Inżynieria oprogramowania II

Wykład 2a:

"Podstawowe zasady RUP"

Marek Krętowski
pokój 206
e-mail: mkret@ii.pb.bialystok.pl
http://aragorn.pb.bialystok.pl/~mkret

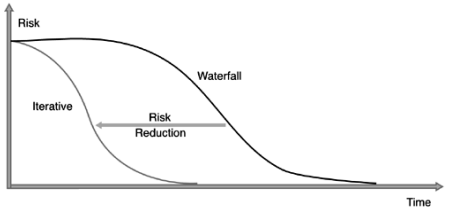
Wersja 1.0

Podstawowe zasady RUP

I. Możliwie wcześnie (i stale) atakuj znaczące zagrożenia (ryzyka), w przeciwnym razie one zaatakują Ciebie
II. Zapewnij, że dostarczasz swojemu klientowi rzeczywistą wartość
III. Pozostań skoncentrowany na działającym oprogramowaniu
IV. Wprowadzaj zmiany możliwie wcześnie
V. Staraj się możliwie wcześnie opracować wykonywalną architekturę
VI. Buduj system na bazie komponentów
VII. Pracuj razem z pozostałymi tworząc jeden zespół
VIII. Niech zapewnianie jakości stanie się nierozerwalną częścią procesu, a nie potencjalnym działaniem wartym rozważenia

I. Atakuj główne zagrożenia

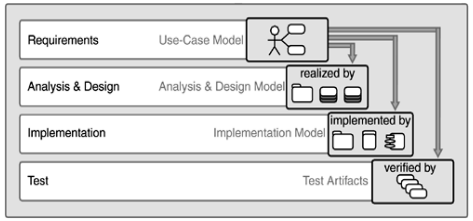
- Jedną z podstawowych korzyści wynikających z zastosowania iteracyjnego procesu wytwórczego jest możliwość rozpoznania i ograniczenia zagrożeń możliwie wcześnie



- Na początku każdej iteracji tworzymy bądź uaktualniamy wykaz głównych zagrożeń => przypisujemy priorytety => tak planujemy działania w ramach iteracji aby zmniejszyć ryzyko związane z głównymi zagrożeniami
- Zagrożenia należy rozpatrywać przede wszystkim biorąc pod uwagę: wymagania, projektowanie i testowanie
- Nie wolno zapominać, że walka z zagrożeniami powinna trwać przez cały okres realizacji przedsięwzięcia oraz, że wykaz zagrożeń i ich istotność może ulegać dynamicznym zmianom

II. Dostarcz rzeczywistą wartość klientowi

- Kluczowym elementem jest uzyskiwanie opinii i akceptacji klientów
- Wykorzystanie przypadków użycia (traktowanych jako opis wymagań funkcjonalnych; „podręczniki użytkownika", ale bez informacji o interfejsie) do komunikacji z użytkownikami
- Przypadki użycia umożliwiają wszystkim członkom zespołu bezpośredni dostęp do wymagań podczas projektowania, kodowania, testowania czy tworzenia podręczników
  - sterowanie procesem
  - wymuszają ciągłe koncentrowanie się na perspektywie użytkownika i umożliwiają ocenę przedsięwzięcia pod kątem wymagań użytkownika
  - pozwalają na rozważanie potrzeb użytkownika podczas planowania przedsięwzięcia i podczas zarządzania jego zakresem



III. Skoncentrowanie się na oprogramowaniu

- Miarą postępu realizacji powinno być oprogramowanie **wykonywalne** (już działające)
  - dążenie do zademonstrowania działającego programu i przeprowadzenie testów
  - na podstawie testów i liczby błędów oszacowanie rzeczywistego postępu
  - nie oznacza to lekceważenia informacji zawartych w pozostałych dokumentach, ale rozważanie ich w oderwaniu od programu może prowadzić do zniekształceń
- Zmniejsza się ryzyko nadmiernego teoretyzowania i analizowania
  - zakończenie prac => wykonywalny program => skuteczne ograniczenie ryzyka
- Wszelkie artefakty niebędące faktycznym oprogramowaniem są jedynie artefaktami pomocniczymi
  - jeśli masz wątpliwości czy tworzyć dany artefakt to zapewne nie jest on niezbędny i prawdopodobnie można z niego zrezygnować
    - nie powinno to być jednak usprawiedliwieniem braku wykonania istotnych czynności!
  - korzyści wynikające z tworzenia wielu artefaktów zwykle ujawniają się i rosną wraz ze zwiększaniem się rozmiaru przedsięwzięcia, stopnia skomplikowania stosunków z udziałowcami, z rozproszeniem zespołu, z kosztami zapewniania jakości, ...

