Politechnika Warszawska

Podstawy implementacji oprogramowania

dr hab. inż. Michał Śmiałek, prof. uczelni
dr inż. Kamil Rybiński

Wydział Elektryczny
POLITECHNIKA WARSZAWSKA

Inżynieria oprogramowania

1

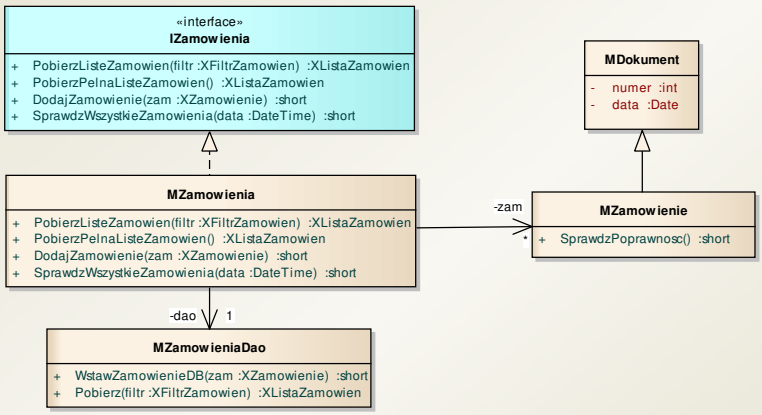
Inżynieria okrężna

- Inżynieria w przód
  - Generowanie kodu z modeli
- Inżynieria odwrotna
  - Odtwarzanie zmian kodu w modelu


```
package pl.edu.pw.dziekanat.dane;
public class DaneUzytkownika {
  private String imie;
  private String nazwisko;
  private String stanowisko;
  private int staz;    float staz;

  public DaneUzytkownika(){
  }

  public int obliczWynagrodzenie(){
    if(stanowisko == KIEROWNIK))
      {return 5000 + staz*100;}
      else
      {return 3000+ staz*50;}
  }
}
```


2

Generowanie kodu z modelu klas (1)



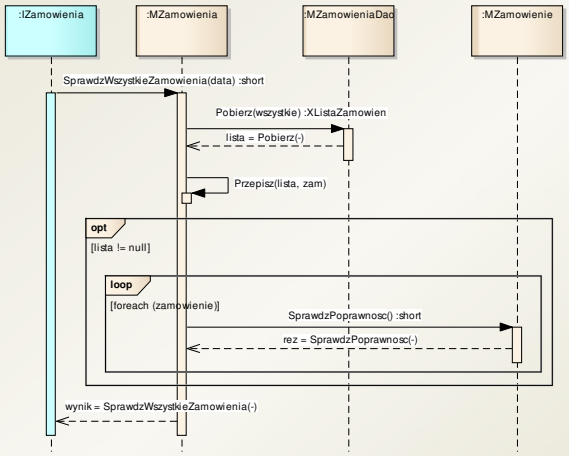
- Przykładowy model, przygotowany do generacji kodu

Generowanie kodu z modelu klas (2)

- Najważniejsze fragmenty kodu
  - Wygenerowane automatycznie z modelu
  - Zawierają całą strukturę zawartą w modelu
  - Odpowiednio realizują nawigowalność asocjacji
  - Nie zawierają treści metod

```
public interface IZamowienia  {

   XListaZamowien PobierzListeZamowien(XFiltrZamowien filtr);
   XListaZamowien PobierzPelnaListeZamowien();
   short DodajZamowienie(XZamowienie zam);

}//end IZamowienia

public class MZamowienia : IZamowienia {

   private List<MZamowienie> zam;
   private MZamowieniaDao dao;

   public MZamowienia(){
   }

   public XListaZamowien PobierzListeZamowien(XFiltrZamowien filtr){
      return null;
   }

   public XListaZamowien PobierzPelnaListeZamowien(){
      return null;
   }   // (...) pozostałe metody pominięte

}//end MZamowienia

public class MZamowienie : MDokument {

   public MZamowienie(){

   }

   public short SprawdzPoprawnosc(){
      return 0;
   }

}//end MZamowienie
```

Pisanie kodu na podstawie modelu sekwencji (1)



