

Dr Katarzyna Grzesiak-Kopeć

Inżynieria oprogramowania



2. Najlepsze praktyki



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

6 najlepszych praktyk

1. Podejście iteracyjne
2. Zarządzanie wymaganiami
3. Architektura modułowa (komponenty)
4. Wizualizacja modelu
5. Ciągła weryfikacja jakości
6. Zarządzanie zmianami



6 najlepszych praktyk

1. Podejście iteracyjne
2. Zarządzanie wymaganiami
3. Architektura modułowa (komponenty)
4. Wizualizacja modelu
5. Ciągła weryfikacja jakości
6. Zarządzanie zmianami



Technika wodospadu

Analiza wymagań

Projektowanie

Kodowanie i testowanie modułów

Integracja podsystemów

Testowanie systemu

- Opóźnia identyfikację ryzyka
- Trudno ocenić rzeczywisty czas do ukończenia projektu
- Odsuwa w czasie integrację i agreguje testowanie
- Wyklucza wczesne wdrażanie
- Częste niezaplanowane iteracje

Technika iteracyjna



Zalety techniki iteracyjnej

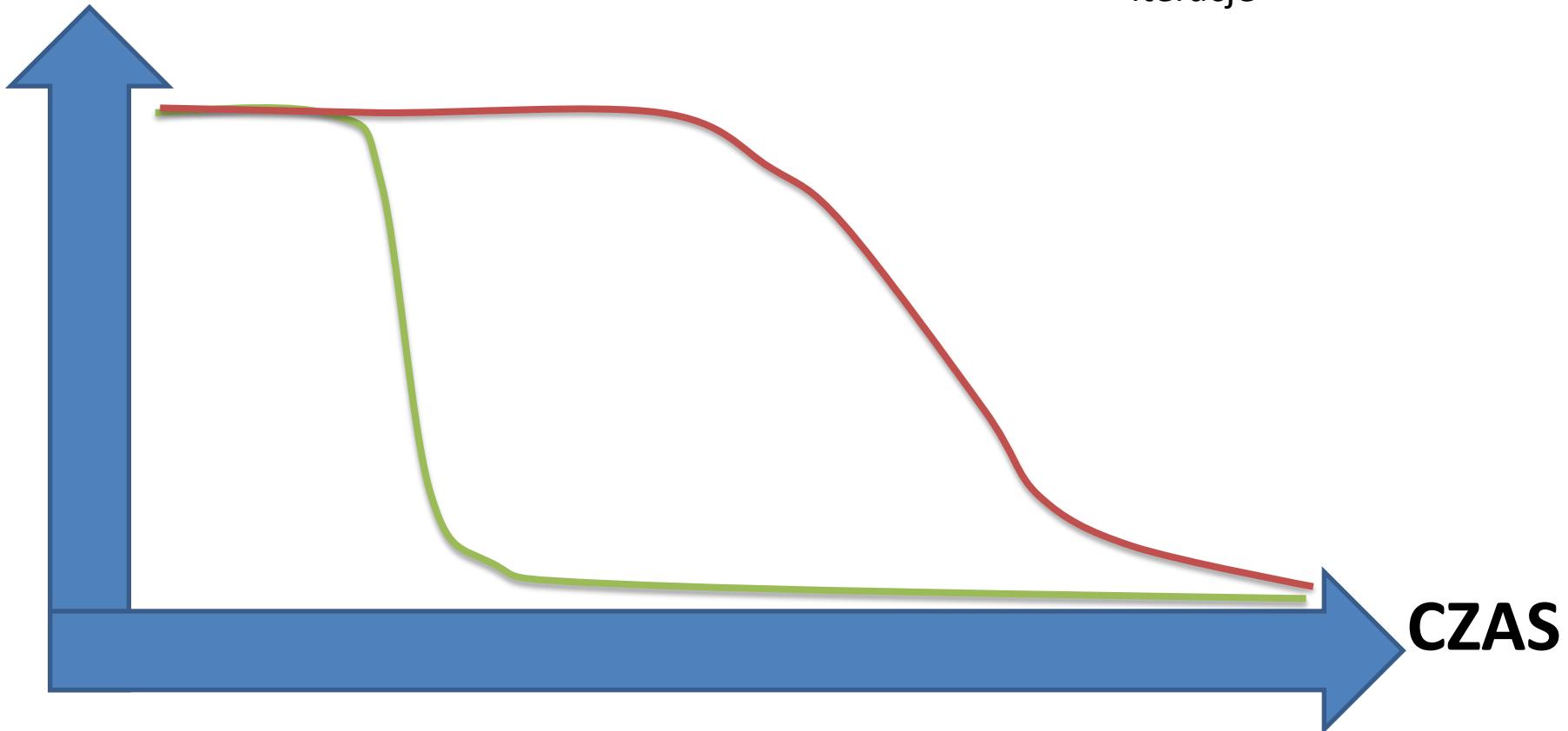
- Rozwiążanie ryzykownych kwestii zanim będą duże inwestycje
- Umożliwia wczesną ocenę przez klienta
- Ciągłe integrowanie i testowanie
- Realizacja zadania w małych, krótkoterminowych krokach
- Umożliwia częściowe wdrażanie produktu



