Dr Katarzyna Grzesiak-Kopeć

Inżynieria oprogramowania

5. Techniki IO

Plan wykładu

- Tworzenie oprogramowania
- Najlepsze praktyki IO
- Inżynieria wymagań
- Technologia obiektowa i język UML
- Techniki IO
- Metodyki zwinne
- Refaktoryzacja
- Mierzenie oprogramowania
- Jakość oprogramowania
- Programowanie strukturalne
- Modelowanie analityczne
- Wprowadzenie do testowania

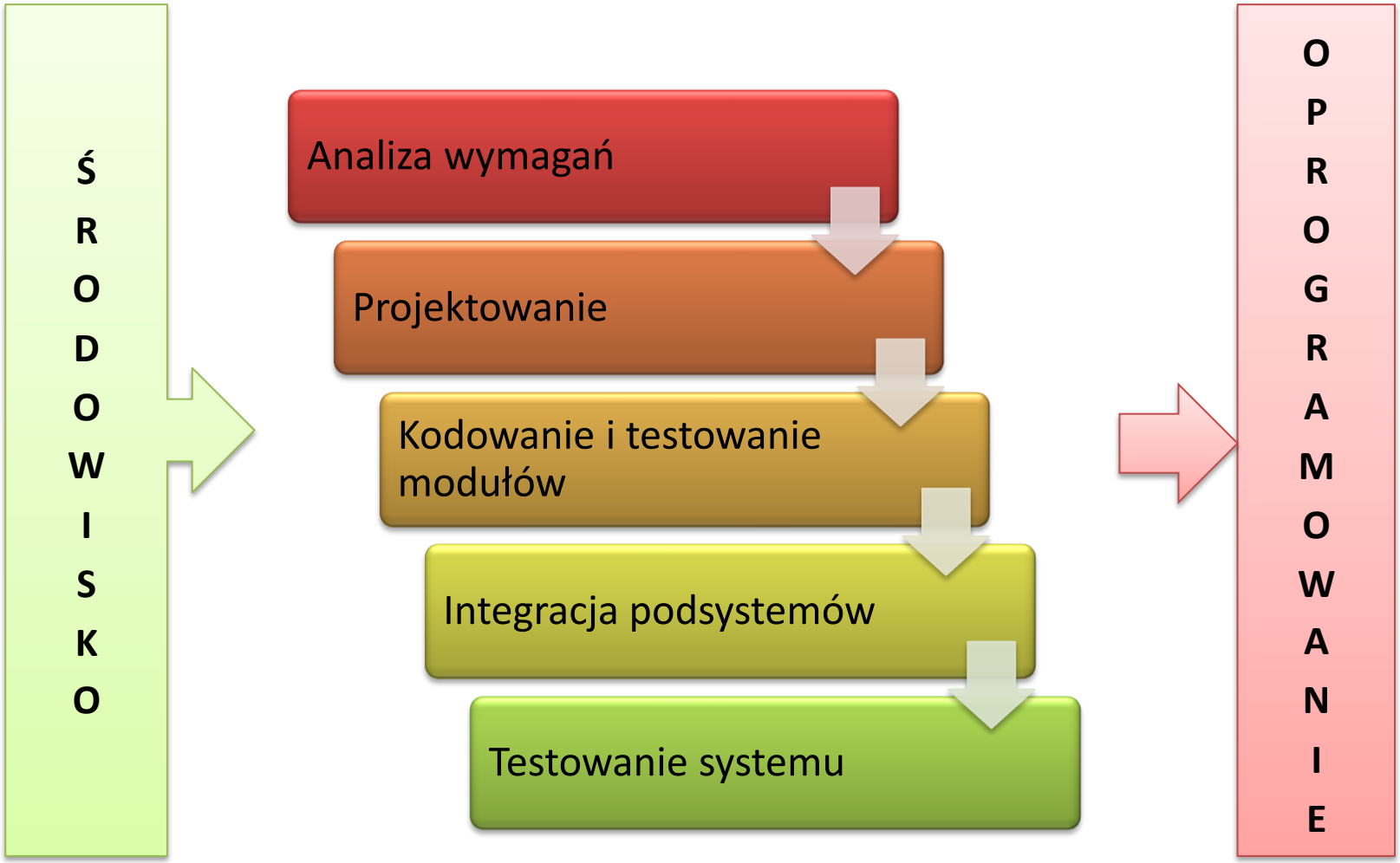
Recepty na dobry proces

- Wzorce procesów
  - Task process patterns, Stage process patterns, Phase process patterns
- Modele procesów
  - technika wodospadu, model spiralny, model oparty na metodach formalnych, prototypowanie, Rapid Application Development (RAD), Component Based Development (CBD), Concurrent Development, Disciplined Software Development, Aspect-Oriented Software Development, Agile Process Models, The Rational Unified Process (RUP)

Metodyka

- Metodyka – najlepsze praktyki w działaniu?
- Grupuje w całość:
  - Notację – język modelowania
  - Techniki – sposoby postępowania
  - Proces techniczny – uporządkowanie wytwarzania poprzez wskazanie atomowych zadań, określenie kolejności ich wykonania i zależności pomiędzy nimi

Cykl życia oprogramowania



Cykl życia oprogramowania

My	Niektórzy	Inni	Tamci	Jeszcze inni
Analiza środowiska	Wymagania	Wymagania zewnętrzne	Wymagania	Specyfikacja
Analiza wymagań	Specyfikowanie	Wymagania wewnętrzne	Projektowanie architektoniczne	Projektowanie wysokiego poziomu
Projektowanie	Projektowanie	Projektowanie	Projektowanie szczegółowe	Projektowanie niskiego poziomu
Kodowanie	Kodowanie	Kodowanie	Kodowanie	Realizacja
Testowanie	Testowanie	Testowanie	Testowanie	Testowanie
Wdrażanie	Wdrażanie	Wdrażanie	Wdrażanie	Wdrażanie

