Inżynieria oprogramowania I

Andrzej Jaszkiewicz

Kontakt

- Andrzej Jaszkiewicz
- p. 424y, Piotrowo 3a
- tel. 66 52 371
- jaszkiewicz@cs.put.poznan.pl
- www-idss.cs.put.poznan.pl/~jaszkiewicz

Literatura

- A. Jaszkiewicz, *Inżynieria oprogramowania*, Helion, Gliwice, 1997.
- B. Begier, *Inżynieria oprogramowania - problemy jakości*, Wydawnictwo Politechniki Poznańskiej, Poznań, 1999.
- Janusz Górski (red.). *Inżynieria oprogramowania w projekcie informatycznym*. Mikom, Warszawa, 2000, wyd. II.
- G. Booch, J. Rambaugh, I. Jacobson, UML przewodnik użytkownika, WNT, Warszawa, 2000.

Literatura

- K. Subieta Wprowadzenie do inżynierii programowania. Wydawnictwo PJWSTK, Warszawa 2002
- S. Szejko (red.), Metody wytwarzania oprogramowania, Mikom, Warszawa, 2002
- M. Fowler, K. Scott, UML w kropelce, Oficyna Wydawnicza LTP, 2002
- S. Maguire, Niezawodność oprogramowania, Helion, Gliwice, 2002
- J. W. Cooper, Java. Wzorce projektowe, Helion, Gliwice, 2001

Rynek oprogramowania 2002

- Świat 207 miliardów euro (USA 97 miliardów, UE 63 miliardy)
  - Bez oprogramowania wytwarzanego na własne potrzeby
- Wzrost 5.1% rocznie
- + 125 miliardów euro dodatkowych usług
- W UE 60-70% oprogramowania jest wytwarzane w firmach, dla których nie jest to główną działalnością

Trochę historii - Rozwój technik wytwarzania oprogramowania

- Lata 50-te
  - Sprzęt o bardzo ograniczonych możliwościach
  - Ograniczone zastosowania
  - Małe programy
  - Programy pisane często dla własnych potrzeb lub potrzeb dobrze znanych osób
  - Dobrze wyspecyfikowane zadania

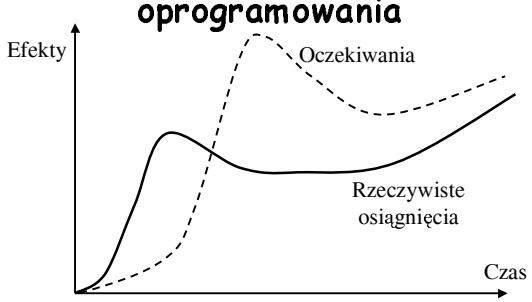
Rozwój technik wytwarzania oprogramowania

- Lata 60-te
  - Profesjonalni programiści
  - Nowe języki programowania – COBOL, Fortran, Algol
  - Sprzęt o dużo większych możliwościach, np. pamięć wirtualna
  - Nowe zastosowania – np. w biznesie
  - Próba realizacji wielu dużych przedsięwzięć programistycznych

Kryzys oprogramowania

- Rozwój technik wytwarzania oprogramowania nie nadąża za rozwojem sprzętu komputerowego
- Czy kryzys oprogramowania trwa do dzisiaj?
  - Nadal większość przedsięwzięć przekracza czas i/lub budżet
  - Około 25% przedsięwzięć programistycznych nie jest kończona
  - 90% firm przyznaje, że dość często zdarzają im się opóźnienia przedsięwzięć
  - Powszechna akceptacja kiepskiej jakości oprogramowania (w pewnych obszarach)

Zależność osiągnięcia-oczekiwania w wytwarzaniu oprogramowania



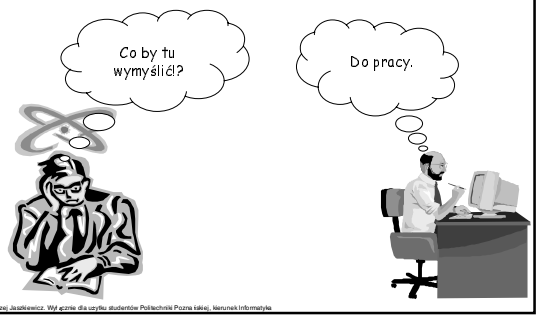
Przyczyny kryzysu oprogramowania

- Duża złożoność systemów informatycznych
- Złożoność, zmienność, nieadekwatność wymagań
- Niepowtarzalność poszczególnych przedsięwzięć
- Nieprzejrzystość procesu budowy oprogramowania
- Pozorna łatwość wytwarzania i modyfikowania oprogramowania
- Potrzeba kreatywności
- Czynnik ludzki
- Mało wymagający rynek
- Niedoskonałość narzędzi