Ryzyko

RYZYKO

wodospad
iteracje



6 najlepszych praktyk

1. Podejście iteracyjne
2. Zarządzanie wymaganiami
3. Architektura modułowa (komponenty)
4. Wizualizacja modelu
5. Ciągła weryfikacja jakości
6. Zarządzanie zmianami



Zarządzanie wymaganiami

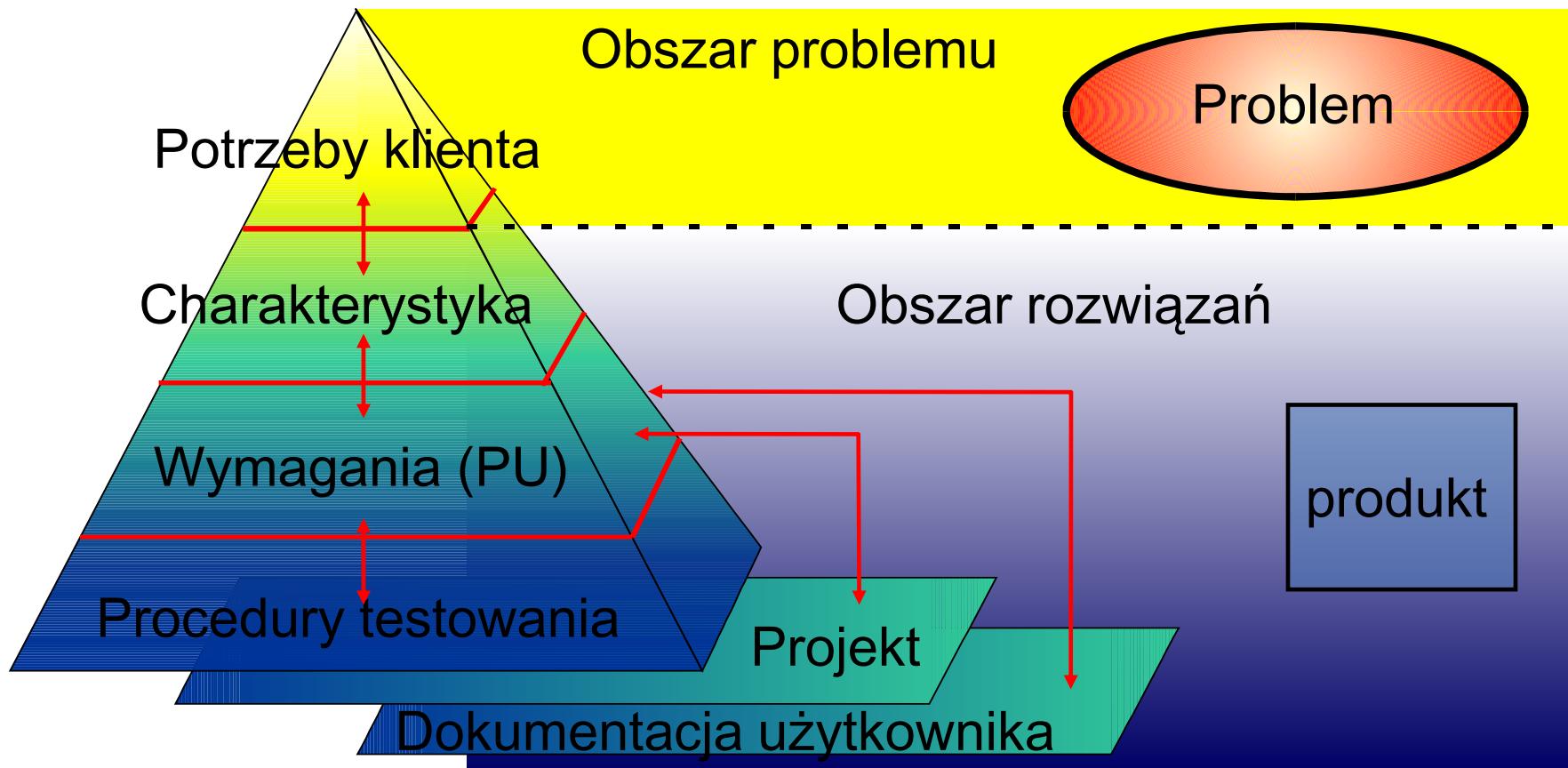
- Należy upewnić się, że:
 - Rozwiążujemy właściwy problem
 - Budujemy właściwy system
- Po przez systematyczne:
 - gromadzenie,
 - organizację,
 - dokumentowanie,
 - zarządzanie
- zmieniającymi się wymaganiami.