Krytyka wodospadu

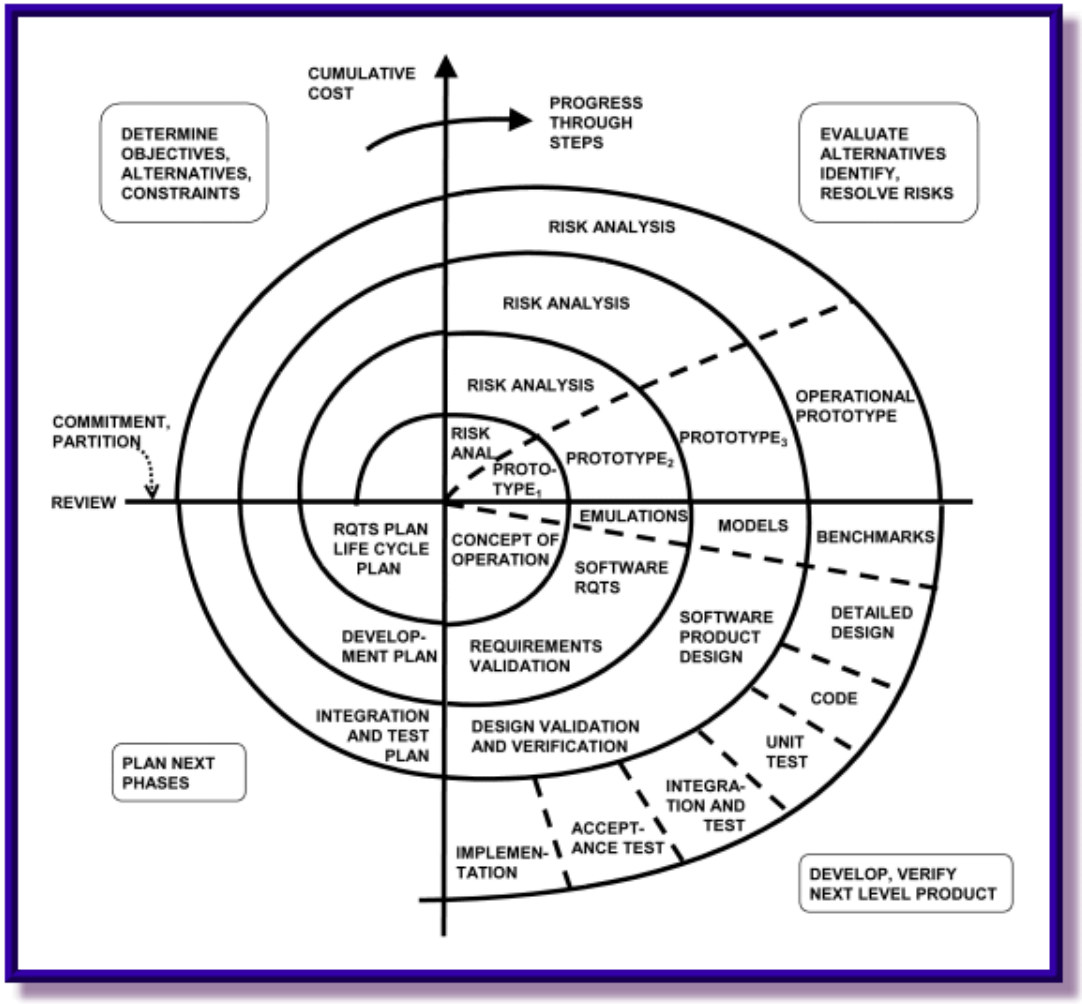
- Sekwencyjnie w praktyce?
- Klient określi na początku wymagania?
- Użytkownik jest cierpliwy?

Model spiralny (Boehm 1988)

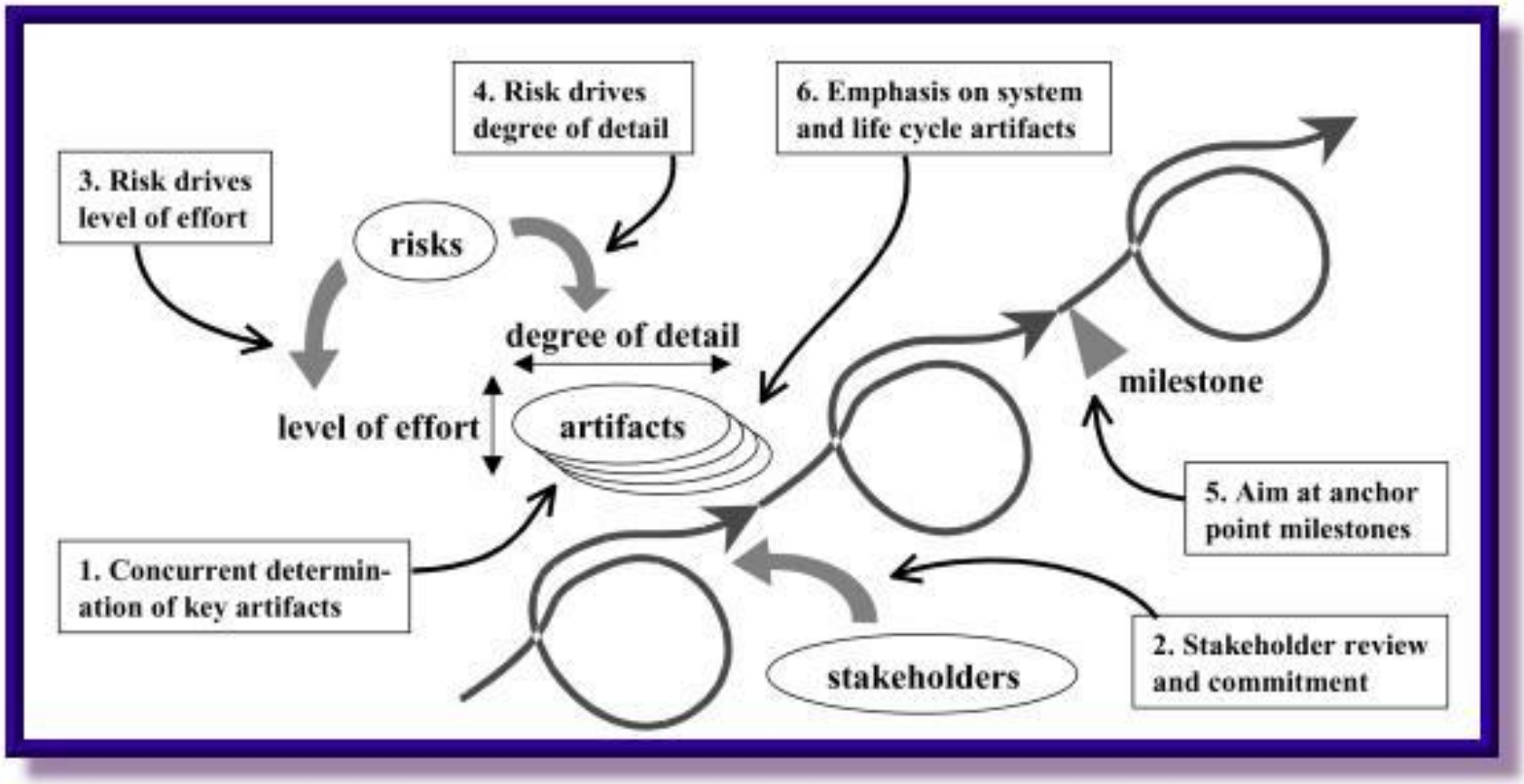
- Wariant kaskadowego cyklu życia
- Sterowany ryzykiem
- Cykliczny (iteracje)
- Punkty milowe

