Modelowanie i Analiza Systemów informacyjnych (MAS)

Studia zaoczne. Wydział informatyki, Wydział zarządzania

wersja z 2020-10-01

Wykładowca: **dr inż. Mariusz Trzaska**
(*mtrzaska@pjwstk.edu.pl*, *http://www.mtrzaska.com*)

1. Wprowadzenie

   Przedmiot poświęcony jest osadzeniu modelu pojęciowego dziedziny problemowej, efektu fazy analizy i specyfikacji wymagań, w konkretnym środowisku implementacyjnym (zarówno obiektowym jak i relacyjnym). Studenci poznają sposoby realizacji konstrukcji, niezbędnych do osadzenia modelu, a nie istniejących w wybranym języku programowania. Dyskutowane są również elementy związane z użytecznością (w tym jej testowaniem) graficznych interfejsów użytkownika. Wiedza teoretyczna poparta jest praktyczną implementacją struktury danych, logiki biznesowej oraz prostych i zaawansowanych graficznych interfejsów użytkownika (m. in. przy wykorzystaniu dedykowanych edytorów). Każdy ze studentów jest zobowiązany do przeprowadzenia analizy dynamicznej oraz wykonania prac projektowych i implementacyjnych, w oparciu o indywidualne wymagania użytkownika (projekt wielkości 12-15 klas). Specyfikacja wymagań i analiza statyczna powinny być przeprowadzone w trakcie nauczania przedmiotu Projektowanie Systemów Informacyjnych (PRI).

2. Plan zajęć

Nr	Wykład	Ćwiczenia
1	Wybrane konstrukcje obiektowych języków programowania	• Krótkie omówienie założeń przedmiotu oraz projektu • Ćwiczenia z analizy dynamicznej.
2	Wykorzystanie klas w obiektowych językach programowania	Ćwiczenia z wybranych konstrukcji obiektowych języków programowania.
3	Realizacja asocjacji w obiektowych językach programowania	Ćwiczenia z wykorzystania klas w obiektowych językach programowania.
4	Realizacja asocjacji w obiektowych językach programowania (2)	• Ćwiczenia z realizacji asocjacji w obiektowych językach programowania (1). • **Odbiór mini-projektu MP1.**

5	Realizacja różnych modeli dziedziczenia w obiektowych językach programowania	Ćwiczenia z realizacji asocjacji w obiektowych językach programowania (2).
6	Realizacja pozostałych, wybranych konstrukcji UML w obiektowych językach programowania	• Ćwiczenia z realizacji różnych modeli dziedziczenia w obiektowych językach programowania. • **Odbiór mini-projektu MP2.**
7	Użyteczność (*usability*) oraz implementacja graficznych interfejsów użytkownika	• Ćwiczenia z tworzenia GUI (m.in. korzystając z edytora). • **Odbiór mini-projektu MP3.** • **Ostateczny termin wysłania dokumentacji projektowej**. Nie ma możliwości późniejszego oddawania dokumentacji.
8	Wykorzystanie modelu relacyjnego w obiektowych językach programowania	**Odbiór projektów końcowych.**

3. Mini-projekty

Ich celem jest praktyczne sprawdzenie zrozumienia sposobów implementacji poszczególnych konstrukcji z modelu pojęciowego (klasy, asocjacje, ekstensja, itp.) oraz współpracy z modelem relacyjnym. Dodatkowo można go potraktować jako „fundament" projektu końcowego (który później zostanie uzupełniony o odpowiednie elementy, m. in. GUI). Należy zaimplementować różne **biznesowe konstrukcje** (odpadają wszelkie techniczne informacje jak np. liczba obiektów w ekstensji, settery/gettery, wyświetlanie obiektów) występujące w modelu. Innymi słowy należy przejrzeć odpowiedni wykład (np. klasy dla MP1, asocjacje dla MP2, itd.) i zaimplementować omawiane tam zagadnienia. Każdy MP musi zawierać przykładowe dane ilustrujące jego poprawną pracę (metoda `main`). Terminy oddawania: patrz załączony plan zajęć. Istnieje możliwość oddawania mini projektów na późniejszych zajęciach, ale wiąże się to z 50% redukcją punktacji. Elementy MP, które podlegają ocenie[1]:

MP 01 Klasy, atrybuty	**MP 02** Asocjacje	**MP 03** Dziedziczenie
• Ekstensja • Ekst. - trwałość • Atr. złożony • Atr. opcjonalny • Atr. powt.	• „Zwykła" • Z atrybutem • Kwalifikowana • Kompozycja *(w każdym przypadku: liczności 1-* lub *-* oraz automatyczne*	• Klasa abstr/ polimorfizm • Overlapping • Wielodziedziczenie • Wieloaspektowe • Dynamiczne

[1] Należy zaproponować własne przypadki biznesowe, tzn. nie wolno wykorzystywać przykładów z wykładów, książki, itp.