Wymagania – aspekty

- Analiza problemu
- Zrozumienie potrzeb klienta
- Zdefiniowanie systemu
- Ogarnięcie całości
- Ciągła weryfikacja definicji systemu
- Budowa właściwego systemu



Śledzenie wymagań



Korzyści ze śledzenia wymagań

- Ocena wpływu zmiany wymagania na nasz projekt
- Ocena stopnia realizacji wymagania na podstawie przygotowanych procedur
- Ogarnąć cały projekt
- Zweryfikować, czy wszystkie wymagania zostały zrealizowane
- Zweryfikować, czy aplikacja robi tylko to, co miała robić
- Zarządzać zmianami

6 najlepszych praktyk

1. Podejście iteracyjne
2. Zarządzanie wymaganiami
3. Architektura modułowa (komponenty)
4. Wizualizacja modelu
5. Ciągła weryfikacja jakości
6. Zarządzanie zmianami



Co to jest ARCHITEKTURA?

**ZBIÓR ZASAD, DECYZJI, REGUŁ
LUB WZORCÓW DLA
DOWOLNEGO SYSTEMU, KTÓRY
ZAPOBIEGA NIEPOTRZEBNEJ
RADOSNEJ TWÓRCZOŚCI
PROJEKTANTÓW.**



Co to jest ARCHITEKTURA?

- Architektura oprogramowania określa najważniejsze elementy organizacji systemu:
 - Wybór elementów strukturalnych oraz ich interfejsów
 - Określenie zachowania systemu jako interakcji pomiędzy tymi elementami
 - Kompozycja elementów strukturalnych i behawioralnych w podsystemy



Co to jest ARCHITEKTURA?

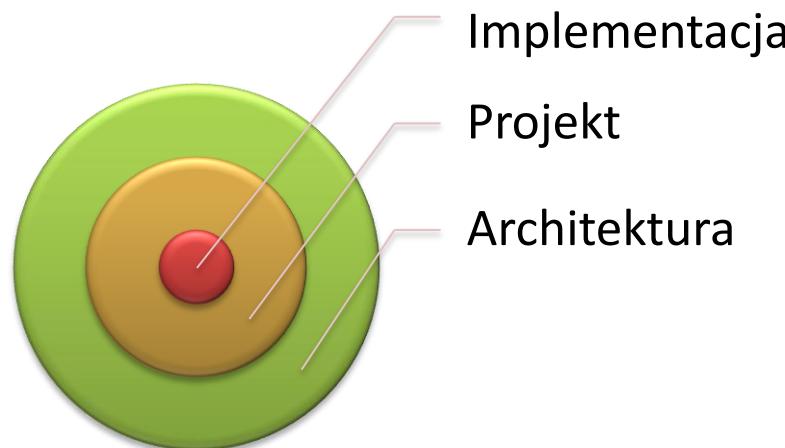
- Ogólna budowa pewnego rozwiązania informatycznego, którego elementem jest zarówno oprogramowanie, jak i pewna infrastruktura techniczna, sprzęt
- Łączy właściwości strukturalne (komponenty fizyczne) oraz funkcjonalne (zewnętrzne i wewnętrzne funkcje systemu)
- Opisuje składowe, powiązania między nimi oraz sposoby i kierunki przepływu danych

Jaka ARCHITEKTURA?

- Architektura logiczna
 - Podział na logiczne fragmenty (moduły, pakiety, komponenty), które powinny współpracować w celu realizacji wymagań
 - UML – komponent + interfejsy
- Architektura fizyczna
 - Określenie fizycznych jednostek (maszyn) oraz metod komunikacji
 - UML – węzeł

Architektura to fundament

- Określenie architektury, to podjęcie strategicznych decyzji, określenie reguł i wzorców determinujących projektowanie i konstrukcję – **FUNDAMENTALNE DECYZJE**



Dobra architektura

- Elastyczna
 - Spełnia bieżące i przyszłe wymagania
 - Umożliwia rozbudowę
 - Umożliwia ponowne użycie
 - Enkapsuluje zależne części systemu
- Modułowa
 - Ponowne użycie i modyfikacja modułów
 - Wybór z komercyjnie dostępnych komponentów
 - Inkrementacyjny rozwój oprogramowania



Moduł - komponent

**Nietrywialna, prawie niezależna i wymienna
część systemu, która realizuje zdefiniowaną
funkcjonalność.**



Po co nam moduły?

- Umożliwia ponowne wykorzystanie
 - Komponentów
 - Architektury
- Podstawa do zarządzania projektem
 - Planowanie
 - Ludzie
 - Dostarczanie produktu
- Kontrola
 - Zarządzanie złożonym systemem
 - Integracja



6 najlepszych praktyk

1. Podejście iteracyjne
2. Zarządzanie wymaganiami
3. Architektura modułowa (komponenty)
4. Wizualizacja modelu
5. Ciągła weryfikacja jakości
6. Zarządzanie zmianami



Po co nam wizualizacja?