Model spiralny

http://www.stsc.hill.af.mil/crosstalk/2001/05/boehm.html



Model spiralny



Metody formalne

Metody formalne wzbudzają wiele kontrowersji. Ich zwolennicy twierdzą, że mogą zrewolucjonizować tworzenie oprogramowania, przeciwnicy, że są zbyt skomplikowane do stosowania w praktyce. Większość wie o nich zbyt mało, żeby ocenić ich przydatność.

Anthony Hall

Metody formalne

- Kompletne, spójne i jednoznaczne specyfikacje
- Systemy opisywane za pomocą wymagań (faktów)
- Fakty zapisane jednoznacznie w notacji opartej na logice i teorii zbiorów
- Analiza specyfikacji – dowód poprawności, niesprzeczności

Metody formalne

Minusy „nieformalne”

- Sprzeczności
- Niejednoznaczności
- Niejasności
- Niezupełność
- Różne poziomy abstrakcji

Plusy matematyki

- Zwięzły i dokładny opis zjawisk, obiektów i rezultatów
- Płynne przejścia pomiędzy etapami tworzenia oprogramowania
- Notacja jest
  - Abstrakcyjna (modelowanie!)
  - Precyzyjna
  - Umożliwia weryfikację: poprawności, niesprzeczności

Dekalog formalistów

Bowan&Hinchley

1. Wybierz odpowiednią notację
2. Formalizuj, ale nie przesadzaj
3. Oceniaj koszty
4. Zapewnij sobie pomoc ekspertów
5. Nie porzucaj znanych metod
6. Dokumentuj
7. Nie rezygnuj z kontroli jakości
8. Nie bądź dogmatyczny
9. Testuj!
10. Dbaj o powtórne wykorzystanie

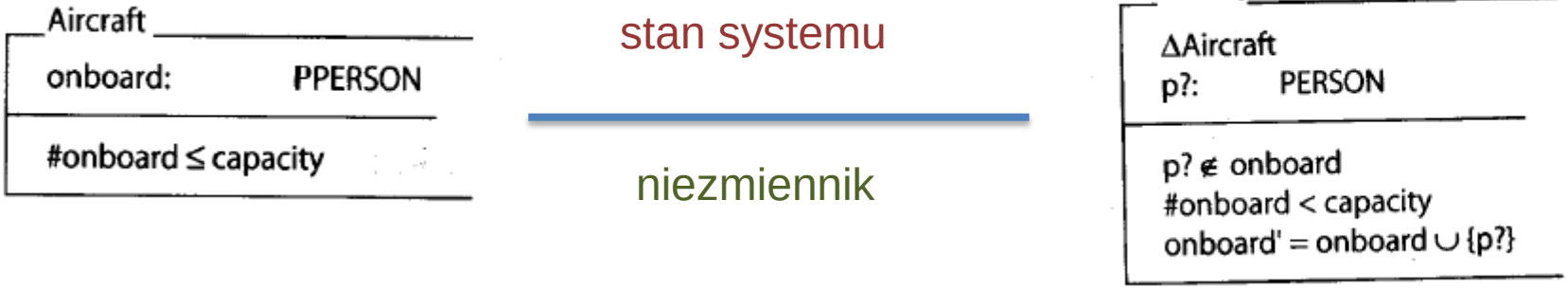
Przykłady

- Analiza
  - OSA
- Specyfikacja
  - Z, BNF, VDM, CSP, Gypsy
- Interfejs
  - Larch
- Systemy dowodzenia
  - HOL, LCF, Boyer-Moore
- Sieci Petriego
  - matematyczna reprezentacja dyskretnych systemów rozproszonych

