

Analiza i programowanie obiektowe

2008/2009

Wykład 1: Wprowadzenie oraz cykl życia oprogramowania i faza określenia wymagań

Jacek Marciniak
Wydział Matematyki i Informatyki
Uniwersytet im. Adama Mickiewicza

1

Plan wykładu

1. Wprowadzenie:
 - Dlaczego analiza i programowanie obiektowe,
 - Literatura.
2. Cykl życia oprogramowania.
3. Faza określenia wymagań.

2

Wprowadzenie: Co jest przedmiotem wykładu

3

Wprowadzenie

- **Cel wykładu:**
 - Wprowadzenie do analizy i projektowania obiektowego,
 - Wprowadzenie do korzystania z wzorców projektowych (patterns).
 - Wprowadzenie do UML (Unified Modelling Language),
 - Umiejętności zdobyte w ramach zajęć pozwolą stworzyć oprogramowanie obiektowe, które będzie dobrze zaprojektowane, wydajne i łatwe w utrzymaniu – możliwe to będzie dzięki znajomości zestawu reguł oraz heurystyk.
- **Założenia wstępne odnośnie słuchacza wykładu:**
 - Znajomość zorientowanych obiektowo języków programowania (Pascal, C++, Java, C#) oraz doświadczenie w programowaniu w tych językach,
 - Znajomość podstawowych pojęć obiektowości (klasa, instancja, dziedziczenie, polimorfizm, enkapsulacja),
 - Znajomość technicznych aspektów stosowania rozwiązań obiektowych.

4

Dlaczego analiza i programowanie obiektowe raz jeszcze?

5

Dlaczego warto raz jeszcze zająć się obiektowością?

- Poznanie składni języka nie jest wystarczające, aby skutecznie programować przy pomocy języków zorientowanych obiektowo (C++, Java, C#) - „Posiadanie młotka nie czyni z nas od razu architekta”.
- Programowanie w tych językach staje się efektywne w przypadku poprawnego „myślenia obiektowego”. Stąd konieczne jest przed przystąpieniem do programowania przeprowadzenie analizy i projektowania obiektowego (object-oriented analysis and design – OOA/D).
- Krytycznym i podstawową umiejętnością OOA/D jest umiejętność przypisywania odpowiedzialności (ang. responsibility) poszczególnym obiektom, rozstrzygnięcie w jaki sposób obiekty powinny pozostawać w interakcji między sobą, określanie co poszczególne klasy powinny realizować.
- Zasady analizy i projektowania obiektowego dają się zapisać w postaci zestawu reguł oraz heurystyk.

6

Programowanie zorientowane obiektowo: przypomnienie (?) (1)

- **Modelowanie świata rzeczywistego**
Programowanie zorientowane obiektowo polega na zbudowaniu w programie modeli obiektów ze świata rzeczywistego. Praca programisty polega na ożywieniu tych obiektów oraz sprawienia aby zaczęły ze sobą współpracować.
- **Powtórne użycie kodu (ang. reusability)**
Kod raz zapisany może zostać wykorzystany wielokrotnie. Realizowane jest to dzięki dziedziczeniu (metody z nadklasy są dostępne w klasach pochodnych).
- **Rozszerzalność (ang. extensibility)**
Dzięki polimorfizmowi wprowadzenie nowych klas nie wymaga modyfikowania fragmentów kodu już raz zapisanego. Metody polimorficzne są odpowiedzialne za wykonywanie czynności związanych z daną klasą. W przypadku wprowadzenia nowych klas wprowadzone zostaną odpowiednie metody polimorficzne.

7

Programowanie zorientowane obiektowo: przypomnienie (?) (2)

- **Większa skuteczność w wykrywaniu błędów**
Dzięki związaniu metod z obiektami nie są możliwe błędy niepoprawnego wywołania funkcji z nieuprawnionymi argumentami (metody operują na danych związanych z obiektem). Minimalizowana jest więc ilość błędów niepoprawnego użycia funkcji.
- **Prostota w programowaniu interfejsów użytkownika**
Programowanie zorientowane obiektowo wprost odpowiada potrzebom tworzenia „okienowych” interfejsów użytkownika. Okna mogą być postrzegane jako obiekty, składające się z innych obiektów (przyciski, elementy menu, itd.).