- Uchwycenie struktury i zachowania
- Pokazanie zależności między elementami
- Utrzymanie spójnego projektu i implementacji
- Ukrywanie oraz wyróżnianie szczegółów kiedy trzeba
- Promuje jeden wspólny język wymiany informacji
 - UML: jeden język dla wszystkich osób związanych z tworzeniem projektu



The Unified Modeling Language

- Język do specyfikowania, wizualizowania, konstruowania i dokumentowania tworów oprogramowania i modelowania biznesu
- Reprezentuje zbiór najlepszych praktyk inżynierskich które dowiodły przydatności przy modelowaniu dużych i skomplikowanych systemów



Historia

- Do połowy lat 90 – około 50 metod
- Potrzeba unifikacji
- '94 Booch, Jacobson i Rumbaugh z Rational Software
- '95 Unifikacja OMT, Boocha i OOSE w UML
- '96 standaryzacja UML pod skrzydłami OMG (<http://www.omg.org>) [wersje 1.1,..,1.5]
- 2005 OMG publikuje UML 2.0
- Maj 2010 UML 2.3
- 2012 UML 2.5



Rynek i UML

- Object Management Group (OMG)
- Hewlett-Packard Company
- IBM Corporation (Rational Software Corporation)
- Microsoft Corporation
- Oracle Corporation
- i inne mniej znane...



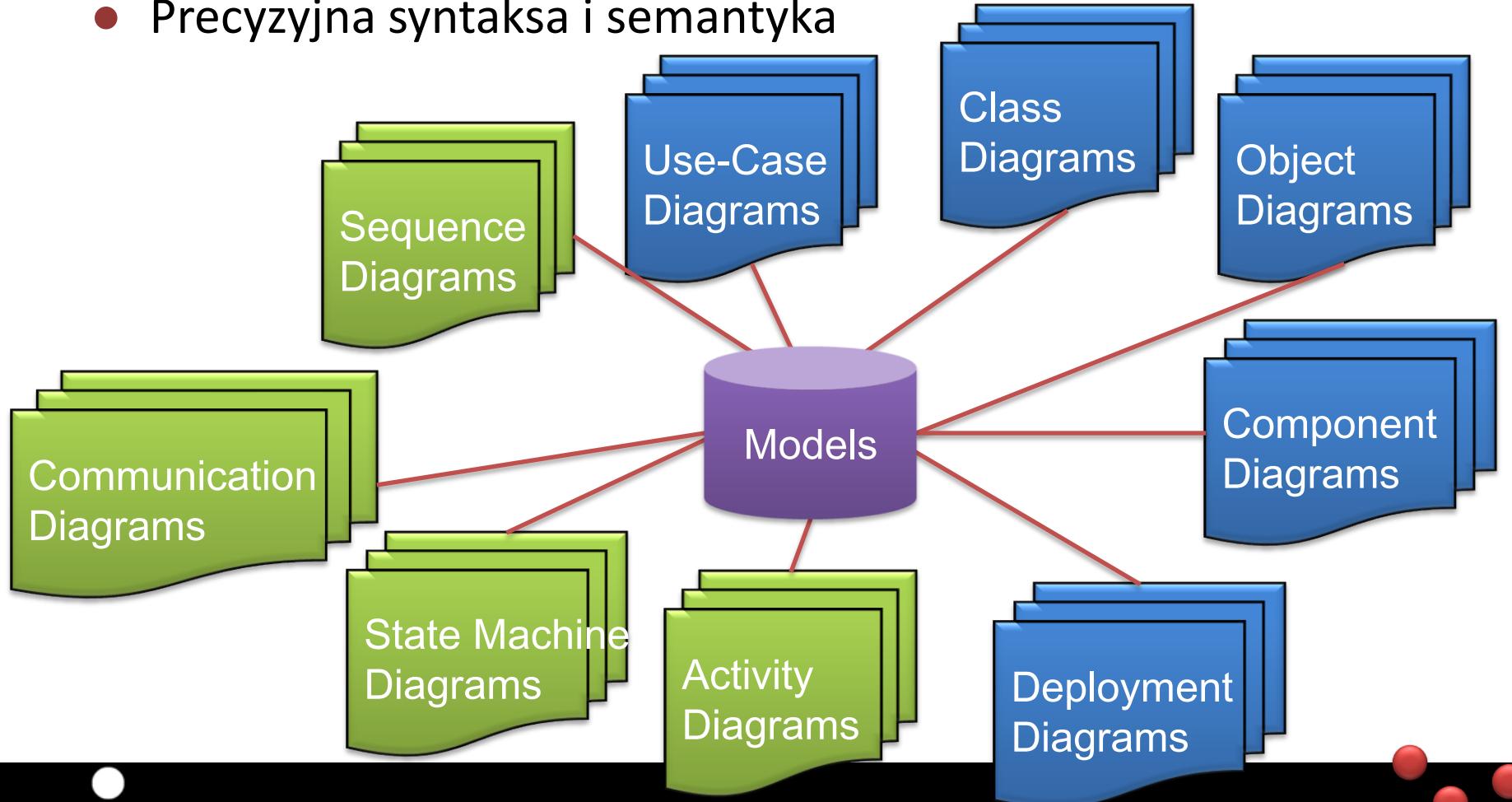
- IBM Rational Rose/IBM Rational Software Architect
- Microsoft Visio
- Poseidon for UML
- Sybase PowerDesigner
- Visual Paradigm
- Umbrello
- Eclipse
- NetBeans
- I wiele innych freeware, shareware i komercyjnych

Dlaczego UML?