Specyfikacja Z

Richard Lawlor

- Samolot – stan systemu
- Operacja wejścia na pokład



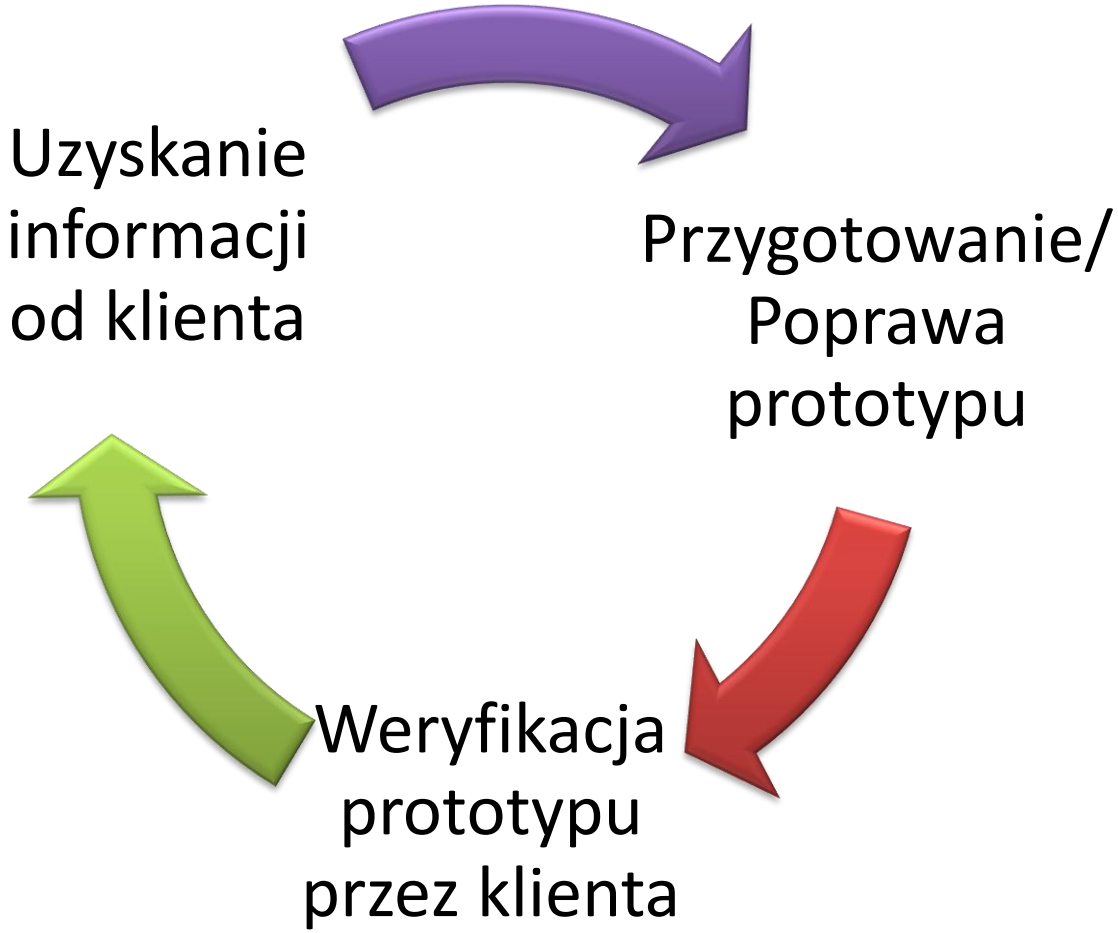
Prototypowanie

- Demonstracja i ustalenie wymagań
- Prototyp może być wykorzystany do szkolenia użytkowników
- Testowanie

Zalety prototypowania

- Wyjaśnienie nieporozumień na linii klient-programista
- Ustalenie brakujących wymagań
- „Działający” system na wczesnym etapie prac
- Prototyp może być punktem wyjścia do stworzenia specyfikacji systemu

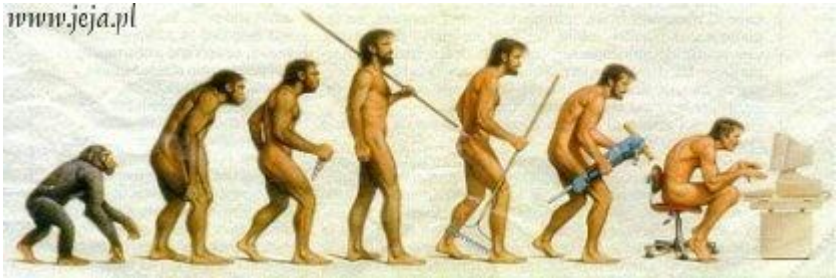
Model prototypowania



Cel prototypowania

Evolutionary prototyping

- Dostarczenie klientowi „działającego” systemu
- Rozpoczynamy od najlepiej zrozumiałych wymagań



Throw-away prototyping

- Ocena lub uzyskanie wymagań
- Zaczynamy od najgorzej zrozumiałych wymagań

Prototypowanie ewolucyjne

- Nie mamy zadanej pełnej specyfikacji
  - np. systemy AI
- Stosujemy techniki pozwalające na szybkie iteracje
- Ciągłe zmiany często skutkują poważną przebudową systemu – drogie utrzymanie
- Potrzebni specjaliści
- Taki system nie będzie „żył” długo

Prototypowanie „do kosza”