8

Analiza obiektowa

- Celem analizy obiektowej (object-oriented analysis) jest udzielenie odpowiedzi na pytanie: jak system ma działać?
- Zadania realizowane podczas analizy:
 - utworzenie logicznego modelu systemu opisującego sposób realizacji przez system postawionych wymagań (model logiczny pomija większość szczegółów implementacyjnych),
 - określenie podstawowego „słownika” wiedzy dziedzinowej w celu ułatwienia analizy i następnie konstruowania aplikacji.
- Realizowane jest to poprzez znalezienie i opisanie obiektów – rozumianych jako koncepty – w rozpatrywanej dziedzinie.
- Tworzone modele zapisywane są przy pomocy notacji definiowanych w językach modelowania (np. UML).

9

Projektowanie obiektowe

- Celem projektowania obiektowego jest udzielenie odpowiedzi na pytanie: jak system ma zostać zaimplementowany?
- Zadania realizowane podczas projektowania:
 - opracowanie szczegółowego opisu implementacji systemu,
 - zdefiniowanie obiektów wykorzystywanych w programie.
- Realizowane jest to poprzez określenie zasad współpracy obiektów w taki sposób aby zrealizowane były wymagania (określenie atrybutów, metod, itd.).
- Struktura tworzonego oprogramowania powinna jak najbardziej zachowywać ogólną strukturę modelu stworzonego w poprzedniej fazie.
- W fazie projektowania wykorzystuje się tą samą notację, co w fazie analizy.

10

Programowanie obiektowe

- Implementacja (kodowanie) projektu oprogramowania w wybranym środowisku implementacyjnym:
 - wykorzystanie języka obiektowego,
 - powtórne użycie już istniejących bibliotek obiektowych,
 - wykorzystanie narzędzi szybkiego wytwarzania aplikacji,
 - wykorzystanie generatorów kodu.
- Podjęcie kroków mających doprowadzić do wytworzenia niezawodnego oprogramowania:
 - unikanie niebezpiecznych technik,
 - zasada ograniczonego dostępu,
 - stosowanie kompilatorów sprawdzających zgodność typów.

11

UML: Rola modelowania w rozwijaniu aplikacji

- Co to jest model?
Model to pewne uproszczenie rzeczywistości.
- Dlaczego modelujemy?
Konstruujemy modele, aby lepiej zrozumieć system, który tworzymy.
- W czym nam pomaga modelowanie:
 - w wizualizacji systemu takim jakim jest, bądź takim jakim powinien być,
 - w specyfikacji struktury lub zachowania systemu,
 - służy jako wzorzec podczas konstruowania aplikacji,
 - dokumentuje wyniki prowadzonych prac.
- Dlaczego potrzebujemy modeli formalnych (sformalizowanych)?
Zunifikowany język ułatwi komunikację oraz pozwoli opisywać tworzony model w jednorodny sposób.

12

Czym jest UML?

- Czym jest UML:
 - UML (Unified Modelling Language) - zunifikowany język do modelowania obiektowego.
 - UML dostarcza projektantowi systemu aparat do wizualizowania, specyfikowania, konstruowania oraz dokumentowania pojęć oraz mechanizmów które składają się na rozwiązanie rozpatrywanego problemu.
- Charakterstyka UML:
 - rozwijany jest przez następujące osoby: Grady Booch, James Rumbaugh i Ivar Jacobson,
 - następca innych metodologii obiektowych takich jak m. in. OMT (Rumbaugh), OOA/OOD (Coad, Yourdon), OOAD (Booch), Objectory (Jacobson),
 - połączenie teorii z praktyką: Rational Software Corporation.