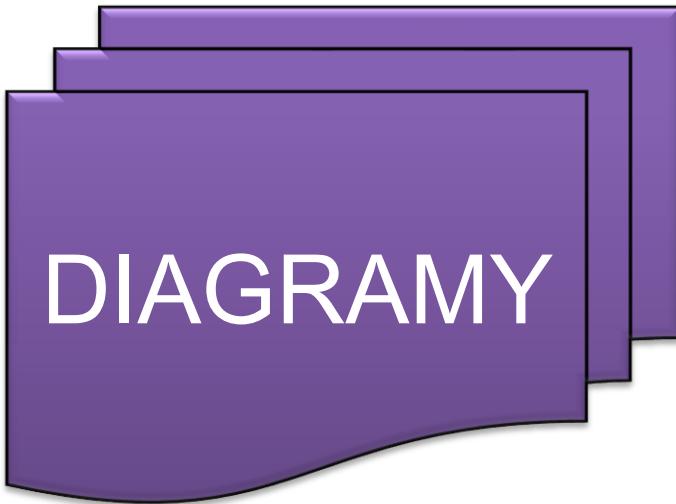
- Gotowy wizualny język modelowania do budowy i wymiany modeli
- Niezależny od języków programowania i procesów tworzenia
- Udostępnia podstawy formalne do zrozumienia języka modelowania
- Zachęca do wzrostu rynku narzędzi OO
- Wspomaga wysoko-poziomowe rozwiązania
- Integruje najlepsze praktyki

Wizualizacja i UML

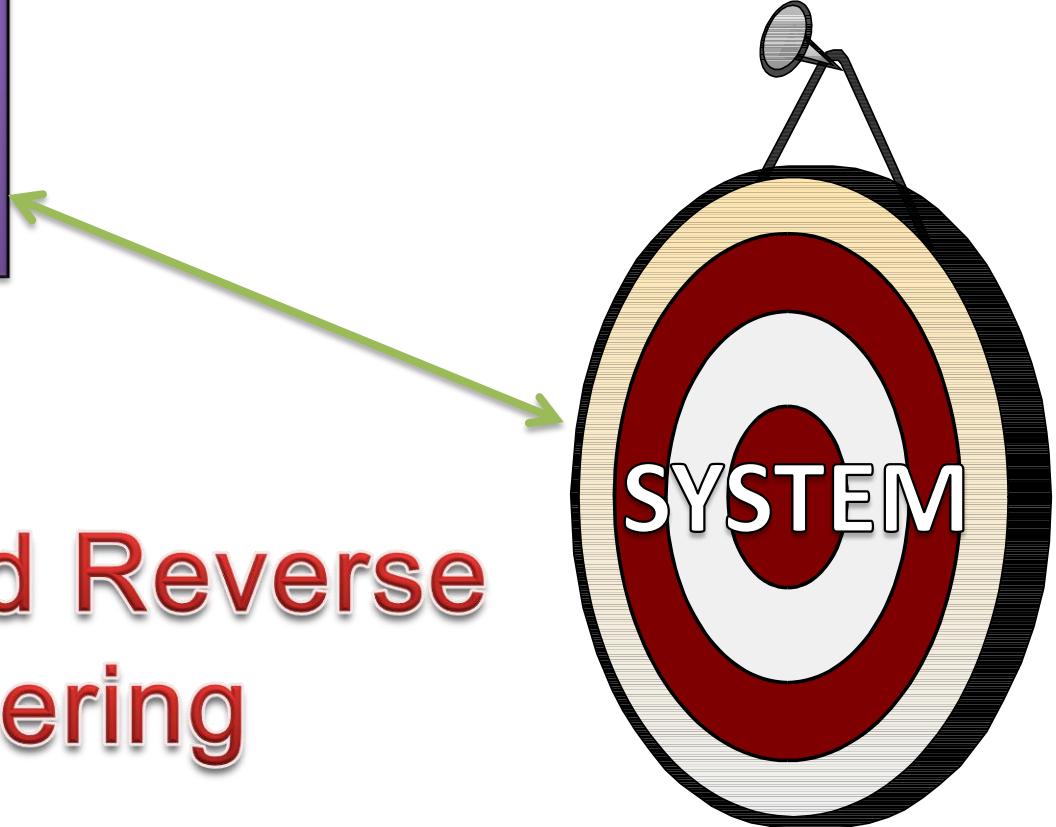
- Wiele punktów widzenia
- Precyzyjna syntaksa i semantyka



Wizualizacja i UML



**Forward and Reverse
Engineering**



Krytyka UMLa



6 najlepszych praktyk

1. Podejście iteracyjne
2. Zarządzanie wymaganiami
3. Architektura modułowa (komponenty)
4. Wizualizacja modelu
5. Ciągła weryfikacja jakości
6. Zarządzanie zmianami



Jakość

Charakterystyka wyniku produkcji oprogramowania określona na podstawie tego, czy produkt spełnia lub przewyższa uzgodnione wymagania oraz czy zostało ono wytworzzone zgodnie z ustalonym procesem, co jest mierzone zgodnie z ustalonymi kryteriami .



