraz nie jest ściśle związany z wykorzystywanym systemem zarządzania bazą danych dzięki czemu

[1] mapowanie obiektowo-relacyjne (ang. Object-Relational Mapping)
[2] strukturalny język zapytań (ang. Structured Query Language)

łatwiej jest nam wprowadzać kolejne modyfikacje w kodzie oraz nawet zmieniać system zarządzania bazą danych na inny.

Wszędzie gdzie pojawiają się zalety mamy też do czynienia z wadami, nie inaczej jest w tym przypadku:

- Konfiguracja jest najczęściej skomplikowana i wymaga sporo czasu.
- Podobnie jest z użytkowaniem, aby robić to w sposób optymalny musimy dobrze poznać daną bibliotekę czy też aplikację szkieletową.
- Proste zapytania są obsługiwane bardzo sprawnie, jednak gdy przetwarzamy duże ilości złożonych zapytań wydajność nie dorówna nigdy zapytaniom napisanym przez specjalistę znającego język SQL.
- Abstrakcja wprowadzona przez narzędzia ORM może okazać się uciążliwa, ponieważ nie zawsze zdajemy sobie sprawę z tego co dzieje się za kulisami w trakcie wykonywania poszczególnch operacji.

Naszym głównym celem jest uczynienie Qubica dobrą alternatywą dla nielicznych, ale istniejących już bibliotek oraz aplikacji szkieletowych realizujących mapowanie obiektowo-relacyjne w języku C++. Wszystkie nam obecnie znane rozwiązania są dostępne za darmo, jednak nie udostępniają one generatora opisu będącego w stanie wygenerować cały projekt aplikacji a ich interfejsy nie należą do intuicyjnych. Wprowadzenie generatora oraz intuicyjnego interfejsu użytkownika powinno uczynić Qubica istotną alternatywą dla istniejących już narzędzi.

Analizę istniejących rozwiązań przedstawia kolejny rozdział a zestawienie z rezultatami naszej pracy znajduje się natomiast w końcowej części pracy, gdzie zostaną przedstawione uzyskane przez nas wyniki oraz podsumowanie wykonanej przez nas pracy.

1.3 Cele pracy

Do najważniejszych celów niniejszej pracy dyplomowej należą:

- Analiza istniejących bibliotek oraz aplikacji szkieletowych realizujących mapowanie obiektowo-relacyjne w języku C++.
- Stworzenie własnej aplikacji szkieletowej realizującej mapowanie obiektowo-relacyjne w języku C++.

- Analiza porównawcza szybkości działania przykładowej aplikacji stworzonej w oparciu o Qubic a o inne istniejące narzędzia.
- Analiza porównawcza ilości kodu przykładowej aplikacji stworzonej w oparciu o Qubic a o inne istniejące narzędzia.

Do celów części praktycznej należą:

- Stworzenie intuicyjnego interfejsu użytkownika – aby sprawdzić w jakim stopniu udało się zrealizować założenia zostanie przeprowadzony test polegający na napisaniu tej samej aplikacji wykorzystującej Qubica oraz inne aplikacje szkieletowe i bibioteki, a następnie porównaniu ilości linii kodu stworzonych aplikacji.
- Stworzenie generatora opisu mapowania obiektowo-relacyjnego – jest to przedmiotem pracy Sebastiana, naszym wspólnym celem jest integracja obu modułów.
- Poprawne zrealizowanie założeń mapowania obiektowo-relacyjnego, a także jak najlepsza optymalizacja zapytań – aby sprawdzić w jakim stopniu udało się zrealizować założenia zostanie przeprowadzony test polegający na napisaniu tej samej aplikacji wykorzystującej Qubica oraz inne aplikacje szkieletowe i bibioteki, a następnie porównaniu ich wydajności.
- Uczynienie konfiguracji Qubica jak najprostszą.

