Wykład 10-11

1. Zarządzanie konfiguracją i zmianami
2. Wdrożenie
3. Konserwacja oprogramowania
4. Metodyki wytwarzania – podejście tradycyjne i zwinne (XP)

Zarządzanie konfiguracją - podstawowe pojęcia

Konfiguracja – zmienny w czasie zestaw ustalonych artefaktów projektu i innych informacji, które są istotne do sprawnej jego realizacji.

Jej elementy to:

- Dokumentacja produktu programowego
- Dokumentacja projektowa
- Standardy, procedury, instrukcje
- Kod programu

Linia bazowa konfiguracji

Zestaw artefaktów, który został poddany przeglądowi i zatwierdzony.

Jest podstawą do dalszego rozwoju.

Może być zmieniony tylko poprzez formalne procedury.

IEEE Std. 610.12-1990

Konfiguracja – podstawowe pojęcia

Jednostka (element) konfiguracji *(ang. SCI)*
- artefakt powstały w trakcie procesu wytwarzania oprogramowania

[R. Pressman]

Identyfikowanie jednostki konfiguracji poprzez:
- **Nazwa**
- **Opis** (typ SCI, identyf. projektu, wersja, informacja o zmianach)
- **Zasoby** wymagane przez SCI (np. zmienne globalne wymagane do kompilacji)
- **Realizacja** (np. odwołanie do pliku tekstowego z kodem)

Typy elementów konfiguracji:
- **oprogramowanie** (kod źródłowy lub binarny)
- **dokumenty** (w tym podręczniki)
- **dane** (np. przypadki testowe)