13

Wzorce projektowe (patterns)

- Wzorce projektowe to zapisany zbiór doświadczeń zaawansowanych projektantów programów zorientowanych obiektowo.
- Wzorce projektowe są zapisany w skodyfikowany sposób pozwalający na opisanie pewnego problemu programistycznego i jego rozwiązania.
- Najbardziej znane wzorce projektowe (23 wzorce) zostały opracowane przez tzw. Gang Czterech (Gamma, Helm, Johnson i Vlissides):
 - Adapter,
 - Factory,
 - Singleton,
 - Strategy,
 - ...

14

Literatura

15

Zalecana literatura

- 1) Craig Larman, Applying UML and Patterns An Introduction to Object-Oriented Analysis and Design and the Unified Process, Prentice Hall, 2002
- 2) A. Holoub Wzorce projektowe, Helion 2005
- 3) A. Shalloway, J. R. Trott Projektowanie zorientowane obiektowo. Wzorce projektowe. Helion 2005
- 4) Rebecca Wirfs-Brock, Alan McKean, Object Design – Roles, Responsibilities and Collaborations, Addison Wesley, 2003
- 5) Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns Elements of Reusable Object-Oriented Software, Addison Wesley, 1995
- 6) Edward Yourdon, Carl Argila, Analiza obiektowa i projektowanie, przykłady zastosowań, WNT, 2000
- 7) James Martin, James J. Odell, Podstawy metod obiektowych, WNT, 1997
- 8) Richard C. Lee, William M. Tepfenhart, UML and C++ A practical guide to object-oriented development, Prentice Hall, 1997

16

Cykl życia oprogramowania

17

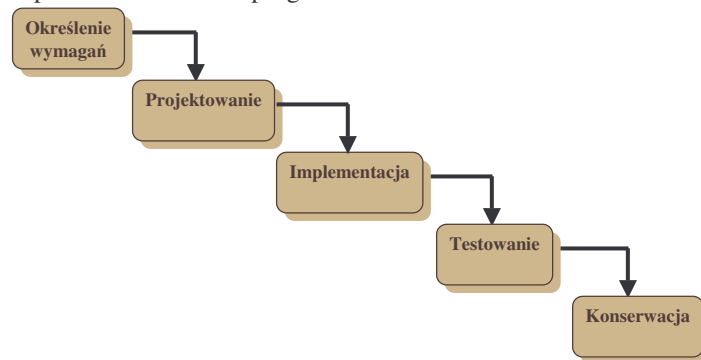
Cykl życia oprogramowania

- Cykl życia oprogramowania to wskazanie kolejnych faz życia oprogramowania, określenie czynności wykonywanych w poszczególnych fazach oraz ustalenie kolejności ich realizacji.
- Cykl życia oprogramowania ma za zadanie opisać proces budowy, uruchamiania oraz utrzymania aplikacji.
- Razem z rozwojem technologii informatycznych oraz pogłębianiem się wiedzy o tym jakie problemy występują podczas tworzenia oprogramowania pojawiały się różne modele opisujące cykl życia oprogramowania:
 - Model kaskadowy.
 - Realizacja przyrostowa,
 - Model spiralny,
 - Prototypowanie,
 - „Metodyki lekkie”.
- W rozwoju oprogramowania bazującego na paradygmacie obiektowym duże znaczenie zyskał Unified Process oraz Rational Unified Process (RUP).

18

Model kaskadowy (model wodospadu)

- Klasyczny model cyklu życia oprogramowania zaproponowany przez analogię do sposobu prowadzenia innych przedsięwzięć inżynierskich.
- Liniowy proces w którym wyróżnia się 5 podstawowych kroków podczas tworzenia oprogramowania:



19

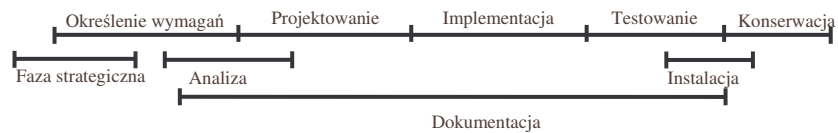
Podstawowe fazy modelu kaskadowego

- Określenie wymagań – określenie celów oraz szczegółowych wymagań wobec tworzonego systemu.
- Projektowanie – szczegółowy projekt systemu spełniającego ustalone wcześniej wymagania.
- Implementacja (kodowanie) – implementacja poszczególnych modułów systemu w konkretnym środowisku programistycznym.
- Testowanie – integracja poszczególnych modułów wraz z testowaniem działania całego systemu.
- Konserwacja – usuwanie błędów, modyfikacje i rozszerzenia funkcji programu w trakcie jego użytkowania.