1.4 Metoda badawcza

Studia literaturowe

Podstawowa literatura wykorzystana podczas pisania niniejszej pracy dotyczy specyfikacji języka C++ oraz zapytań w języku SQL, są to powszechne zagadnienia, więc wybór pozycji książkowych jest spory. Materiały dotyczące samego mapowania obiektowo-relacyjnego pochodzą najczęściej z źródeł elektronicznych w języku angielskim, co jest związane z faktem, że wiedza ta wciąż się rozwija.

Analiza istniejących rozwiązań

Stworzenie własnej aplikacji szkieletowej

Praktyka jest najczęściej najlepszą z dostępnych metod nauki i to właśnie podczas tworzenia własnej aplikacji szkieletowej realizującej mapowanie obiektowo-relacyjne można się najbardziej z danym tematem zapoznać. Wszystkie problemy,

które pojawiały się poczas pisania Qubica musiały zostać w pewien sposób rozwiązane i to właśnie analiza tych problemów i ich rozwiązywanie było główną metodą badawczą wykorzystaną podczas pisania niniejszej pracy.

Analiza porównawcza oraz testy

1.5 Przegląd literatury w dziedzinie

Literatura dotycząca języka C++

Literatura dotycząca języka SQL

Literatura dotycząca mapowania obiektowo-relacyjnego

Literatura dotycząca aplikacji szkieletowej Qt

1.6 Układ pracy

Rozdział 2

Zagadnienia teoretyczne dotyczące mapowania obiektowo-relacyjnego oraz aplikacji szkieletowych

2.1 Podstawowe definicje

Rozdział 3

Technologie i metody użyte w części badawczej

Rozdział 4

Aplikacja szkieletowa Qubic

4.1 Dokumentacja techniczna

4.2 Dokumentacja użytkownika

4.3 Przykładowa aplikacja wykorzystująca Qubica

4.4 Testy oraz ich wyniki

4.5 Perspektywy rozwoju Qubica

W celu dalszego rozwoju stworzonej aplikacji szkieletowej warto rozważyć wprowadzenie następujących usprawnień:

- System adnotacji – obecnie na opis mapowania składają się odpowienie nazwy funkcji oraz makra Qt. Istnieje jednak możliwość wprowadzenia własnych makr, które miałyby opisywać mapowanie pomiędzy nazwami tabel z baz danych a odpowiednimi polami klas napisanych z języku C++. Dzięki temu zaistniałaby możliwość uniezależnienia nazw pól klas od nazw tabel w bazie danych.
- Interfejs zapytań – choć jest już zaimplementowany, nadal nie udostępnia on wszystkich możliwych funkcjonalności języka SQL. Implementacja obsługi takich poleceń jak JOIN czy UNION z pewnością byłaby dodatkowym atutem.

- Identyfikacja tabel także za pomocą kluczy złożonych – w tej chwili tabele identyfikowane są za pomocą kluczów głównych, co z kolei wymusza ich nadawanie w każdej z tabel.
- Pamięć podręczna – wprowadzenie pamięci podręcznej może znacznie polepszyć wydajność w przypadku ciągłych operacji na tych samych danych.
- Konfiguracja z poziomu kodu – obecnie większość konfiguracji jest zapisana w plikach konfiguracyjnych i tylko tam może być zmieniana, w celu rozwoju wprowadzenie dodatkowej możliwości jego konfiguracji wydaje się być dobrym pomysłem.
- Wsparcie dla różnych rodzajów baz danych – wprowadzenie tego usprawnienia ogranicza się do implementacji kilku interfejsów dla innych niż MySQL rodzajów baz danych. Biorąc pod uwagę możliwość wzorowania się na zaimplementowanej już logice nie powinno to stworzyć problemu gdy zaistnieje taka konieczność.
- Serializacja danych – dodanie możliwości serializacji może okazać się użyteczne w przypadku pracy z dużymi ilościami danych, w tym celu można skorzystać z wielu istniejących już bibliotek udostępniających tę możliwość.
- Internacjonalizacja – w tej chwili wszystkie logi zlokalizowane są w języku angielskim, istnieje jednak możliwość zmiany obecnego stanu poprzez wykorzystanie modułu translacji udostępnianego przez Qt.
- Wielowątkowość – wykorzystanie wielowątkowości w przypadku mapowania-obiektowo relacyjnego z pewnością nie należy do najłatwiejszych zadań, jednak znacznie może to usprawnić wykonywanie bardziej wymagających operacji.