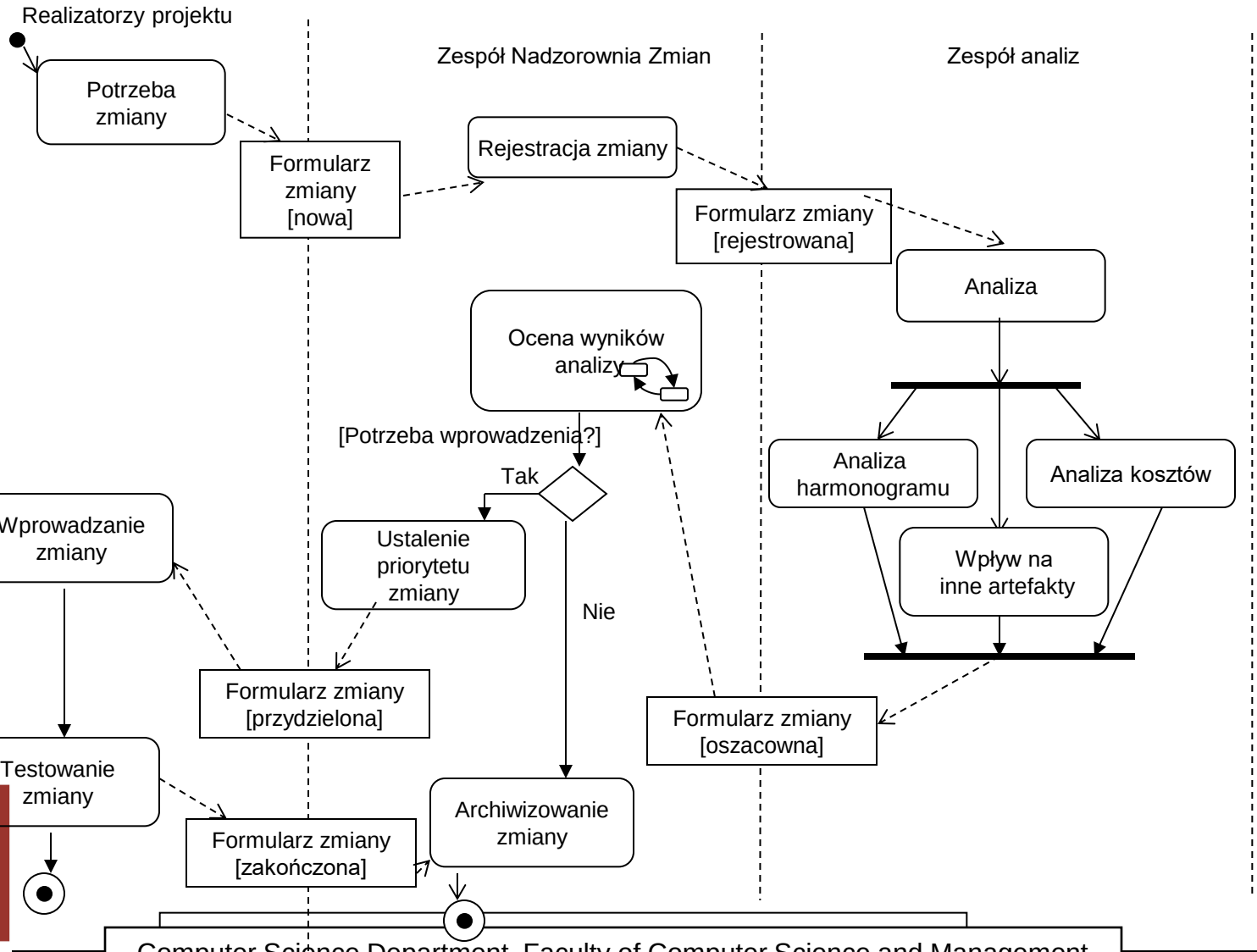
Zarządzanie zmianami

Zarządzanie zmianami (ZZ) obejmuje proces **modyfikacji elementów linii bazowej** w jednolity, spójny sposób.

Żądanie zmian może dotyczyć różnorodnych elementów np. zmian wymagań, dokumentowania defektów, zmiany czasu przejścia między iteracjami.

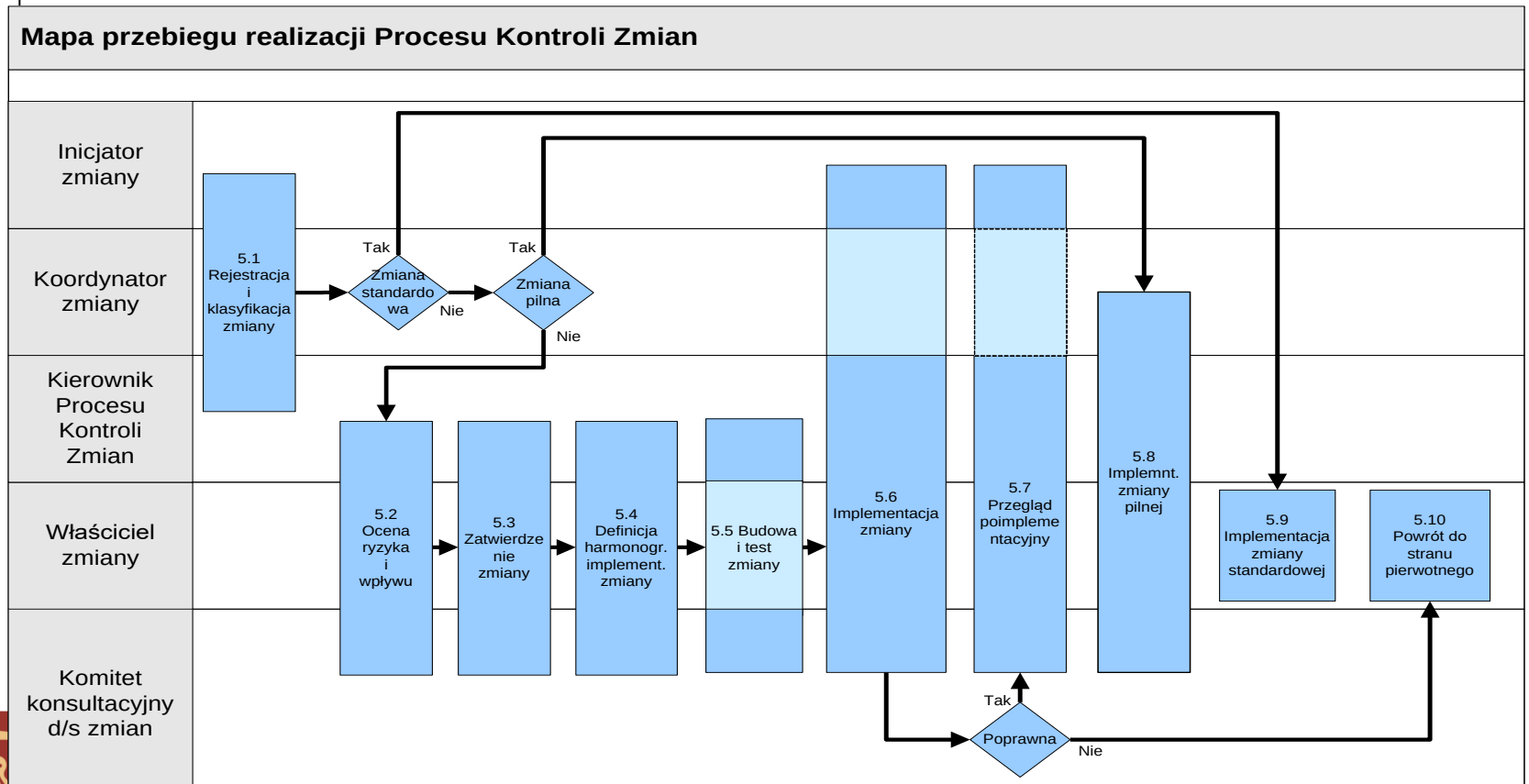
ZZ dotyczy struktury procesu.