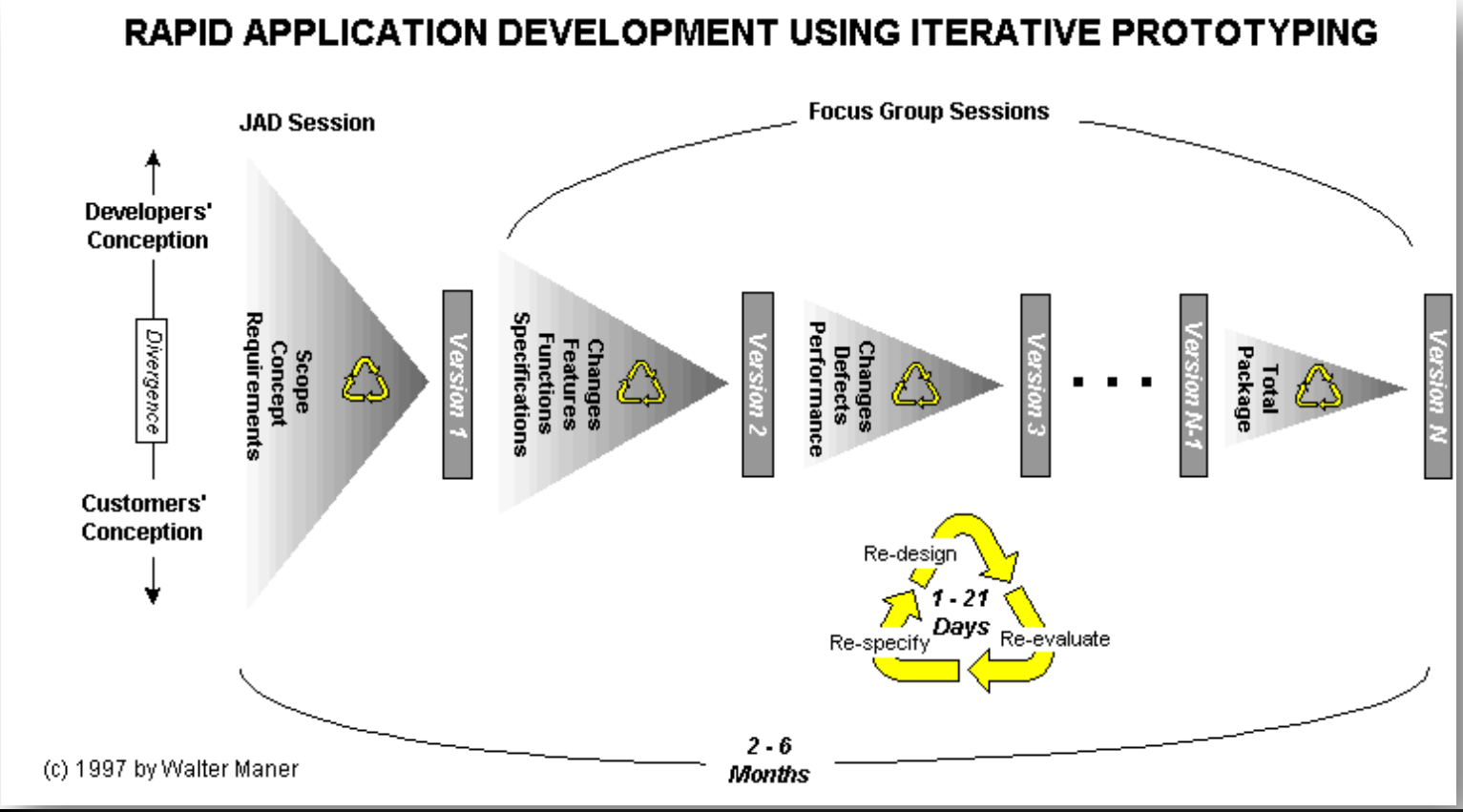
- Stosowany by zmniejszyć ryzyko błędów związanych z wymaganiami
- Prototyp jest
  - tworzony na podstawie pierwotnej specyfikacji,
  - poddawany testom,
  - i wyrzucany do kosza
- Prototyp nie może być produktem finalnym
  - szczątkowy
  - jak utrzymać?
  - kiepsko wykonany

Techniki prototypowania

- Języki specyfikacji formalnej
- Języki wysokiego poziomu
- Generatory aplikacji & 4GLe
- Składanie komponentów

Rapid Application Development

- Pozwala zbudować system w bardzo krótkim czasie (60-90 dni), często idąc na pewne kompromisy



To RAD or not to RAD

- Zakres projektu
  - Wąski, precyzyjnie określony, jasne wymagania
- Dane
  - Istnieją lub są zadane częściowo
  - Projekt dotyczy analizy danych i raportowania
- Decyzje
  - Mała grupa dyspozycyjnych osób
- Zespół
  - Mały, najlepiej liczący do 6 osób

To RAD or not to RAD

- Architektura systemu
  - Zdefiniowana i jasna
  - Kluczowe komponenty technologiczne są dostępne i przetestowane
- Wymagania techniczne
  - czas reakcji, wydajność, rozmiar bazy, etc.
  - Sensowne i adekwatne do architektury
    - Mniej niż 70% oficjalnie dla technologii

RAD czyli szybko!

- Struktura procesu stawia na szybkość (listy zadań)
- Techniki zarządzania
  - Prototypowanie
  - Iteracje
  - Timeboxing

RAD z sukcesem

- Niezależna aplikacja
- Zbudowana z istniejących komponentów
- Wydajność nie jest krytyczna
- Wąska dystrybucja (domowa☺, vertical market)
- Niezawodność nie jest krytyczna
- Używana technologia co najmniej rok istnieje na rynku

Component-Based Development

- Tworzenie oprogramowania z gotowych komponentów
  - z różnych źródeł
  - napisanych w dowolnym języku
  - działających na różnych platformach
  - technologia OO

Komponent

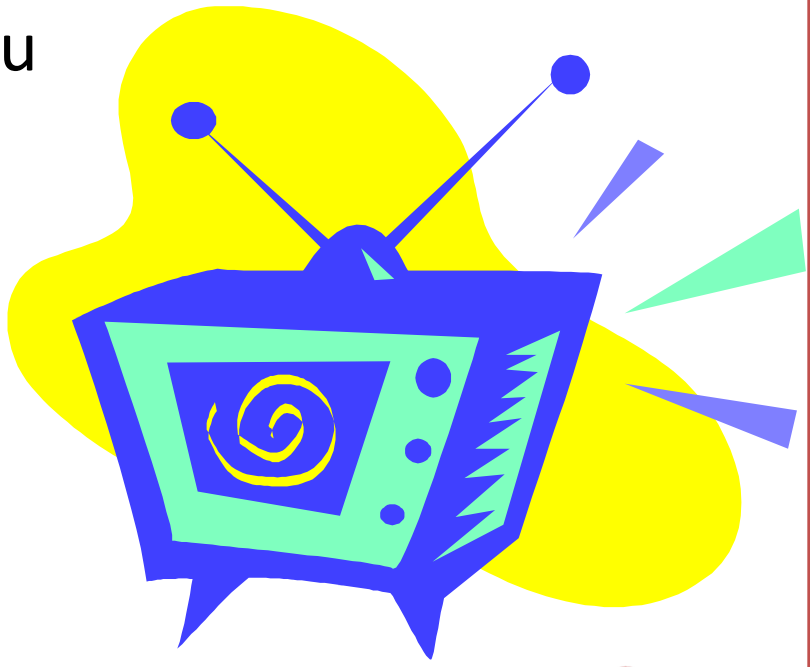
- Fizyczna i zastępowalna część systemu, która realizuje zestaw interfejsów - the UML User's Guide, Booch, 1999
- Czarna skrzynka posiadająca zewnętrzną specyfikację, która jest niezależna od wewnętrznych mechanizmów