Rozdział 5

Podsumowanie

5.1 Dyskusja wyników

5.2 Perspektywy rozwoju pracy

Tematem pracy jest: ....., zaś za główny cel przyjęto ..... .
Rozdział zawiera wstęp i cele pracy. W rozdziale drugim opisano/..... w Rozdziale 3. zawarto..... Rozdział 4. przedstawia.....
W podsumowaniu pracy przedstawiono....., z czego wynika, że .....
Najważniejszym wnioskiem/wynikiem/rezultatem pracy jest.....
wyraźnie określić CO TO JEST.

Mapowanie obiektowo-relacyjne (ang. Object-Relational Mapping ORM) to sposób odwzorowania obiektowej architektury systemu informatycznego na bazę danych (lub inny element systemu) o relacyjnym charakterze. Implementacja takiego odwzorowania stosowana jest m.in. w przypadku, gdy tworzony system oparty jest na podejściu obiektowym, a system bazodanowy (System Zarządzania Bazą Danych) operuje na relacjach. Z ORM związany jest szereg problemów wydajnościowych. ORM czyli Object-Relational Mapping jest to rozwiązanie w którym dane są mapowane i zwracane w postaci obiektów.

Odwzorowania obiektowo relacyjne A) ORM pozwala na reprezentację danych pobieranych z bazy w postaci obiektów (kolekcji obiektów, obiektów powiązanych przez referencje). Wiązanie to jest zdefiniowane w sposób deklaratywny (adnotacje lub XML). Izoluje to kod aplikacji (opierający się na paradygmacie obiektowym) od bazy danych (opierającej się na paradygmacie relacyjnym). B) ORM zapewnia obiektom trwałość (ang. persistence). Modyfikacje struktury obiektowej są zapisywane w bazie. A) I B) pozwala na wysokopoziomowy dostęp do bazy. PROBLEM: Niezgodność modeli (impedance mismatch) relacje w bazie są reprezentowane inaczej niż relacje między obiektami. Ponadto baza nie pozwala na dziedziczenie a obiekty tak. ORM wprowadzają sposoby radzenia sobie z tą niezgodnością.

ORM - projektowanie aplikacji Tworzenie warstwy trwałości korzystającej z ORM może być realizowane na 3 sposoby: A) Mapowanie do przodu (forward mapping) - mamy klasy i na ich podstawie będziemy tworzyć bazę danych. To jest najprostsze i nie wymaga zbyt dużej ilości metadanych w kodzie (domyślnie nazwa właściwości jest mapowana do nazwy pola) B) Mapowanie do tyłu (reverse mapping) - mamy bazę danych i na jej podstawie będziemy tworzyć klasy. To jest trudniejsze - wybaga definiowania większej ilości metadanych. C) Mapowanie Meet in The Middle - mamy i schemat bazy i klasy i chcemy je do siebie wzajemnie dopasować.

http://stackoverflow.com/questions/1279613/what-is-an-orm-and-where-can-i-learn-more-about-it

5.3 Istniejące rozwiązania w dziedzinie

W tym podrozdziale zostaną opisane.....

5.3.1 Sprzęt

.....

5.3.2 Oprogramowanie i wdrożone systemy

.....

5.3.3

.....

5.4 Wady i słabe punkty istniejących rozwiązań

5.4.1 Efektywność

.....