Zarządzanie zmianami - proces



Zarządzanie zmianami – przykład procesu

(BRE BANK SA.)



Zarządzanie wersjami

Dotyczy: elementu linii bazowej konfiguracji

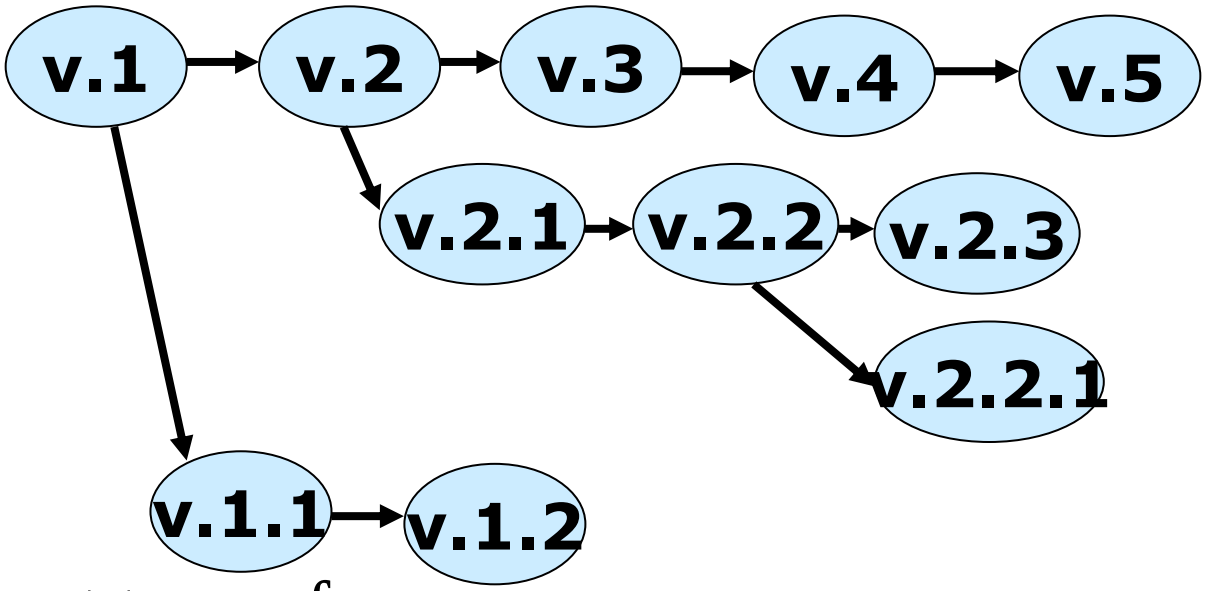
Cele:

- Kontrolowanie zmian
- Zarządzanie bazą wersji
- Kontrolowanie i przechowywanie historii zmian

Wersja – instancja artefaktu, która **różni się** od innych instancji **oferowanymi** funkcjami

Wydanie – *wersja* dostarczona użytkownikowi

Zarządzanie wersjami - oznaczenia



Schemat trzycyfrowy

<wersja> = <nazwa_SCI>.<major>.<minor>.<revision>

<major> - Wydanie (dla klienta)

<minor> - Wersja w ramach wydania (dla programisty)

<revision> - Poprawki w ramach wersji wydania (dla programisty)

Zarządzanie wersjami - narzędzia

- **CVS** (Concurrent Versions System)
- **SubVersion** następca CVS; najbardziej popularny