Jakość

- Widoczna
- Ukryta
- Testy to za mało
- Oceny użytkownika
 - Inspekcje modeli
 - Inspekcje kodu
- Kontrola jakości w każdej iteracji!

Weryfikacja jakości

- Wady oprogramowania są od 100 do 1000 razy droższe do zidentyfikowania i do naprawienia po wdrożeniu
 - Koszt naprawy błędów
 - Koszt straconych możliwości
 - Koszt straconych klientów



Kompletne testowanie

Czy aplikacja robi to co trzeba?

Funkcjonalność

✓ Weryfikacja scenariuszy użycia

Czy osiągnięto zadawalającą niezawodność?

Niezawodność

✓ Weryfikacja najdłużej trwających operacji

Wydajność

✓ Weryfikacja działania przy oczekiwany i najgorszym obciążeniu

Czy system działa przy rzeczywistym obciążeniu?

6 najlepszych praktyk

1. Podejście iteracyjne
2. Zarządzanie wymaganiami
3. Architektura modułowa (komponenty)
4. Wizualizacja modelu
5. Ciągła weryfikacja jakości
6. Zarządzanie zmianami



Co chcemy kontrolować?

- Zmiany
- Podział pracy w zespole
- Integrację
- Równoległe prace



Istotne aspekty

- Zarządzanie zgłoszonymi zmianami
 - Formalny proces zgłaszania zmian i oceny ich wpływu na projekt
 - Blokujemy „przeciekanie” nowych wymagań
- Raportowanie stanu konfiguracji
 - Mierzenie rodzaju i liczby błędów
- Zarządzanie konfiguracją
 - Narzędzia do wersjonowania i pracy grupowej

Istotne aspekty

- Śledzenie zmian
 - Formalny proces realizacji zmian
- Wybór wersji i produkowanie systemu
 - Automatyzacja kompilacji, testowania, pakietowania i rozprowadzania