5.4.2 Utrudniony dostęp

.....

5.4.3 Wysokie koszty

.....

Rozdział 6

Dalsze uwagi o edycji i formatowaniu pracy

Pracę w LaTeX'u najlepiej składać w szablonie `report`, ze względu na jendostronny wydruk (jak w `article`) i możliwość dzielenia pracy na rozdziały, a co za tym idzie, tworzenia spisu treści, spisu tabel, rysunków.

Przykład 6.1 *Przyklad*

Wniosek 6.1 *Wniosek*

6.1 Bibliografia i przypisy

Spis litertury dołącza się w LaTeX'u automatycznie na końcu pracy (zob. komenda `beginthebibliography`). Informacje o sposobie cytowania zawarte są na stronie Bibilioteki Głównej PŁ
także udostępnione na `http://ics.p.lodz.pl/~aniewiadomski`.

Przykład cytowania – jak podaje praca [1], ..., jednakże autorzy [2] twierdzą, iż.....

Za każdym razem, kiedy w pracy pojawia się treść na podstawie jakiegoś tekstu źródłowego czyjegoś autorstwa, oznaczamy takie miejsce przypisem[1]. Przypis zawierać musi numer jakim w spisie literatury, czyli bibliografii, oznaczono tę pracę, np. tak[2]. **Wszystkie źródła tekstów, rysunków, danych, wykresów, schematów,**

[1] Treść przypisu pierwszego

[2] [3], ss. 3–6 (czyli praca trzecia w spisie literatury, wykorzystany fragment znajduje się na stronach od 3. do 6.)

kodów i informacji wykorzystanych w pracy muszą być zamieszczone w bibliografii. Wszystkie pozycje literatury zamieszczone w bibliografii muszą być cytowane w treści pracy, na dowód, iż zostały rzeczywiście użyte przy pisaniu pracy.

Źródła elektroniczne

Źródła elektroniczne, zwłaszcza internetowe należy cytować z należytą uwagą na ich jakość. Nie cytujemy źródeł wątpliwej jakości lub wtórnie przekazujących czy też powielających wiedzę zawartą w innych źródłach, np. fora internetowe lub wikipedia.

Wszystkie wykorzystane źródła elektroniczne powinny być przez Autora pracy skopiowane w dniu ich wykorzystania i dołączone np. na CD/DVD do wersji drukowanej pracy.

Odnośniki do źródeł elektronicznych muszą zawierać pełną ścieżkę, np. do pliku lub rysunku, a nie jedynie domenowy adres portalu, np.
`http://serwer.com/temat/podtemat/katalog/plik_strony.html` **(stan na dzień: 2009-12-05)**
ale nie
`www.portal.pl`**.** (!!!!!)

Niedochowanie tego wymogu może stać się powodem odrzucenia pracy ze względów formalnych („brak możliwości weryfikacji źródeł wykorzystanych w pracy").

6.2 Polskie akapity, cudzysłowy, itp.

Akapity stosujemy zawsze z wcięciem, ale bez wiersza odstępu pomiędzy akapitami. Ta forma jest przyjęta dla publikacji polskojęzycznych. **W szczególnych przypadkach (także w tym szablonie)** akapit występujący bezpośrednio po tytule rozdziału, sekcji, podsekcji itp. NIE JEST WCIĘTY.

Ten akapit JEST WCIĘTY. NIE MA także PUSTEGO WIERSZA pomiędzy tym akapitem a poprzednim.

Podobne uwagi dotyczą wszystkich innych elementów formatowania pracy – muszą być zgodne ze zwyczajami przyjętymi W JĘZYKU POLSKIM. Np. cudzysłowy wyglądają tak: „cudzysłów", ale nie "cudzysłów", albo też ‘cudzysłów’ czy “cudzysłów”.

Rys. 6.1: Funkcja przynależności zbioru rozmytego – Podpis ZAWSZE POD rysunkiem, numeracja w postaci #.##.