Początek inżynierii oprogramowania

- 1968 NATO Conference on Software Engineering

Podejście amatorskie a inżynierskie



Definicje inżynierii oprogramowania

- Duże systemy wymagające pracy wielu osób – praca grupowa

- Wielowersyjność oprogramowania

Definicja inżynierii oprogramowania

- *Wiedza techniczna*, dotycząca wszystkich *faz cyklu życia oprogramowania*, której celem jest uzyskanie wysokiej jakości *produktu - oprogramowania*.

Jakość oprogramowania

- Użyteczność (*usefulness*)
- Niezawodność (*reliability*)
- Ergonomia (*usability*)
- Efektywność (*efficiency*)
- Łatwość konserwacji (*maintability*)
- Bezpieczeństwo użytkownika (*user safety*)

- Koszt?

Zakres inżynierii oprogramowania

- Wytwarzanie oprogramowania i innych produktów (np. dokumentacji)
- Zarządzanie wytwarzaniem oprogramowania

Plan wykładów - I semestr

- Wprowadzenie i podstawowe modele cyklu życia oprogramowania
- Analiza/modelowanie systemów z wykorzystaniem języka UML, w tym elementy analizy wymagań
- UML jako narzędzie projektowania i dokumentowania oprogramowania
- Projektowanie oprogramowania
- Niezawodność oprogramowania
- Dokumentacja techniczna i użytkowa
- Narzędzia inżynierii oprogramowania

Pan wykładów II semestr

- Większy nacisk na pracę grupową
- Modele dojrzałości: CMMI i ISO 9001. Wprowadzenie do inżynierii wymagań. Planowanie i kontrola przedsięwzięć. Szacowanie rozmiarów i kosztów oprogramowania. Przeglądy oprogramowania. PSP. TSP. Wprowadzenie do PRINCE 2. Zasady skutecznego działania. Programowanie Ekstremalne. Przedsięwzięcia dyplomowe

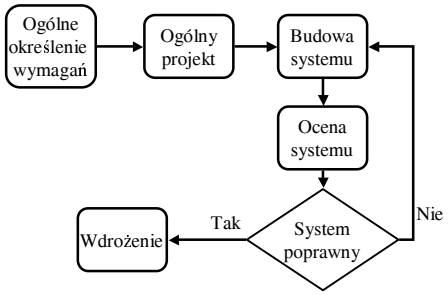
Plan laboratoriów

- Semestr I – seria mniejszych ćwiczeń
- Semestr II - seria mniejszych ćwiczeń (uzupełnienie) + większy projekt

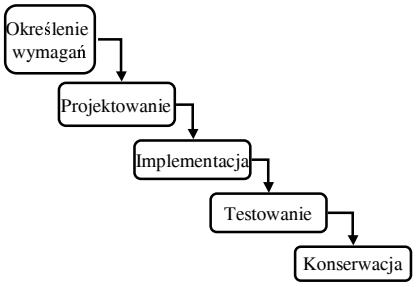
Modele cyklu życia oprogramowania

- Uporządkowanie prac.
- Ustalenie kolejności prac.
- Planowanie i monitorowanie realizacji.