20

Dodatkowe fazy modelu kaskadowego

- Faza strategiczna – podjęcie strategicznych decyzji dotyczących dalszych etapów prac wykonywanych przed podjęciem formalnych decyzji o uruchomieniu danego projektu.
- Faza analizy – budowanie logicznego modelu systemu.
- Faza dokumentacji – wytworzenie dokumentacji użytkownika.
- Faza instalacji – przekazanie systemu użytkownikowi.
- Następnstwo czasowe faz modelu kaskadowego:



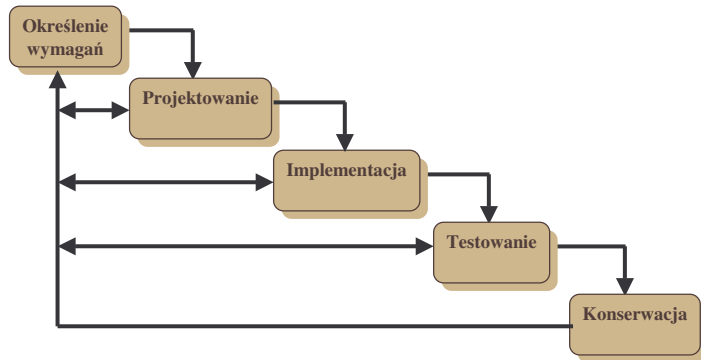
21

Ocena modelu kaskadowego

- Wady modelu kaskadowego:
 - Narzucenie twórcom oprogramowania ścisłej kolejności wykonywania prac,
 - Wysoki koszt błędów popełnianych we wstępnych fazach tworzenia oprogramowania (błędy popełnione w fazie wymagań zostaną wykryte dopiero w fazie testowania),
 - Długa przerwa w kontaktach z klientem (ryzyko utraty zainteresowania klienta realizowanym projektem).
- Przydatność modelu kaskadowego:
 - Przyjmuje się, że w czystej postaci model kaskadowy w praktyce nie jest nigdy stosowany,
 - Fazy modelu kaskadowego wyróżnia się w większości projektów informatycznych.

22

Zmodyfikowany model kaskadowy z iteracjami



23

Unified Process

- Unified Process (UP) opracowany został przez twórców UML (Jacobson, Booch, Rumbaugh) i upowszechniony w 1999 roku.
- Unified Process opiera się na powszechnie akceptowanych praktykach, takich jak cykl iteracyjny oraz rozwój oprogramowania oparty na zarządzaniu zagrożeniami (risk-driven development).
- Cykl iteracyjny polega na:
 - organizacji procesu w krótkie (np. 4 tygodniowe) mini-projekty nazywane iteracjami,
 - każdą z iteracji składa się z analizy wymagań, projektowania, implementacji i testowania,
 - wynikiem każdej iteracji jest uruchamialny, ale nie kompletny system; oprogramowanie rozwijane jest poprzez sukcesywne wprowadzanie nowych funkcjonalności z gwarancją natychmiastowej oceny kierunku rozwoju aplikacji przez odbiorcę końcowego.

24

Zmiany w procesie tworzenia oprogramowania, a UP

- Problemem nie iteracyjnych metodyk rozwoju oprogramowania jest próba pełnego opisania i zamrożenia wymagań w fazie początkowej, następnie opracowania pełnego modelu oraz przystąpienie do programowania.
- Podejście iteracyjne w UP bazuje na fakcie, że zmiany funkcjonalności systemu są naturalne i nieuchronne i jako takie powinny być uwzględniane jako kluczowe w całym procesie tworzenia oprogramowania .
- W każdej iteracji wprowadzone są nowe funkcjonalności; jest wiadome, że w fazach początkowych wybór funkcjonalności może nie być trafiony!
- Dzięki natychmiastowej konfrontacji wytworzonego systemu z użytkownikiem ostatecznym następuje ustabilizowanie funkcjonalności (w systemie są zaimplementowane funkcjonalności, które odpowiadają potrzebom użytkownika).