(wypada podać źródło, czyli literaturę, z której rysunek pochodzi, ewentualnie *opracowanie własne*.)

6.3 Definicje i wyrażenia matematyczne

Definicja 1 *Niech $\mathcal{X}$ będzie przestrzenią.....*

Do definicji odnieść sie można poprzez jej etykietę: jak podano w Def. 1

Przykładowe podkreślenie... tekst podkreślony, pogrubienie: **tekst pogrubiony** oraz wyrożnienie *tekst wyróżniony, czyli kursywa*. Dalszy tekst rozdziału Dalszy tekst rozdziału Dalszy tekst rozdziału Dalszy tekst rozdziału Dalszy tekst rozdziału Dalszy tekst rozdziału Dalszy tekst rozdziału Dalszy tekst rozdziału Dalszy tekst rozdziału Dalszy tekst rozdziału a teraz koniec linii...

... i nowy akapit. Akapity muszą być standardowo wcięte.

Przykład wzoru matematycznego numerowanego

$$E = m \cdot c^2 \tag{6.1}$$

Wszystkie symbole matematyczne występujące w tekście „na bieżąco", czyli nieoznaczone numerem równania TAKŻE PISZEMY W TRYBIE MATEMATYCZNYM, CZYLI K U R S Y W Ą : $a = b \cdot c$, ale nie: a= b*c (!!)

Numeracja wzoru – ZAWSZE w POSTACI (#.##) Jak podaje wzor (6.1).... (koniec linii).

Wyrażenia matematyczne można też wpisywać w wierszu – używamy wówczas znaku '$', który rozpoczyna i kończy wyrażenie, np. wg Einsteina $E = m \cdot c^2$...

6.4 Jak wstawiać rysunki? tabele?

A teraz pora na rysunek:

Rysunki i tabele nie powinny przekraczać 0.9 szerokości tekstu i zasadniczo powinny występować na górze strony.

Odnosić się do rysunku można poprzez jego etykietę "label", np. jak widać na rys. 6.1.....

Tab. 6.1: Tytuł tabeli ZAWSZE NAD TABELĄ, numeracja w formie #.##. (wypada podać źródło, czyli literaturę, z której tabela pochodzi, ewentualnie *opracowanie własne*.)

Alg.	**tytuł kolumny 1**	**tytuł kolumny 1**	**Tytuł kolumny 3**	**....**
a	b	c	d	e
a	b	c	d	e

Jak widać, rysunek nie wypada w dokumencie w tym samym miejscu co w kodzie, choć czasem się tak zdarza. Jeśli potrzebujesz przenieść rysunek, zajrzyj do rozdzialu 2.11. manuala pt. *Wstawki*.

6.5 Listy wypunktowana i numerowana

- pierwszy element listy wypunktowanej
- drugi...
- trzeci...

Nowy akapit z lista numerowaną.

1. pierwszy element listy NUMEROWANEJ
2. drugi...
3. trzeci...
4. trzeci...
5. trzeci...

6.6 Przenoszenie wyrazów

Skorzystaj z polecenia `hyphenation`
w preambule dokumentu, lub dziel wyrazy „ręcznie” czyli właśnie tak jak tu: podzielone wyrazy.

Tytuł tego rozdziału ma dwie wersje: zwykłą, (w kodzie: w nawiasach klamrowych), która pokazuje sie na stronie rozpoczynającej rozdział, oraz krótką (w kodzie: w nawiasach kwadratowych), która pokazuje sie w spisie treści i w nagłówku

W rozdziale **??** podano podstawy teoretyczne i ogólny zakres pracy. W niniejszym rozdziale opisana zostanie technologia XYZ oraz metoda ABC użyta w części praktycznej, patrz rozdział 7.

6.7 Sprzęt

.....

6.7.1 Element 1

.....

6.7.2 Element 2

.....