- Diagram sekwencji modeluje dynamikę komunikacji między obiektami

5

Pisanie kodu na podstawie modelu sekwencji (2)

```
public short SprawdzWszystkieZamowienia(DateTime data)
{
   short wynik = 0;
   lista = dao.Pobierz(wszystkie);
   Przepisz(lista, zam);
   if (null != lista)
   {
      foreach (MZamowienie zamowienie in zam)
      {
         wynik = zamowienie.SprawdzPoprawnosc();
      }
   }
   return wynik;
}
```

- Realizacja modelu sekwencji w kodzie
  - Ramki opt/alt jako instrukcje „if”
  - Ramki loop jako instrukcje „foreach”
  - Komunikaty synchroniczne jako wywołania procedur

6

Dobre praktyki redaktorskie w zakresie kodowania

- Formatowanie tekstu
  - Wcięcia, odstępy, załamania wierszy, …
- Komentowanie kodu
  - Bardzo wskazane, jeśli kod nie jest oczywisty
  - Uzupełniają modele projektowe
  - Bez przesady – nie należy komentować wszystkiego
  - Formalne – umożliwiają generowanie dokumentacji kodu
- Konwencje nazewnicze
  - Zgodne z konwencjami przyjętymi dla danego języka programowania oraz organizacji (projektu)
  - Zachowują czytelność nazw

Przykład formatowania kodu

```
/// <summary>
/// [GetSupportedResourceRange] Get ResourceRange for the given ComputationRelease.
/// </summary>
/// <param name="releaseUid">The Uid of the ComputationRelease</param>
/// <returns>A single ResourceRange record.</returns>
[HttpGet("range")]
public IActionResult GetSupportedResourceRange([FromQuery] string releaseUid)
{
    try
    {
        var reservationRange = _taskManager.GetSupportedResourceRange(releaseUid);
        if (null == reservationRange)
           return HandleError("Release not found", HttpStatusCode.BadRequest);

        return Ok(new XReservationRange(reservationRange));
    }
    catch (Exception e)
    {
        return HandleError(e);
    }
}
```

- Wcięcia, puste wiersze, formalne komentarze
- Zachowany standard nazewniczy dla języka C#

Praktyki merytoryczne dla pisania kodu (1)

- Projektowanie podczas kodowania
  - Zmiana struktury kodu to projektowanie
  - Stosowanie zasad inżynierii odwrotnej
- Parametryzowalność kodu
  - Stałe zapisywane symbolicznie
  - Parametry wpływające na kompilację w plikach zewnętrznych
- Unikanie „magicznych wartości”
  - Stosowanie nazw symbolicznych, a nie konkretnych liczb w formułach
- Posługiwanie się wzorcami projektowymi
  - Fabryka abstrakcyjna, singleton, komenda, leniwa inicjalizacja. Konstruktor prywatny, …

9

Praktyki merytoryczne dla pisania kodu (2)

- Uwzględnienie kodu dla testów
  - Pisany równolegle z kodem właściwym
- Walidacja parametrów wejściowych
  - Kontrola zachowania „kontraktu”
  - Zawsze zakłada, że kod wywołujący może podać błędne dane
- Obsługa raportowania wykonania kodu
  - Wysyłanie informacji do dzienników zdarzeń
  - Odpowiednia polityka reakcji na błędy (wyjątki)
- Dbałość o dane
  - Uniezależnienie danych od konkretnego systemu
  - Zabezpieczenie danych przed jednoczesnym zapisem

10

Praktyki merytoryczne dla pisania kodu (3)

- Dbałość o zasoby
  - Oszczędne wykorzystanie zasobów maszyn wykonawczych
- Przenośność
  - Przestrzeganie reguł zalecanych przez systemy operacyjne
  - Stosowanie tylko stabilnych wersji procedur dostarczanych przez środowisko
- Skalowalność
  - Reakcja na różne poziomy obciążenia obliczeniami
  - Dbanie o fragmenty kodu szczególnie wrażliwe
- Przestrzeganie zasad paradygmatu obiektowego
  - Rozkładanie odpowiedzialności między klasy