Charakterystyka komponentu

- Wewnątrz
- Na zewnątrz – usługi, towary ☺
- Relacje – in/out
  - specyfikacja, implementacja, hermetyzacja
- Kontekst

Ważne terminy

- Ponowne użycie (reuse)
- Hermetyzacja (encapsulation)
  - Ukrywanie implementacji
  - Korzystanie tylko z interfejsu



Kilka głosów za CBD

- Mniejsze koszty
- Im częściej używany komponent, tym jest on cenniejszy
- Szybka i niekosztowna personalizacja aplikacji
- Mniej błędów ☺

Concurrent Development

- Szybciej, szybciej, szybciej!
- Przejście z modelu sekwencyjnego do współbieżnego
- Redukcja czasu kosztem zwiększenia złożoności procesu
  - Stąd częste porażki

Trudności ze współbieżnością

- Większa częstotliwość i liczba wymian informacji pomiędzy fazami projektowymi
- Więcej zadań rozpoczyna się z niekompletnymi lub wstępnymi wymaganiami
- Sekwencyjne zarządzanie zawodzi

Zyski ze współbieżności

Czas budowy	30%-70% krótszy
Czas wypuszczenia	20%-90% krótszy
Ogólna jakość	200%-600% wyższa
Zyski	5%-50% większe
Zwrot kosztów	20%-120% wyższy
Wydajność ekspertów	20%-110% wyższa
National Institute of Standards & Technology, Thomas Group Inc. Published in Business Week Special Report, 30.IV.1990	

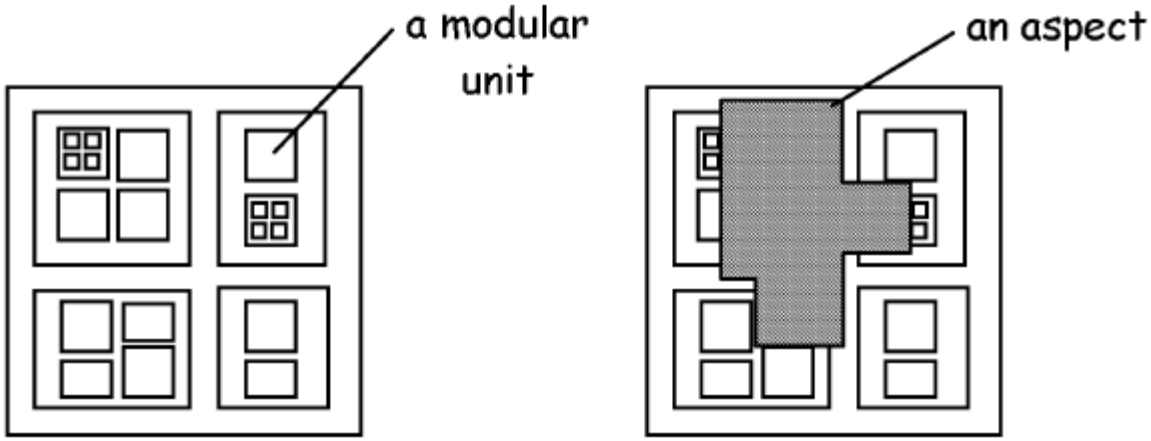
Aspect-Oriented Software Development

- http://aosd.net/
- Aspect-Oriented Programming
  - Gregor Kiczales, Xerox PARC 97

Dekompozycja problemu

- Funkcjonalna
  - Obiekty, moduły, procedury, itd.
- Aspektowa
  - Dotyczą więcej niż jednego elementu funkcjonalnego, np. synchronizacja, komunikacja
  - Nie da się tego „czysto” zrobić stosując takie techniki jak OO

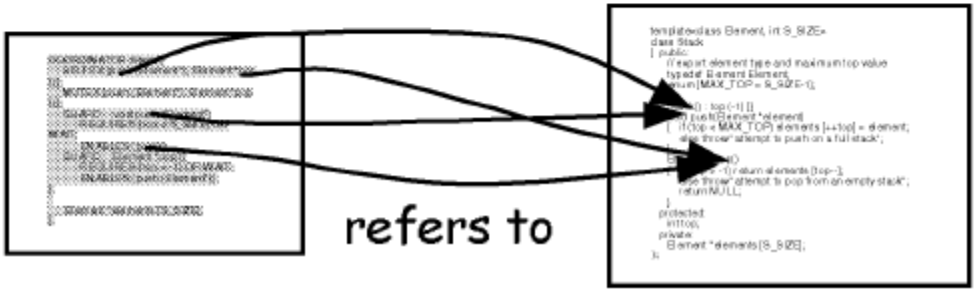
Moduł vs aspekt



K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Techniques, and Applications*. Addison-Wesley, 1999.

