INKS105 (*INK9117*)
Podstawy inżynierii oprogramowania

dr Marek Piasecki

Marek.Piasecki@pwr.wroc.pl

http://Marek.Piasecki.staff.iiar.pwr.wroc.pl/dydaktyka/io

Celem kursu jest zaprezentowanie aktualnych technik projektowania i modelowania dużych systemów informatycznych oraz problematyki kierowania projektem programistycznym.

W ramach kolejnych wykładów omawiane są
fazy cyklu życia systemu, metody i narzędzia ich realizacji

Zalecana literatura:

A. Jaszkiewicz – Inżynieria oprogramowania, Helion 1997, 2000.

R. Pressman – Inżynieria oprogramowania, WNT 2004.

I. Somerville – Inżynieria oprogramowania, WNT 2003.

J. Górski – Inżynieria oprogramowania w projekcie informatycznym, MIKOM 1999

PROGRAM WYKŁADU

1. Wstęp. Proces tworzenia Systemów Informatycznych - podstawowe definicje. Źródła i rola inżynierii oprogramowania (kryzys oprogramowania, zakres inżynierii oprogramowania, narzędzia CASE)

2. Pojęcie procesu i projektu (budowy) oprogramowania. Modele cyklu życia oprogramowania (model kaskadowy, realizacja kierowana dokumentami, prototypowanie, model spiralny)

3. Określanie wymagań i analiza. Rodzaje i role notacji wykorzystywanych w fazie analizy, obiektowe i strukturalne metody analizy.

4. Metody projektowania architektonicznego. Projektowanie komunikacji z użytkownikiem. Optymalizacja projektu.

5. Implementacja. Typowe środowiska (języki proceduralne, relacyjne bazy danych, środowiska programistyczne, narzędzia szybkiego wytwarzania aplikacji)

6. Tworzenie dokumentacji. Cele istnienia dokumentacji. Części składowe. Jakość dokumentacji.

7. Weryfikacja i testowanie. Rodzaje testów. Koncepcje testowania względem specyfikacji i względem kodu. Ocena liczby błędów. Bezpieczeństwo oprogramowania.

8. Zarządzanie projektami. Planowanie, analiza ryzyka, standardy, estymacja kosztów, metryki oprogramowania, metody zapewnienia jakości.

9. Zarządzanie konfiguracjami i kontrola wersji. Wybrane narzędzia wspomagające.

10. Podsumowanie wykładu. Kolokwium zaliczeniowe.

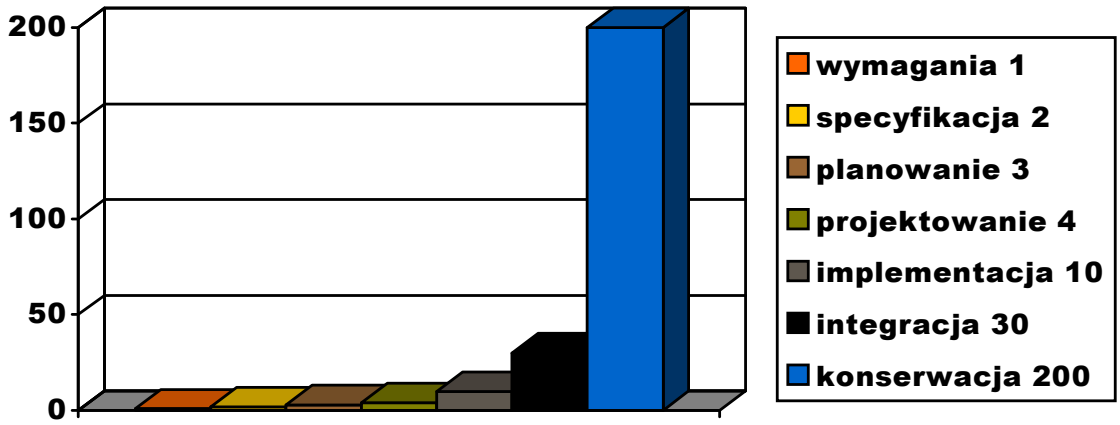
WSTĘP

Oprogramowanie - zbiór programów komputerowych, procedur, zasad działania i danych.

Typowe pytania:

- Dlaczego tak długo trwa opracowanie oprogramowania?
- Dlaczego wytworzenie oprogramowania kosztuje tak dużo?
- Dlaczego nie da się znaleźć wszystkich błędów przed oddaniem oprogramowania klientowi?
- Dlaczego ciągle mamy trudności w mierzeniu postępu w produkcji oprogramowania?

Przybliżony względny koszt naprawy błędu



Mity klienta:

- Ogólne stwierdzenia dotyczące końcowego produktu są wystarczające, by rozpocząć pisanie programu, szczegóły można sprecyzować później.