IV. Możliwie wczesne zmiany

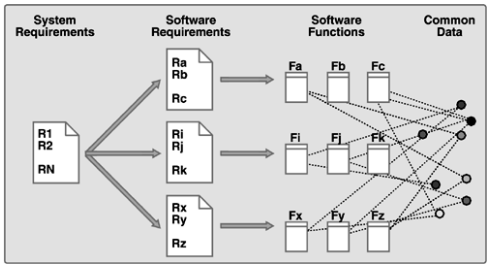
- Współczesne systemy są zwykle zbyt złożone, aby udało się sformułować idealne wymagania na starcie przedsięwzięcia
  - zmiany są nie tylko niezbędne, ale i wskazane, gdyż prowadzą do doskonalenia produktu
  - często zmiany wynikają z lepszego zrozumienia problemu
- Różne rodzaje zmian mają różny wpływ na kolejnych etapach realizacji przedsięwzięcia
  - uzgodniona wizja (koniec I etapu); solidna architektura (II); zamrożenie wprowadzania nowej funkcjonalności (III)
- Zmiany mogą jednak być uciążliwe => niezbędne jest zapanowanie nad procesem ich wprowadzania (zarządzanie zmianami)
  - ustalone procedury podejmowania decyzji co do wprowadzania zmian (zatwierdzanie zmian; komisja kontroli zmian – Change Control Board)
  - możliwość oceny wpływu zmian (śledzenie powiązań między wymaganiami, projektem kodem i testami)
  - minimalizowanie kosztu zmian

V. Architektura wykonywalna

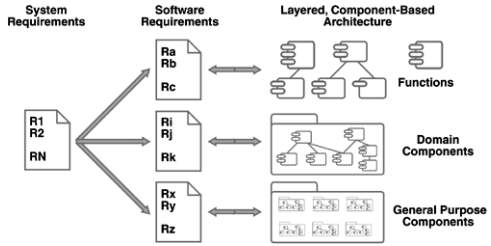
- Wiele zagrożeń przedsięwzięcia jest związanych z architekturą, zwłaszcza gdy tworzona jest I generacja danej aplikacji
- Architektura obejmuje najważniejsze elementy składowe systemu oprogramowania oraz ich interfejsy
  - stanowi szkieletową strukturę systemu i obejmuje zwykle 10-20% kodu końcowego
  - mechanizmy architektoniczne – wspólne rozwiązania typowych problemów (np. sposoby zapewniania trwałości danych)
- Właściwa i odpowiednio przetestowana architektura daje pogląd na elementy składowe i komponenty wymagane w produkcie końcowym
  - pozwala na precyzyjniejsze oszacowanie niezbędnych zasobów i czasu
  - ułatwione powiększanie zespołu i wprowadzanie nowych lub niedoświadczonych członków

VI. Buduj system z komponentów

- Architektury oparte na dekompozycji funkcjonalnej prowadzą do odseparowania funkcjonalności i danych => bardzo wiele zależności => brak elastyczności => utrudnione modyfikacje i rozwój systemu
  - np. zmiana sposobu przechowywania danych wymusza przebudowę wielu funkcji



- Architektury oparte na komponentach (ang. *component-based architecture*) są odporniejsze na zmiany poprzez zgrupowanie danych i funkcjonalności operującej na danych w komponencie
  - wykorzystanie enkapsulacji
  - interfejsy



VII. Współpraca tworzy zespół

- Iteracyjność procesu zwiększa potrzebę ścisłej współpracy w zespole
  - nie sprawdzają się struktury funkcjonalne (grupowanie kompetencyjne)
  - potrzebne zespoły interdyscyplinarne (międzyfunkcjonalne) złożone ze specjalistów wszystkich dyscyplin (analityków, programistów, testerów, ...)
  - w dużych przedsięwzięciach pomocne jest stworzenie grupy skupionej wokół architektury - „zespół zespołów”podejmujący decyzje architektoniczne (strukturalizacja i interfejsy) i standardowe zespoły odpowiedzialne za poszczególne podsystemy
- Odpowiednia infrastruktura komunikacyjna
  - do każdego członka zespołu powinny docierać właściwe informacje => sprzyjanie synchronizacji i inżynierii odwrotnej (reverse engineering)
  - minimalizowanie zbędnej dokumentów => zachęcanie do komunikacji „twarz w twarz”
- Ścisła współpraca w zespole wymusza poczucie współautorstwa produktu końcowego

VIII. Jakość jako część stylu pracy

- W praktyce często po prostu tańsze jest odnajdowanie rozwiązania problemu poprzez wczesną implementacje i eksperymentalną weryfikację (testowanie) niż poprzez szczegółowe przeglądy i analizy projektu
- Iteracyjność wymusza nie tylko wcześniejsze, ale i częstsze testowanie
  - testowanie regresyjne
  - automatyzacja testów w celu zminimalizowania kosztów