• Atr. klasowy • Atr. pochodny • Met. klasowa • Przesłonięcie, przeciążenie	*tworzenie poł. zwrotnego)*	

Tematyka MPx może, ale nie musi, być związana z tematyką projektu końcowego.

Nie wolno łączyć ze sobą przykładów dla poszczególnych konstrukcji, np. każdy rodzaj atrybutu musi mieć swój własny biznesowy przykład.

Mini-projekty są integralną częścią składową oceny końcowej (patrz punkt 5.1.1). W związku z tym warto je dobrze wykonać.

4. Projekt

Projekt składa się z dwóch części: dokumentacyjnej oraz implementacyjnej. Jego tematyka powinna być tak dobrana aby dało się stworzyć odpowiednią liczbę (12 - 15)[2] sensownych klas biznesowych. W związku z tym raczej odpadają wszelkie aplikacje narzędziowe, systemowe, odtwarzacze multimediów, itp. W razie wątpliwości należy skonsultować się z prowadzącym ćwiczenia.

4.1. Dokumentacja

4.1.1. Dokumentacja zawiera część „starą”, czyli tą, która została wyprodukowana na przedmiocie PRI (w postaci załącznika) oraz dokumentację „nową”. Dokumentacja „nowa” powiela schemat dokumentacji „starej”, ale dochodzi tu też trochę nowych elementów, o których poniżej. Osoby, które nie posiadają „starego” projektu z PRI muszą opracować co najmniej: wymagania (jako „historyjkę”), diagram przypadków użycia oraz analityczny diagram klas.

4.1.2. Przypadki użycia: „Nowa” dokumentacja przypadków użycia powinna zawierać oprócz diagramu powielonego ze „starej” dokumentacji szczegółowy opis jednego nietrywialnego (odwołującego się do innego przypadku użycia) przypadku użycia (zgodnie z materiałem podawanym na wykładzie z PRI). Scenariusz, dla tego przypadku, powinien być sporządzony zarówno z wykorzystaniem języka naturalnego, jak i diagramów aktywności.

4.1.3. Projekt interfejsu użytkownika: W oparciu o wybrany, nietrywialny przypadek użycia powinien być sporządzony projekt interfejsu użytkownika (zgodnie z wytycznymi podanymi na wykładzie).

4.1.4. Dla wybranego przypadku użycia należy przeprowadzić analizę dynamiczną (czyli wykorzystać diagramy: interakcji[3], aktywności i stanu[4]). Analiza dynamiczna powinna zakończyć się nie tylko pojawieniem się metod na diagramie klas, ale też jawnym omówieniem jej skutków. „Skutki” analizy dynamicznej (być może nie tylko metody,

[2] Dla Wydziału Zarządzania Informacją (WZI) obowiązuje limit 8 – 15 klas.
[3] Nie dotyczy Wydziału Zarządzania
[4] Diagram stanu robimy dla konkretnej klasy, w miarę możliwości z analizowanego przypadku użycia.

ale też nowe atrybuty, nowe asocjacje, itp.) powinny być umieszczone na diagramie klas przerysowanym ze „starego” projektu i w miarę możliwości wyróżnione innym kolorem. Projekt musi wykorzystywać wszystkie wymienione rodzaje diagramów.

4.1.5. Przed „ostatecznym” diagramem klas, stanowiącym podstawę do implementacji, należy koniecznie dołączyć elementy specyfikujące podjęte decyzje projektowe (na przykładach, w oparciu o odpowiednie fragmenty diagramu), np. sposób realizacji ekstensji klasy.