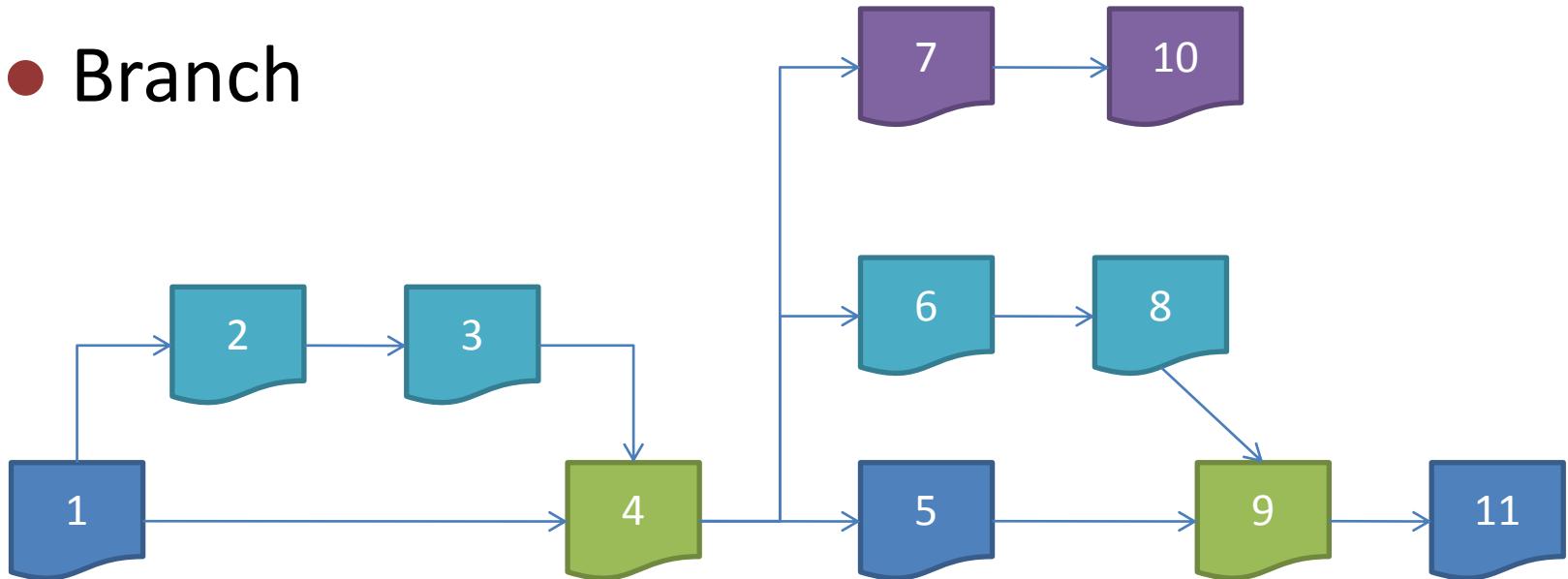


System kontroli wersji

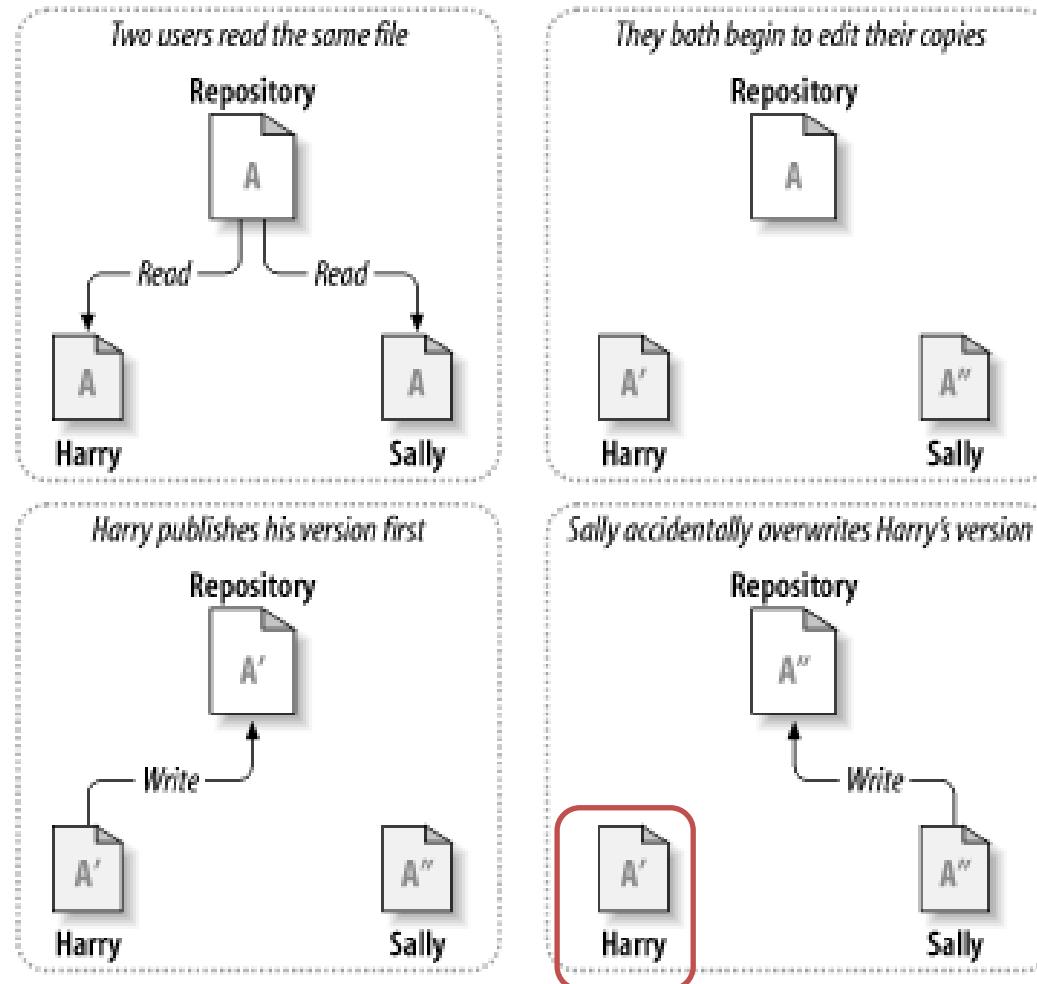
- Zarządzanie konfiguracją
- Zarządzanie katalogami, plikami i zmianami
- Architektura
 - Scentralizowana klient-serwer
 - Rozproszona P2P
- Organizuje pracę w zespole