- Wprawdzie wymagania ciągle się zmieniają, ale to nie szkodzi, bo oprogramowanie jest elastyczne.

REALIA:

27% projektów powstaje w założonym czasie, budżecie i funkcjonalności

33% projektów przekracza czas, budżet i ma mniejszą funkcjonalność

40% projektów jest przerywanych

53% projektów przekracza koszty o 51%

68% projektów przekracza czas o 51%

Produkcja oprogramowania jest procesem składającym się z wielu faz. **Kodowanie (pisanie programów) jest tylko jedną z nich**, niekoniecznie najważniejszą.

- Faza strategiczna
- Faza specyfikacji i analizy wymagań
- Faza projektowania
- Faza konstrukcji (implementacji - kodowania)
- Faza testowania
- Faza konserwacji

INŻYNIERIA OPROGRAMOWANIA

- jest działem informatyki zajmującym się wiedzą techniczną dotycząca wszystkich faz cyklu życia oprogramowania. Traktuje oprogramowanie jako produkt, który ma spełniać potrzeby techniczne, ekonomiczne lub społeczne.
- to zastosowanie systematycznego, zdyscypli-nowanego, ilościowego podejścia do wykonywania, wykorzystywania i konserwowania oprogramowania [IEEE 93]
- to zastosowanie metod naukowych i matematycznych, dzięki czemu możliwości sprzętu komputerowego stają się użyteczne dla ludzi dzięki programom, procedurom i odpowiedniej dokumentacji [Boehm-S.E. Economics 1981]
- to wiedza techniczna dotycząca wszystkich faz cyklu życia oprogramowania, której celem jest uzyskanie wysokiej jakości produktu - oprogramowania [Jaszkiewicz-I.O. 1997]
- dyscyplina, której celem jest przyczynienie się do produkowania oprogramowania o dobrej jakości, oddawanego na czas, mieszczącego się w budżecie i spełniającego potrzeby użytkowników.
- To konsekwentne stosowanie ustalonych przez naukę i zweryfikowanych w praktyce zasad, metod i środków w pracach nad wytwarzaniem i konserwacją oprogramowania oraz ich organizowanie ze względu na cele ekonomiczno-techniczne.

Dwa kierunki rozwoju Inżynierii Oprogramowania

- **formalny** – postulujący jak najszersze stosowanie metod formalnych: języków specyfikacji, transformacji, dowodów poprawności.

  Różne metody dowodzenia, że program spełnia specyfikację:
  - indukcja matematyczna we wszystkich możliwych odmianach;
  - formalna transformacja specyfikacji w spełniający ją program;
  - metody uproszczone - np. logika Hoare'a oparta na niezmiennikach pętli.

- **praktyczny** – stosujący notacje nie w pełni sformalizowane, graficzne; proponujący metody, w których dużą rolę odgrywa wiedza i doświadczenie ludzkie.

  Podejście praktyczne w inżynierii oprogramowania jest słabo zintegrowaną stertą dobrych rad na różne okazje.
  Te rady wynikają:
  - ze starannej obserwacji problemów powstających w trakcie cyklu życiowego oprogramowania,
  - z wielu lat drobiazgowo dokumentowanych doświadczeń,
  - ze zdrowo rozsądkowej refleksji trochę podpartej wiedzą teoretyczną.

Cele inżynierii oprogramowania:

- zorganizowanie prac nad oprogramowaniem, żeby zapanować nad całą tą złożoności¡ i nad wszystkimi etapami życia systemu (nie tylko jego tworzenia na samym początku);
- wypracowanie standardów jakości porównywalnych do obowiązujących w innych dziedzinach inżynierii;
- wypracowanie procedur postępowania sprzyjających wysokiej jakości.

KRYZYS OPROGRAMOWANIA

lata	programy	błędy	inżynieria oprogramowania
1950-1960	programy dla siebie	niewielkie błędy	nie istnieje
1960-1970	duże systemy	kosztowne błędy początki kryzysu	rodzi się
1970-1990	olbrzymie systemy oraz komputery dla mas	wyniszczające błędy uznanie błędów za rzecz zwykłą	w rozkwicie
1990-2003	nic już nie możemy bez komputerów	kryzys oprogramowania w rozkwicie	niezbędnie potrzebna

Symptomy kryzysu oprogramowania:

- zarówno wytwarzanie, jak i utrzymywanie oprogramowania kosztuje zbyt dużo,
- oprogramowanie jest zawodne,
- współdziałanie pomiędzy produktami programistycznymi stanowi poważny problem

Przyczyny kryzysu oprogramowania:

- Złożoność systemów informatycznych, syndrom współczesnych produktów, przekleństwo ciążące na większości projektów i produktów informatyki.
- Szybkie zmiany w przemyśle informatycznym (5-7 miesięcy w porównaniu do 5-7 lat w innych dziedzinach) stanowi to powód do frustracji nie tylko wytwórców oprogramowania, ale także ich klientów.
- Sprzeczność pomiędzy odpowiedzialnością, jaka spoczywa na współczesnych SI, a ich zawodnością wynikającą ze złożoności i ciągle niedojrzałych metod tworzenia i weryfikacji oprogramowania.
- Ogromne koszty utrzymania oprogramowania.
- Niepowtarzalność poszczególnych przedsięwzięć
- Nieprzejrzystość procesu budowy oprogramowania (np. trudności w ocenie stopnia zaawansowania prac),.
- Długi i kosztowny cykl tworzenia oprogramowania, wysokie prawdopodobieństwo niepowodzenia projektu programistycznego
- Pozorna łatwość wytwarzania i dokonywania poprawek
- Długi i kosztowny cykl życia SI, wymagający stałych (często globalnych) zmian.

PROBLEM OGRANICZEŃ CZŁOWIEKA
(twórcy oprogramowania)

Po przekroczeniu pewnego progu złożoności
człowiek przestaje panować nad przygotowywanym produktem,
ponieważ całościowe zrozumienie funkcjonowania systemu
w różnych jego aspektach i na dowolnym poziomie szczegółowości
z zasady przekracza jego możliwości.

Opinie ekspertów:

- programista nie może panować na raz nad więcej niż jednym ekranem tekstu programu
- testowanie może wykazać błędność programu, ale nie może wykazać jego bezbłędności
- nie ma programów bezbłędnych; są tylko takie, w których dotąd nie znaleziono błędu

Co można na to poradzić:

- Mechanizmy abstrakcji
  (eliminacja, ukrycie lub pominięcie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji; wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzaniu pojęć lub symboli oznaczających takie cechy)
- Mechanizmy kompozycji i dekompozycji
  (rozdzielenie złożonego problemu na podproblemy, które można rozpatrywać i rozwiązywać niezależnie od siebie i niezależnie od całości)
- Ponowne użycie
  (wykorzystanie wcześniej wytworzonych schematów, metod, wzorców, komponentów projektu, komponentów oprogramowania)
- Zorientowanie technologii komputerowych na ludzi
  (dopasowanie modeli pojęciowych i modeli realizacyjnych systemów, do wrodzonych ludzkich własności psychologicznych oraz mentalnych mechanizmów percepcji i rozumienia świata)

JAKOŚĆ OPROGRAMOWANIA

- **Poprawność** określa, czy oprogramowanie wypełnia postawione przed nim zadania (tzn., czy spełnia wyspecyfikowane wymagania) i czy jest wolne od błędów.
- **Łatwość użycia** jest miarą stopnia łatwości korzystania z oprogramowania.
- **Czytelność** pozwala na określenie wysiłku niezbędnego do zrozumienia oprogramowania.
- **Ponowne użycie** (inne terminy to: wielokrotne użycie lub wielo-użycie) charakteryzuje przydatność oprogramowania, całego lub tylko pewnych fragmentów, do wykorzystania w innych produktach programistycznych.
- **Stopień strukturalizacji** (modularność) określa, jak łatwo oprogramowanie daje się podzielić na części o dobrze wyrażonej semantyce i dobrze wyspecyfikowanej wzajemnej interakcji.
- **Efektywność** opisuje stopień wykorzystania zasobów sprzętowych i programowych stanowiących podstawę działania oprogramowania.
- **Przenaszalność** mówi o łatwości przenoszenia oprogramowania na inne platformy sprzętowe czy programowe.
- **Skalowalność** opisuje zachowanie się oprogramowania przy rozroście liczby użytkowników, objętości przetwarzanych danych, dołączaniu nowych składników do systemu, itp.
- **Współdziałanie** charakteryzuje zdolność oprogramowania do niezawodnej współpracy z innym niezależnie skonstruowanym oprogramowaniem

CYKL ŻYCIOWY OPROGRAMOWANIA

- Faza strategiczna (określenie strategicznych celów, planowanie i definicja projektu)
- Określenie wymagań
- Analiza
  (dziedziny przedsiębiorczości, wymagań systemowych)
- Projektowanie
  (projektowanie pojęciowe, projektowanie logiczne)
- Implementacja/konstrukcja
  (rozwijanie, testowanie, dokumentacja)
- Testowanie
- Dokumentacja
- Instalacja
- Przygotowanie użytkowników, akceptacja, szkolenie
- Działanie, włączając wspomaganie tworzenia aplikacji
- Utrzymanie, konserwacja, pielęgnacja