25

Korzyści z iteracyjnego podejścia w UP

- Wczesne wykrywanie zagrożeń (technicznych, wymagań, celów, itp.).
- Widoczny postęp prac.
- Wczesna informacja zwrotna od użytkownika końcowego pozwalająca na reagowanie w sytuacjach rozmiłowania się z oczekiwaniami.
- Możliwość panowania nad procesem projektowania – brak syndromu „paraliżu projektowego”.
- Podział zadań na iteracje pozwala w miarę rozwoju prac nad oprogramowaniem poprawiać jakość samego procesu realizacji systemu.

26

Fazy UP

- Faza wstępna (ang. Inception) – studium wykonalności projektu pozwalające na ustalenie głównych celów projektu, określenie zakresu i ograniczeń budowanego systemu, wstępne oszacowanie złożoności, określenie potencjalnego ryzyka związanego z projektem.
- Faza opracowanie (ang. elaboration) – iteracyjna implementacja kluczowych elementów aplikacji, identyfikacja większości wymagań, oszacowanie złożoności.
- Faza konstrukcji (ang. construction) – iteracyjna implementacja pozostałych fragmentów aplikacji, przygotowanie do wdrożenia (zbudowanie beta wersji produktu).
- Faza przejścia (ang. transition) – beta testy systemu, uzyskanie wersji gotowej do wdrożenia, poprawianie parametrów systemu.

27

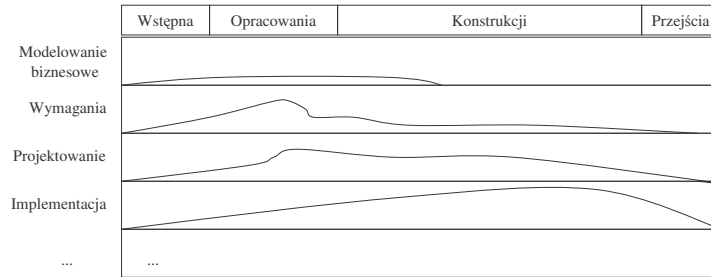
Obszary tematyczne UP

- Unified Process opisuje zadania realizowane w ramach procesu projektowego w tzw. obszarach tematycznych (ang. discipline).
- Obszar tematyczny to zestaw aktywności realizowanych w trakcie cyklu życia oprogramowania oraz powiązanych z nimi artefaktami (wytworzone produkty: kod, dokumenty tekstowe, diagramy, modele, schematy baz danych itp.).
- Podstawowe obszary tematyczne:
 - Modelowanie biznesowe – modelowanie dziedziny przedmiotowej oraz procesów biznesowych,
 - Wymagania – określenie wymagań funkcjonalnych oraz niefunkcjonalnych dla systemu,
 - Projektowanie – wszystkie aspekty projektowania: obiektowe, baz danych, architektury systemu, itp.
 - Implementacja – programowanie i budowanie systemu.

28

Fazy, a obszary tematyczne

- W trakcie realizacji kolejnych faz oraz iteracji prace nad systemem są realizowane w większości obszarów tematycznych jednakże rozkład tych prac zmienia się w czasie:



29

Artefakty tworzone w fazie wstępnej i opracowania

Dokument wizji i przypadków biznesowych	Opisuje cele i ograniczenia systemu na wysokim poziomie
Model przypadków użycia	Opisuje wymagania funkcjonalne i związane z nimi wymagania нефункционалне
Specyfikacje dodatkowe	Opisuje pozostałe wymagania systemowe
Słownik	Terminologia związana z rozpatrywaną dziedziną
Lista zagrożeń i plan zarządzania ryzykiem	Opis zagrożeń biznesowych, technologicznych, zasobów i harmonogramów oraz sposobów radzenia sobie z nimi
Plan iteracji	Opis zadań do zrealizowanie w pierwszej iteracji

30

Określenie wymagań

31

Czym jest określenie wymagań?