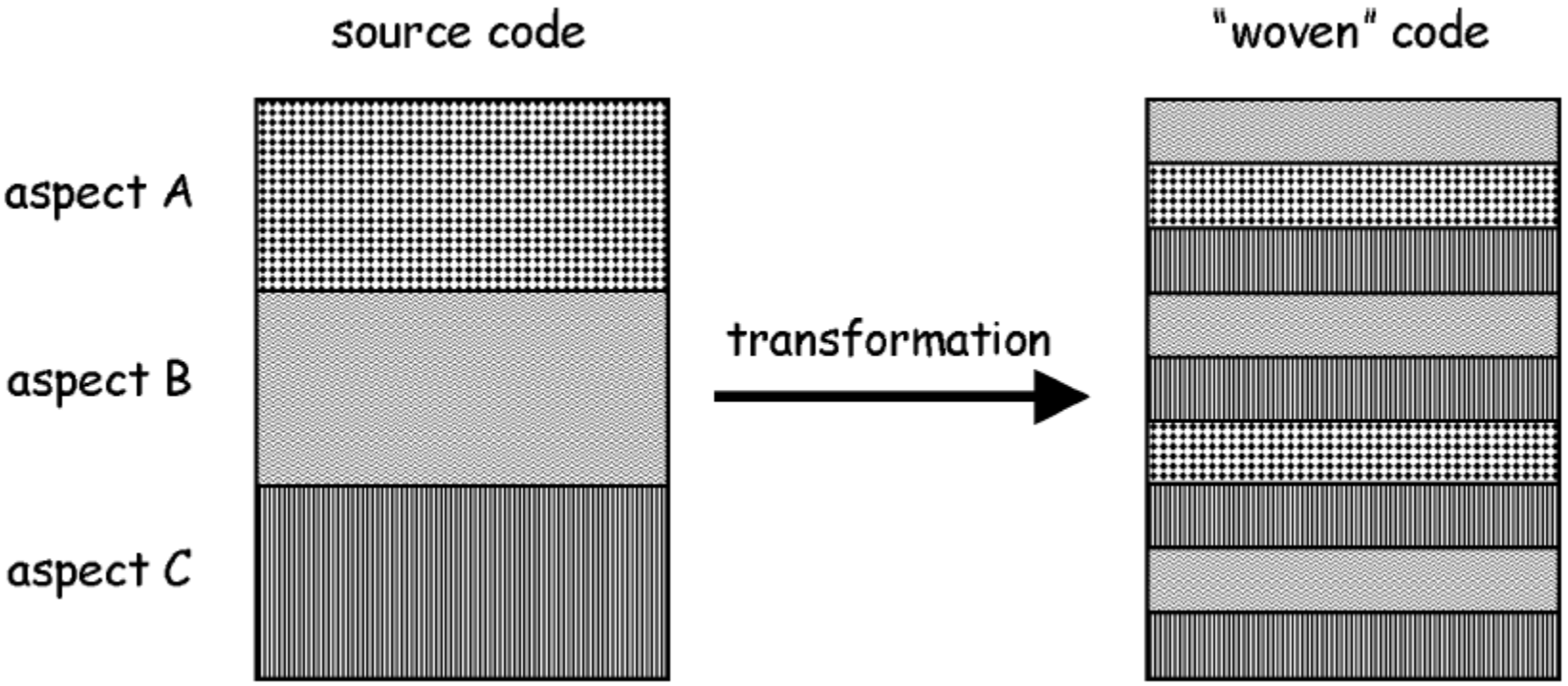
Rozdzielenie struktur

code with merged crosscutting structures



code with separated crosscutting structures

Tkanie (weaving)



Person owns Dog...


```
/**
 * not a dog in sight...
 */
public class Person {

   private String lastName;
   private Address address;
   ...

   public String getLastName() { return lastName; }

   ...

}
```

```
/**
 * not a person in sight...
 */
public aspect DogOwnership {

   public interface IDogOwner {};

   private int IDogOwner.numDogsOwned;

   public int IDogOwner.getNumDogsOwned() { return numDogsOwned; }

}
```

```
public aspect PersonOwnsDog {
   declare parents : Person implements DogOwnership.IDogOwner;
}
```

```
Person person = new Person();
person.getNumDogsOwned();
```

```
...
if (person instanceof DogOwnership.IDogOwner) {
  ((DogOwnership.IDogOwner)person).getNumDogsOwned();
}
...
```

Cel AOD

- Dostarczenie metod i technik umożliwiających funkcjonalną i aspektową dekompozycję problemu
- Aspekty „przecinają” komponenty funkcjonalne
- Dostarczenie metod i technik umożliwiających złożenie komponentów funkcjonalnych i aspektów w spójną implementację systemu

Przykładowe aspekty

- Synchronizacja
- Kontrola błędów
- Zarządzanie pamięcią
- Bezpieczeństwo
- Przechowywanie (persistency)
- Testowanie
- Reprezentacja danych
- Optymalizacja (specyficzna)

Przykładowe aspekty

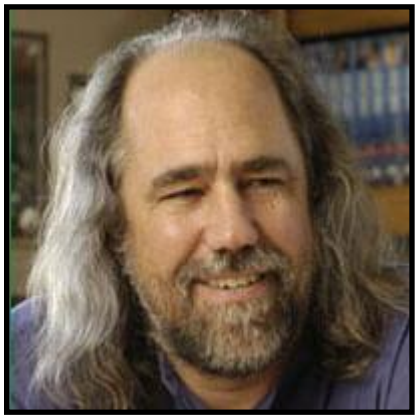
- Systemy rozproszone
  - Interakcja komponentów
  - Synchronizacja
  - Zdalne wywołanie
  - Strategie przekazywania parametrów
  - Replikacja
  - Obsługa wyjątków
  - Rozproszone transakcje
  - Równoważenie obciążenia (load balancing)

Implementacja

- C#/VB.NET
  - Puzzle.Naspect, AspectDNG, Aspect# …
- Java
  - AspectJ, Jakarta Hivemind, JBoss AOP, InjectJ …
- C/C++
  - AspectC++, XWeaver project, AspectC …
- Python, PHP, JavaScript, Perl, Common Lisp, XML, ColdFusion, inne

The Rational Unified Process

- 1995 Rational Software "three Amigos"
  - Grady Booch
  - Ivar Hjalmar Jacobson
  - James Rumbaugh



The Rational Unified Process

- 2003 IBM przejmuje Rationala
- RUP ⇨ IBM Rational Method Composer
  - http://www-128.ibm.com/developerworks/rational/library/nov05/kroll/
  - Eclipse Process Framework