4.1.6. Podsumowując, „ostateczny” (projektowy) diagram klas różni się od diagramu dostarczonego na PRI w następujących punktach:

4.1.6.1. jest uszczegółowiony,

4.1.6.2. wszystkie konstrukcje nie istniejące w danym języku programowania są zamienione (zgodnie z podjętymi decyzjami projektowymi),

4.1.6.3. jest uzupełniony o metody wynikłe z analizy dynamicznej.

4.1.7. Należy zadbać o **czytelność całej dokumentacji**, a szczególnie diagramów (zalecany jest jakiś format wektorowy (np. *Enhanced Metafile*). W tym celu warto upewnić się, że:

- prawidłowo stosujemy notację UML (a nie „podobne” elementy graficzne),
- każdy z nich jest prawidłowo podpisany,
- został przygotowany w odpowiednim narzędziu (np. UMLet),
- stosujemy rozmiar czcionek niewymagający nadmiernego powiększania (czytelny diagram A4 w widoku 100%).
- elementy (np. klasy) są właściwe rozplanowane, m. in. dziedziczenie w pionie, asocjacje w poziomie, unikamy przecinania linii.

4.1.8. Dokumentację projektową oddajemy w postaci pojedynczego pliku PDF (*MAS_Grupa_Nazwisko_Imię_NrIndeksu.PDF*) wysłanego na adres mailowy prowadzącego ćwiczenia. Ostateczny termin – patrz załączony plan zajęć.

4.1.9. Podsumowanie zawartości dokumentacji projektowej (PRI + MAS):

4.1.9.1. Wymagania użytkownika

4.1.9.2. Diagram przypadków użycia

4.1.9.3. Diagram klas – analityczny

4.1.9.4. Diagram klas – projektowy

4.1.9.5. Scenariusz przypadku użycia (jako tekst)

4.1.9.6. Diagram aktywności dla przypadku użycia

4.1.9.7. Diagram stanu dla klasy

4.1.9.8. Diagram interakcji (sekwencji) dla przypadku użycia

4.1.9.9. Projekt GUI

4.1.9.10. Omówienie decyzji projektowych i skutków analizy dynamicznej

4.1.10. Ocena za dokumentację będzie wystawiana zgodnie z poniższą tabelą (warunek wstępny: spełnienie wymagań z pkt. 4.1):

Kryterium	**Maks. pkt.**
Skomplikowanie dziedziny biznesowej	10
Udokumentowanie przypadku (ów) użycia (scenariusz i diagram)	10
Poprawność i złożoność projektowego diagramu klas	25
Poprawność i złożoność diagramu interakcji	10
Poprawność i złożoność diagramu aktywności	10
Poprawność i złożoność diagramu stanu	10
Projekt GUI	10
Omówienie decyzji projektowych	10
Czytelność i organizacja dokumentu	5
Razem	*100*

4.2. Implementacja[5]

4.2.1. Cała struktura (wszystkie klasy z odpowiednimi powiązaniami),

4.2.2. Metody potrzebne do realizacji wybranego przypadku (lub przypadków) użycia,

4.2.3. Elementy graficznego interfejsu użytkownika (GUI)[5], które są niezbędne do zaprezentowania pracującej implementacji z działającym wybranym przypadkiem użycia. Każdy projekt **musi** posiadać GUI[5].

4.2.4. Minimalna implementacja GUI powinna obejmować interakcję co najmniej dwóch klas połączonych asocjacją (wymagana liczność docelowa: „wiele"), np. mamy dwie klasy: Pracownik i Firma; odpowiedni *widget* (wyświetlający wiele elementów, np. *ListBox*) zawiera listę firm, po kliknięciu w dowolną pozycję wyświetla się inny widget (wyświetlający wiele elementów, np. *ListBox*) zawierający listę jej pracowników. Zaimplementowane GUI, które **tylko** tworzy połączenia pomiędzy obiektami i nie pozwala na w/w interakcję, nie wystarczy do zaliczenia. Analogicznie, niewystarczające są np. rozwiązania z jednym *widget*'em, *TextBox*'em, licznością docelową „1" lub filtrujące dane z ekstensji (zamiast korzystania z uprzednio zdefiniowanej asocjacji).