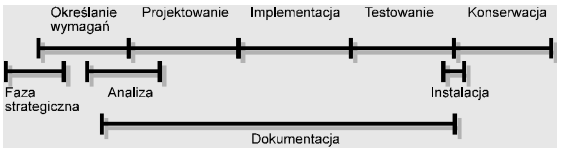
Programowanie odkrywcze



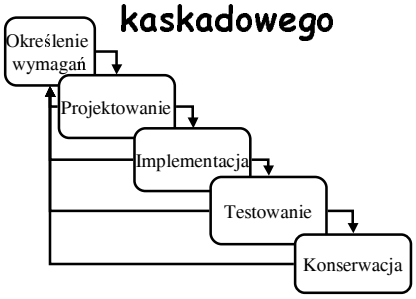
Model kaskadowy



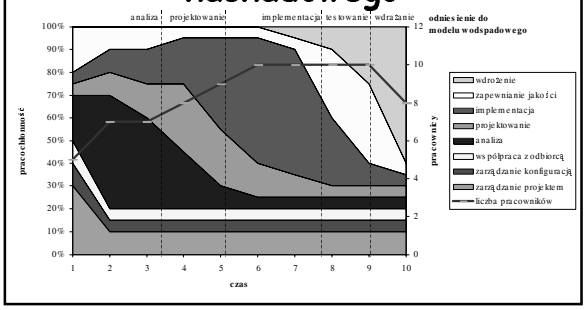
Dodatkowe fazy w modelu kaskadowym



Ścisłe i elastyczne rozumienie modelu kaskadowego



Przykład elastycznego podejścia do modelu kaskadowego



Wady i zalety modelu kaskadowego (rozumianego ściśle)

- + Łatwość zarządzania – planowanie i monitorowanie
- - Wysoki koszt błędów popełnionych we wstępnych fazach
  - Koszt błędu w wymaganiach 100-1000 razy większy od kosztu błędu programistycznego!
- - Długa przerwa w kontaktach z klientem
- - Nie lubiany przez wykonawców

Prototypowanie

- Cel – lepsze określenie wymagań
- Fazy:
  - Ogólne określenie wymagań.
  - Budowa prototypu.
  - Weryfikacja prototypu przez klienta.
  - Pełne określenie wymagań.
  - Realizacja pełnego systemu zgodnie z modelem kaskadowym.

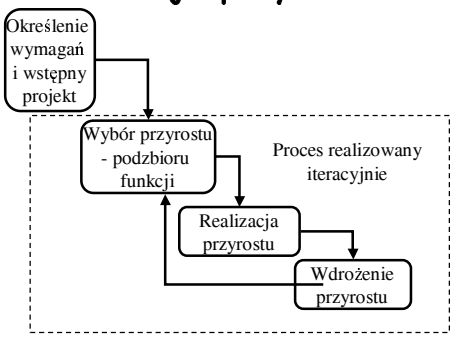
Sposoby budowy prototypu

- Prototyp musi być zbudowany szybko i niskim kosztem.
  - Niepełna realizacja.
  - Języki wysokiego poziomu .
  - Wykorzystanie gotowych komponentów.
  - Generatory interfejsu użytkownika.
  - Szybkie programowanie (*quick-and-dirty programming*).
  - Papier.
  - Programowanie odkrywcze.

Wady i zalety prototypowania

- + Mniejsze ryzyko popełnienia kosztownych błędów we wczesnych fazach.
- + Możliwość szybkiej demonstracji prototypu i szkolenia użytkowników.
- - Koszt budowy prototypu, który może się nie zwrócić.
- - Możliwość nieporozumień z klientem.

Realizacja przyrostowa



Wady i zalety realizacji przyrostowej

- + Możliwość wcześniejszego korzystania z pewnych funkcji systemu.
- + Skrócenie przerw w kontaktach z klientem.
- + Możliwość elastycznego reagowania na opóźnienia.
- - Kłopoty z integracją oddzielnie realizowanych modułów.

Realizacja przyrostowa

- Zalecana w większości lekkich (żwawych) metodyk – np. w programowaniu ekstremalnym – często małe przyrosty (kilka tygodni)
- Dobrze opisuje realizację wielu (zwłaszcza udanych) projektów wolnego oprogramowania (free/open source)

Wybór modelu do konkretnego przedsięwzięcia

- Duże przedsięwzięcia, np. > 6 miesiecy – realizacja przyrostowa, mniejsze m. Kaskadowy
  - W lekkich metodykach także dla mniejszych przedsięwzięć
- Trudności w określeniu wymagań:
  - nowatorski system z punktu widzenia klienta
  - mała znajomość dziedziny problemu przez wykonawcę:

  Jeżeli tak, to prototypowanie

Unified Modeling Language – UML

- Obiektowa notacja graficzna służąca do modelowania, projektowania i specyfikacji oprogramowania
- Następca licznych notacji obiektowych z lat 80-tych i 90-tych
- Powstał na bazie metod Boocha, Rumbaugh (OMT) i Jacobsona – stworzony przez tych właśnie autorów

Unified Modeling Language – UML

- Standard Object Management Group (OMG)
- Wspierany przez firmę Rational
- De facto standard przemysłowy
- Pierwsza wersja w 1997

- **Notacja, a nie metodyka**