The Spirit of the RUP

- Od początku i na bieżąco rozwiązuj najbardziej ryzykowne kwestie
  - Lista największych zagrożeń
- Koncentruj się na działającym oprogramowaniu
  - Plany i projektowanie jest dobre – działające oprogramowanie jest lepsze
  - Realizuj tylko niezbędne artefakty

Per Kroll, Director, RUP Development and Product Management Teams

The Spirit of the RUP

- Koncentruj się na potrzebach klienta
  - Iteracje
  - Przypadki użycia
    - Projektowanie, implementacja i testowanie skoncentrowane na potrzebach klienta
  - Dokumentowanie wymagań jest dobre –realizacja lepsza

Per Kroll, Director, RUP Development and Product Management Teams

The Spirit of the RUP

- Szybko uwzględniaj zmiany
  - Procedury wprowadzania zmian
  - Ocena konsekwencji danej zmiany
  - Minimalizacja kosztów zmiany
- Zacznij od architektury
  - Najpierw zbuduj szkielet, potem uzupełnij brakujące elementy
- Buduj z komponentów

Per Kroll, Director, RUP Development and Product Management Teams

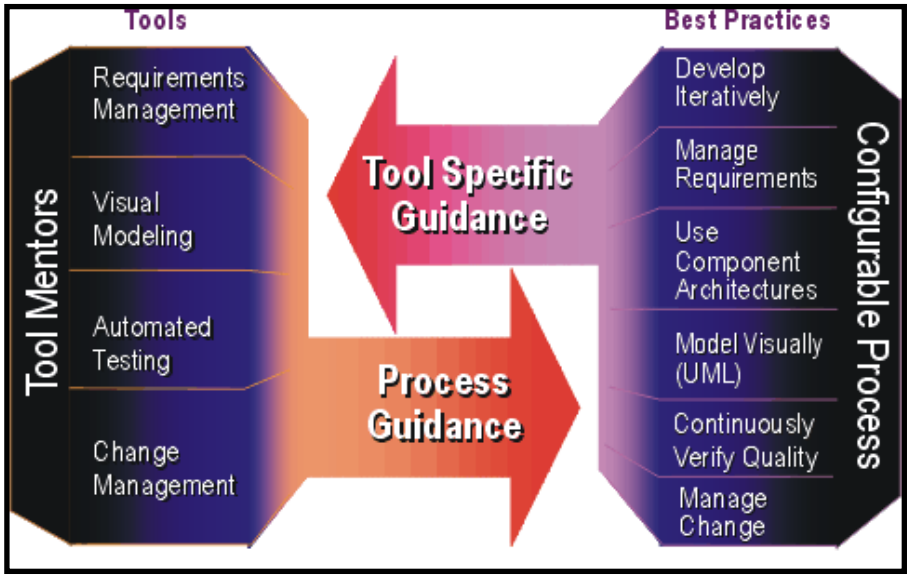
The Spirit of the RUP

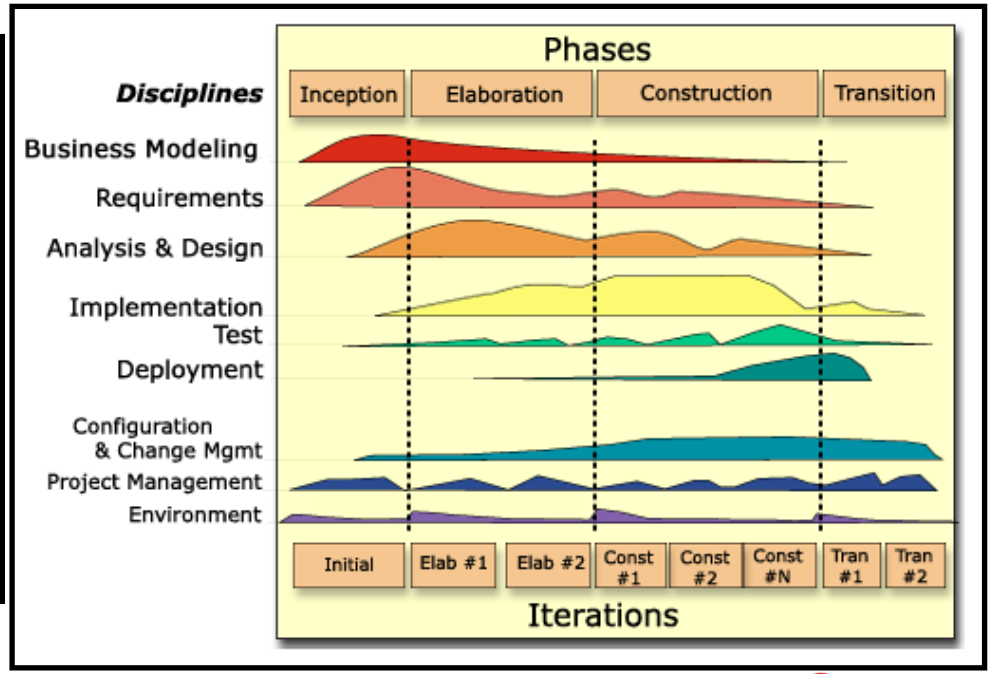
- Stwórz zespół
  - Nie buduj zespołów wokół pojedynczych funkcji
  - Rezultaty są współwłasnością członków zespołu
- Zapewnienie jakości determinuje realizację
  - Quality By Design
  - Testowanie od początku realizacji

Per Kroll, Director, RUP Development and Product Management Teams

Podstawa RUP

- Najlepsze praktyki (Best Practices)
- Etapy (Phases)
- Dyscypliny (Disciplines)





Co daje nam RUP?

- Pomaga w budowie wysokiej jakości oprogramowania
- Dostarczonego na czas i w budżecie
- Wymaga więcej niż poświęcenie jednostek
- Spójna praca zespołu i wspólna wizja zadań
  - Optymalizacja

Co daje nam RUP?

- Zapewnia przewidywalność i powtarzalność implementacji
- Pomaga skoncentrować się na działającym oprogramowaniu
- Pozwala na udźwignięcie oraz efektywne adaptowanie nowych technik i narzędzi

KONIEC