- Celem fazy określenia wymagań jest zebranie wymagań klienta wobec tworzonego systemu.
- Zbieranie wymagań to proces w którym użytkownik ostateczny łącznie z przedstawicielami producenta systemu konstruuje zbiór wymagań wobec systemu zgodnie z postawionymi celami i przyjętym zakresem – udzielana jest odpowiedź na pytanie „Co i przy jakich założeniach system ma robić?”.
- Inżynieria wymagań (ang. Requirements engineering) to całość działań związanych z pozyskiwaniem, reprezentowaniem, analizą i zarządzaniem wymaganiami.
- Etap zbierania wymagań jest traktowany jako jeden z najtrudniejszych w całym procesie produkcji oprogramowania.
- Osiągnięcie całkowicie poprawnej i pełnej specyfikacji wymagań jest niezwykle trudne ze względu na ciągłą ewolucję każdego systemu informatycznego (ewolucję dziedziny, którą obejmuje system oraz przede wszystkim ewolucję wyobrażeń użytkownika o tym co system ma robić).

32

Dlaczego określenie wymagań jest trudne?

- Klient z reguły nie wie dokładnie w jaki sposób osiągnąć założone cele. Ponadto cele klienta mogą być osiągnięte na wiele sposobów.
- Duże systemy są wykorzystywane przez wielu użytkowników. Ich cele są często sprzeczne. Różni użytkownicy mogą posługiwać się inną terminologią mówiąc o tych samych problemach.
- Zleceniodawcy i użytkownicy to często inne osoby. Głos zleceniodawców może być w tej fazie decydujący, chociaż nie zawsze potrafią oni właściwie przewidzieć potrzeby przyszłych użytkowników.
- Klient z reguły spodziewa się poprawy stanu aktualnego organizacji, często nie przewidując jakie zmiany w organizacji przedsiębiorstwa spowoduje wprowadzenie systemu informatycznego.

33

Czym jest wymaganie?

- Przez wymagania rozumie się warunki lub zdolności jakie system – a szerzej cały projekt - musi spełniać lub osiągnąć.
- Wymaganie względem systemu informatycznego to zdolność oprogramowania do rozwiązania problemu użytkownika lub osiągnięcia zakładanych przez niego celów lub też zgodność oprogramowania do spełnienia warunków zapisanych w umowie, specyfikacji lub innej formalnej dokumentacji.
- Podstawową rolą fazy zbierania wymagań jest znalezienie wymagań, ich uzgodnienie i zapisanie w postaci czytelnej dla klienta oraz zespołu informatyków realizujących projekt.
- Koszt naprawy błędów w specyfikacji wymagań rośnie zgodnie z zasadą 1:10 tzn. naprawa błędu specyfikacji kosztuje:
 - 10 razy więcej na etapie projektowania,
 - 100 razy więcej na etapie kodowania,
 - 1000 razy więcej na etapie użytkowania i konserwacji,niż na etapie specyfikacji!!!

34

Rodzaje wymagań i sposoby ich zapisu

- Podstawowe rodzaje wymagań:
 - Wymagania funkcjonalne – opisują one funkcje (czynności, operacje) wykonywane przez system, dotyczą rezultatów oczekiwanych przez użytkownika podczas kontaktu z systemem,
 - Wymagania niefunkcjonalne – opisują ograniczenia, przy zachowaniu których system powinien realizować swoje funkcje.
- Sposoby zapisu wymagań:
 - W języku naturalnym (opis systemu przy użyciu języka „potocznego”),
 - W języku naturalnym strukturalnym (język z ograniczonym słownictwem i składnią),
 - Tablice, formularze - wyspecyfikowanie wymagań w postaci (zwykle dwuwymiarowych) tablic, kojarzących różne aspekty,
 - Przypadki użycia - poglądowy sposób przedstawienia aktorów i funkcji systemu (rozwiązanie zalecane w UP).