Pojęcia

- Commit (checkin)
- Checkout
- Trunk (baseline, mainline)
- Branch



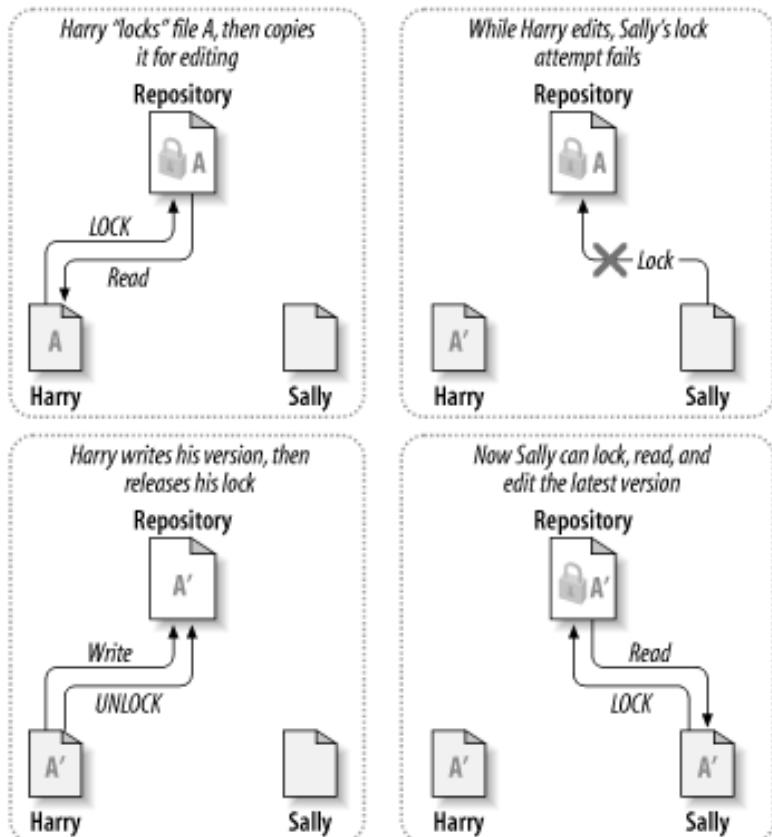
Problem współdzielenia



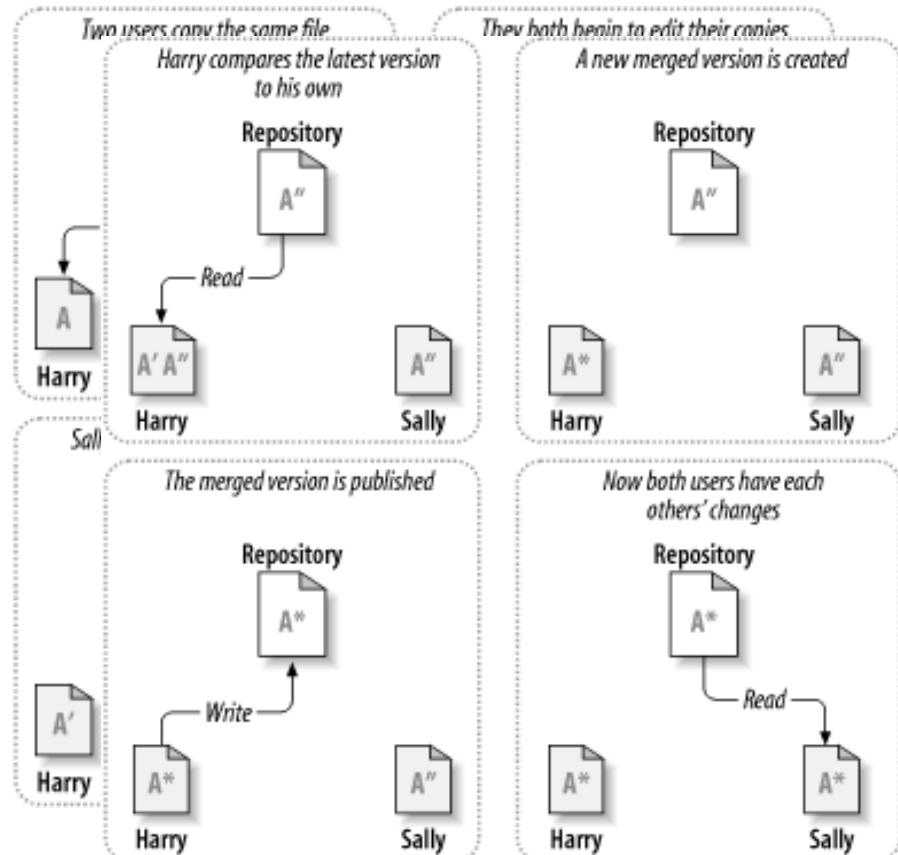
Problem współdzielenia



Blokuj-Modyfikuj-Odblokuj



Kopiuj-Modyfikuj-Scal



6 najlepszych praktyk

1. Podejście iteracyjne
2. Zarządzanie wymaganiami
3. Architektura modułowa (komponenty)
4. Wizualizacja modelu
5. Ciągła weryfikacja jakości
6. Zarządzanie zmianami
7. ?



Nie odkrywaj koła!

- Poziomy ponownego użycia
 - Klasy
 - Wzorce projektowe
 - Komponenty
 - Interfejsy i wzorce analityczne
 - Wzorce architektury
 - Wzorce wymagań
- Wyższy poziom ⇒ trudniej wykorzystać
- Wyższy poziom ⇒ większe korzyści



KONIEC