6.8 Oprogramowanie

.....

6.8.1 Serwer baz danych

.....

6.8.2 Środowisko zintegrowane

.....

6.8.3 Oprogramowanie klienckie

6.9 Technologie i metodologie programistyczne

.....

6.9.1 Język programowania

.....

6.9.2 Biblioteki

.....

6.9.3 Wzorce projektowe

.....

6.10 Inne, np. narzędzia i metody symulacji,

Rozdział 7

Biblioteka programistyczna Qubic

Ta część pracy może być podzielona na więcej rozdziałów, np kiedy autor chce w szczególności podkreślić któryś z etapów projektu. W zależności od tematu i celów pracy, pewne sekcje można dodać (np. przy projektowaniu sieci, instalacji i konfiguracji serwerów usług sieciowych), inne zaś pominąć.

7.1 Analiza wymagań

7.1.1 Studium możliwości

7.1.2 Wymagania funkcjonalne

.....

7.1.3 Ograniczenia projektu

7.2 Projekt

7.2.1 Projekt warstwy danych

1. normalizacje baz danych
2. projekt bazy/baz
3. grupy użytkowników i ich prawa dostępu do danych (zależne od implementacji bazy)
4. ew. diagramy klas warstwy danych

7.2.2 Projekt warstwy logiki

1. Diagramy i scenariusze przypadków użycia
2. Diagramy przepływu danych (lub ich odpowiedniki)
3. ew. diagramy klas, wzorce projektowe itp.

7.2.3 Projekt warstwy interfejsu użytkownika

Wybór środowiska i platformy działania

Rodzaj aplikacji (klient-serwer, thick/thin client, aplikacja „biurkowa”, usługa, klient hybrydowy, itp.

Technologie projektowania i realizacji interfejsu użytkownika, np. biblioteki

7.3 Implementacja: punkty kluczowe

7.4 Testy i wdrożenie

7.4.1 Testy wydajności

7.4.2 Testy regresyjne

7.4.3 Testy bezpieczeństwa

7.4.4 Dalsze testy

7.4.5 Testy...

7.5 Konserwacja i inżynieria wtórna

Jak przebiega eksploatacja systemu/projektu? Jakie wady i zalety ujawniły się po np. 2-miesięcznym okresie testowania i użytkowania?

Jak można skorzystać z tej wiedzy praktycznej pod kątem rozbudowy pracy? Jakie elementy systemu powinny zostać w pierwszej kolejności zmodyfikowane?

Rozdział 8

Podsumowanie

8.1 Dyskusja wyników

Dzięki zrealizowaniu pracy poprawie uległa wydajność ..... Ponadto, o ?? % skrócony został czas ....., a koszty osiągnięcia zamierzonego efektu zostały zmniejszone z ???pln do ???pln za godzinę/ dzień/ jednostkę sprzętu.....

Które cele pracy udało sie zrealizować? co z tego wynika? Które cele pracy pozostały niezrealizowane i dlaczego?

8.2 Ocena możliwości wdrożenia proponowanych rozwiązań...

... ich wartość praktyczna, lokalne i globalne możliwości zastosowania, kwestia praw autorskich do powstałych produktów, itp.

Bibliografia

[1] Kacprzyk J. (1986) Fuzzy sets in system analysis. PWN, Warsaw (in Polish).

[2] Kacprzyk J., Strykowski P. (1999) Linguistic Data Summaries for Intelligent Decision Support, Proceedings of EFDAN'99. 4-th European Workshop on Fuzzy Decision Analysis and Recognition Technology for Management, Planning and Optimization, Dortmund, 1999, 3–12.

[3] Kacprzyk J., Yager R. R. (2001) Linguistic summaries of data using fuzzy logic. International Journal of General Systems 30:133–154

Spis rysunków

Spis tabel

Załączniki

1. Załącznik nr 1
2. Załącznik nr 2
3. Załącznik nr 3