4.2.5. Należy zwrócić uwagę na jakość, ergonomię i użyteczność (*usability*) GUI (np. skalowanie okien, kolorystyka, filozofia działania) – jest to **ważny** składnik oceny końcowej. Projekt i wykonanie GUI (można

[5] Projekt dla WZI na st. zaocznych nie musi posiadać GUI. Zamiast tego można stworzyć metodę main ilustrującą działanie zrealizowanych konstrukcji (przykładowe dane i operacje na nich, np. stworzenie powiązań i wyświetlenie na konsoli powiązanych obiektów).

używać dedykowanych edytorów) powinno być zgodne z zasadami użyteczności (*usability*), podawanymi na wykładzie.

4.2.6. **Wszystkie** dane przechowywane w systemie muszą być trwałe (np. plik, baza danych, dedykowana biblioteka, itp.).

4.2.7. Część implementacyjna projektu będzie indywidualnie odbierana w czasie zajęć (patrz dalej). W związku z tym nie trzeba jej przekazywać w żadnej trwałej formie.

4.2.8. Językiem implementacji może być Java, C# lub C++. Inne języki wymagają zgody prowadzącego ćwiczenia.

4.2.9. Ocena za implementację będzie wystawiana zgodnie z poniższą tabelą (warunek wstępny: spełnienie wymagań z pkt. 4.2):

Kryterium	**Maks. pkt.**
Trudność zadania	10
Zakres i poprawność zrealizowanej funkcjonalności	15
Zakres i poprawność zrealizowanych konstrukcji z obiektowości	25
Jakość kodu (nazwy, komentarze, *JavaDoc*, itp.)	5
Elegancja zaimplementowanych rozwiązań	15
Sposób implementacji trwałości	10
Implementacja GUI (w tym użyteczność/ergonomia)	20
Razem	*100*

Projekt końcowy nie musi zawierać wszystkich konstrukcji z MPx.

4.3. **Każdy** projekt będzie indywidualnie odbierany. W trakcie odbioru można spodziewać się szczegółowych pytań dotyczących sposobu implementacji, np. „co by było gdyby...”, „dlaczego jest to zrobione w ten sposób...”, „proszę dokonać następującej modyfikacji...”. Osoby, które samodzielnie wykonały projekt nie powinny mieć problemów z udzieleniem odpowiedzi na powyższe pytania. Brak umiejętności odpowiedzi na powyższe pytania oznacza **brak zaliczenia ćwiczeń**.

4.4. Ocenie będzie podlegać (patrz też pkt. 4.1.10, 4.2.9):

4.4.1. Trudność zadania,

4.4.2. Zakres zrealizowanej funkcjonalności,

4.4.3. Udokumentowanie kodu w tym komentarze umożliwiające automatyczne wygenerowanie dokumentacji API,

4.4.4. Elegancja zaimplementowanych rozwiązań, w tym ergonomia GUI.

5. Zaliczenie ćwiczeń

Ocena końcowa z ćwiczeń składa się z następujących składowych:

5.1. Punkty (maks 100 pkt.; liczy się suma - nie trzeba indywidualnie zaliczać każdego elementu)

5.1.1. Punkty za mini-projekty (maks. 35 + 35 + 30 = 100 pkt.),

5.2. Ocena z projektu,

5.2.1. Ocena za implementację

5.2.2. Ocena za dokumentację

Z każdej części 5.1 i 5.2 (w tym 5.2.1 oraz 5.2.2) trzeba otrzymać ocenę pozytywną. W związku z tym osoby, które np. zaliczą MP, a nie zaliczą dokumentacji projektu **nie zaliczą ćwiczeń**.

Dodatkowo można zdobyć do 15 pkt. za rozwiązanie zadań podanych w czasie wykładów. Te bonusowe punkty doliczane są do ogólnej liczby punktów (patrz pkt. 5.1) pod warunkiem posiadania co najmniej 50% pkt.

6. Terminy

Terminy realizacji poszczególnych zadań (mini-projekty, dokumentacja projektu końcowego, implementacja projektu końcowego) są podane w tabeli (pkt. 2). Ich niedotrzymanie wiąże się z poważną redukcją oceny, aż do niezaliczenia włącznie.

Nie ma również możliwości zaliczania przedmiotu po zakończeniu semestru, warunkowo na egzaminie, w sesji poprawkowej, itp.

7. Egzamin

Egzamin z MAS składa się z dwóch części (liczy się suma punktów):

7.1. **Testowej**. Należy ocenić każde z pytań (T/N). Odpowiedź prawidłowa oznacza +1 pkt., błędna -1 pkt., brak odpowiedzi 0 pkt.