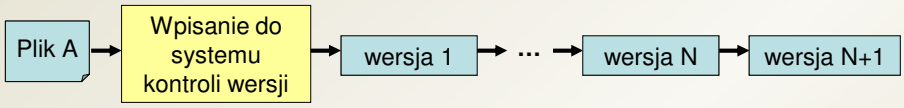
11

Praktyki pracy zespołowej i wersjonowania kodu

- „Czystość” własnego kodu
  - Kod wstawiany do repozytorium jest wolny od błędów kompilacji, ostrzeżeń i innych wad
- Praca na aktualnym kodzie
  - Synchronizowanie z repozytorium przez dokonaniem zmian
- Umiejętne korzystanie ze środowiska
  - Dobre poznanie możliwości środowiska
  - Stosowanie funkcji IDE ułatwiających programowanie (np. kontrola składni, autouzupełnianie)

12

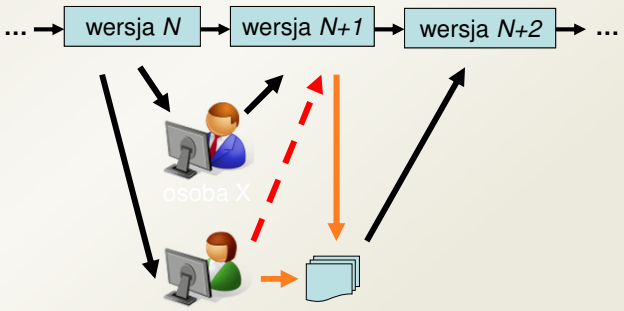
Zarządzanie wersjami plików



- Zapisywanie kolejnych zmian w pliku
  - Każda zmiana to nowa wersja
  - Możliwość przywrócenia poprzednich wersji
- Systemy zarządzania wersjami
  - Scentralizowane: wszystkie wersje przechowywane na centralnym serwerze (CVS, SVN)
  - Zdecentralizowane: centralny serwer j.w., ale lokalne maszyny mają możliwość lokalnego wersjonowania (Git)

Strategie wersjonowania

- Stosowanie blokad
  - Użytkownik blokuje możliwość dokonywania zmian przez innych
- Stosowanie scaleń
  - Równoległa praca nad kodem, który może być scalony z kliku wersji



Elementy systemów kontroli wersji

- Dodatkowy opis pliku
  - Numer wersji, identyfikator osoby, data zmiany, krótki opis zmian
- Delta
  - Różnica między poprzednią a obecną wersją pliku/zasobu
- Gałąź
  - Ciąg wersji pliku tworzonych niezależnie od innych wersji
  - Główna gałąź tworzy pień
  - Stabilne gałęzie są scalane z pniem
- Znacznik
  - Nazwa symboliczna ułatwiająca szybki powrót do wybranej wersji pliku/zasobu

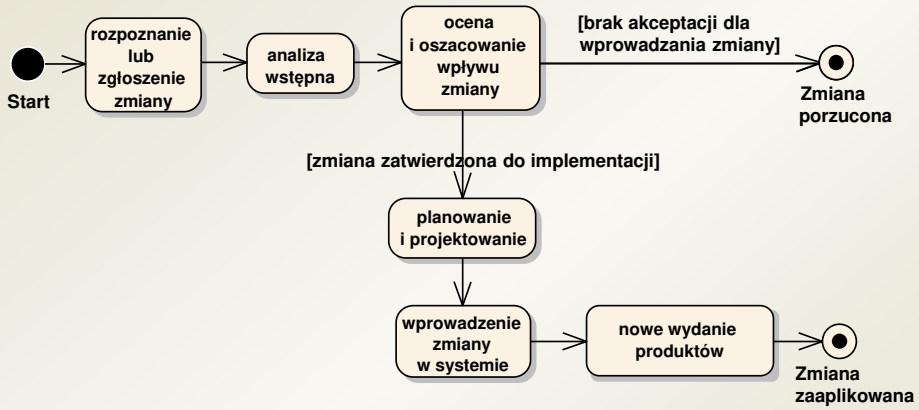
15

Zarządzanie zmianami

- Każdy produkt pracy w projekcie może podlegać zmianom
  - Zmiany wynikają z różnych czynników: zmiany otoczenia, potrzeb, wyników analiz, błędów, …
- Zmiany muszą być dokumentowane (atrybuty)
  - Źródło zmiany
  - Opis zmiany
  - Opis efektów zmiany (wpływu na inne elementy)
  - Osoby zaangażowane w zmianę
  - Identyfikator, data, …

16

Proces aplikacji zmiany



- Każda zmiana powinna podlegać analizie i ocenie wpływu na projekt