7.2. **Zadaniowej**. Należy nazwać oraz krótko omówić sposób implementacji zaznaczonych konstrukcji na otrzymanym diagramie klas.

Nie ma zwolnień z egzaminu.

Przykładowy egzamin: http://www.mtrzaska.com/mas-egzamin.

8. Materiały

8.1. Książka: M. Trzaska: „Modelowanie i implementacja systemów informatycznych”. Rok 2010. Wydawnictwo PJWSTK. Stron 299. ISBN 978-83-89244-71-3.

- Wersja elektroniczna (eBook działający na Windows, iOS, Android, urządzeniach mobilnych oraz czytnikach) w księgarni internetowej Ibuk: wersja PDF.
- Wersja drukowana: sklep PJWSTK.

- Fragmenty w PDF: http://www.mtrzaska.com/mas-ksiazka

8.2. Informacje ogólne (mogą pojawiać się nowsze wersje).

http://www.mtrzaska.com/mas-informacje

8.3. Wersja elektroniczna wykładów:

http://www.mtrzaska.com/mas

Ze względu na skomplikowanie omawianych zagadnień, zaleca się uczęszczanie na wykład (niezależnie od tego, że powyższe materiały są dostępne *on-line*).

8.4. Przykładowe zadania programistyczne

Niniejsze zadania programistyczne mają na celu jedynie przypomnienie materiału dotyczącego programowania (przyswojonego w czasie wcześniejszych kursów) i ich rozwiązania nie będą oceniane w ramach przedmiotu MAS (Modelowanie i Analiza Systemów informacyjnych). Mogą być wykorzystane w czasie zajęć „*Wybrane konstrukcje obiektowych języków programowania*”. Studenci przystępujący do kursu MAS, powinni umieć rozwiązać zdecydowaną większość z nich.

http://www.mtrzaska.com/plik/mas/mas-zadania-programistyczne

8.5. Bezpłatne książki on-line:

8.5.1. Bruce Eckel - Thinking in Java: http://www.mindview.net/Books/TIJ/

8.5.2. Allen B. Downey - How to Think Like a Computer Scientist: Java Version: http://www.greenteapress.com/thinkapjava/

8.5.3. Robert Sedgewick and Kevin Wayne - Introduction to Programming in Java: An Interdisciplinary Approach: http://introcs.cs.princeton.edu/home/

9. Narzędzia implementacyjne

Ze względu na fakt iż istnieje dość duża dowolność wyboru technologii realizacji projektu, nie ma listy narzędzi obowiązkowych. Niemniej poniżej umieszczono listę aplikacji, które mogą być przydatne:

- Narzędzia CASE (m.in. edytory diagramów):
  - UMLet (*open source*, wieloplatformowy): https://www.umlet.com/
  - Modelio (*open source*, wieloplatformowy): https://www.modelio.org/,
  - Visual Paradigm Community Edition: http://www.visual-paradigm.com,
  - ArgoUML: http://argouml.tigris.org/,
  - MagicDraw Community Edition: http://www.magicdraw.com,
  - StarUML: http://staruml.sourceforge.net/en/,
  - NetBeans for Java: http://www.netbeans.org/ (umożliwia m.in. tworzenie diagramów UML i generowanie kodu źródłowego)
  - MS Visio (dostępny na licencji ELMS dla studentów PJATK),

  - Obszerna lista narzędzi: http://en.wikipedia.org/wiki/List_of_UML_tools.
- IDE
  - IntelliJ IDEA (darmowy *Community*): https://www.jetbrains.com/idea/
  - Eclipse for Java: http://www.eclipse.org/,
  - NetBeans for Java: http://www.netbeans.org/,
  - MS Visual Studio (dostępny na licencji ELMS dla studentów PJWSTK).
- Edytory GUI
  - wbudowany w NetBeans;
  - dla Eclipse: Jigloo SWT/Swing GUI Builder (http://www.cloudgarden.com/jigloo/);
  - dla Eclipse: WindowBuilder Pro - po przejęciu przez Google darmowy (http://code.google.com/intl/pl/webtoolkit/tools/wbpro);
  - wbudowany w MS Visual Studio.

W razie wątpliwości proszę o kontakt: *mtrzaska@pjwstk.edu.pl*