- **GIT**
- **Mercurial**

Narzędzia wersjonowania – pojęcia podstawowe

- *Repository* (repozytorium) - miejsce, gdzie przechowywane są pliki
- *Check-Out* - pobranie pliku z repozytorium
- *Update* (aktualizacja) - kopiuje zmiany dokonane w repozytorium do katalogu, w którym pracujemy
- *Change* (zmiana) – reprezentuje modyfikację do dokumentu znajdującego się pod kontrolą wersji.
- *Commit* (również: install, submit, check-in) - wgranie zmian do repozytorium
- *Change List* (lista zmian) – zespół zmian dokonanych podczas jednej operacji *commit*
- *Merge* (łączenie) – łączy konkurencyjne zmiany do jednej zunifikowanej wersji
- *Conflict* (konflikt) – pojawia się, kiedy algorytm łączenia nie jest w stanie wygenerować poprawnej wersji zawierającej zmiany wprowadzone do łączonych plików
- *Resolve* (rozwiązanie konfliktu) – akt interwencji użytkownika w celu wsparcia algorytmu do nanoszenia różnych zmian tego samego do kumentu

Funkcje narzędzi wersjonowania

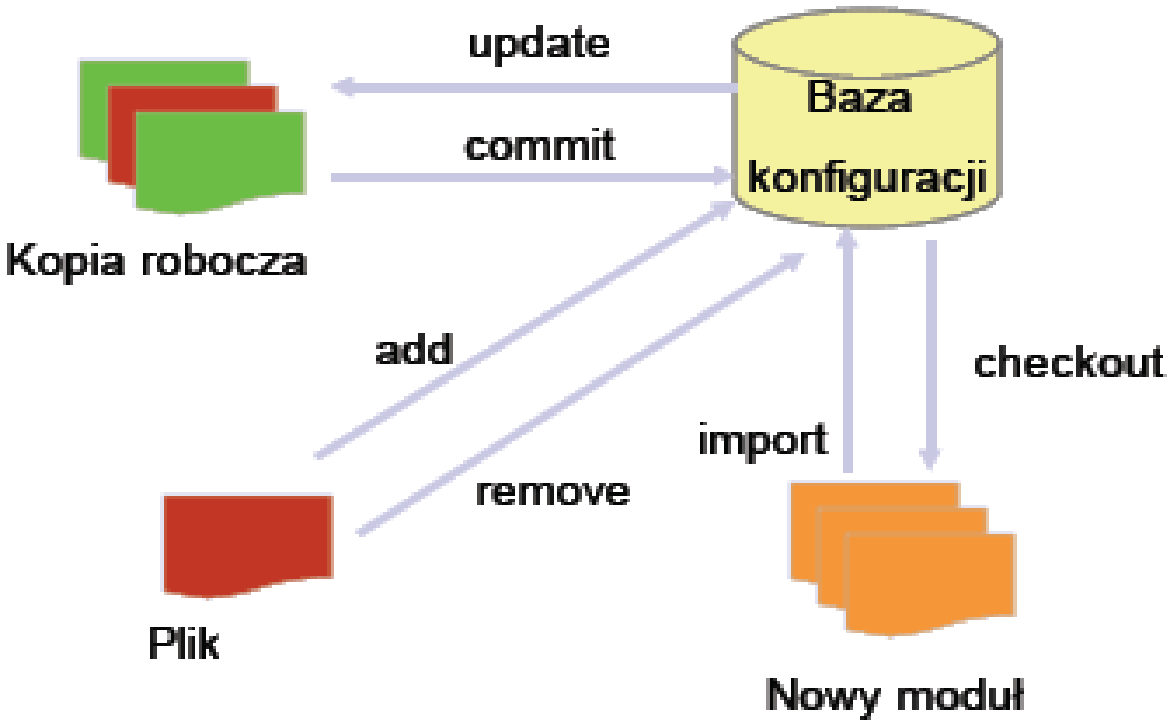
- Zarządzanie repozytorium komponentów
  - Zarządzanie wersjami
    - Historia
    - Przechowywanie delt
    - Zarządzanie wielodostępem
  - Zarządzanie relacjami między obiektami
    - Zawieranie
    - Zależność

- Wsparcie inżynieryjne
  - Kompilowanie i budowa projektu
  - Wsparcie środowiska pracy
  - Wsparcie pracy kooperacyjnej
- Wsparcie procesu projektowania

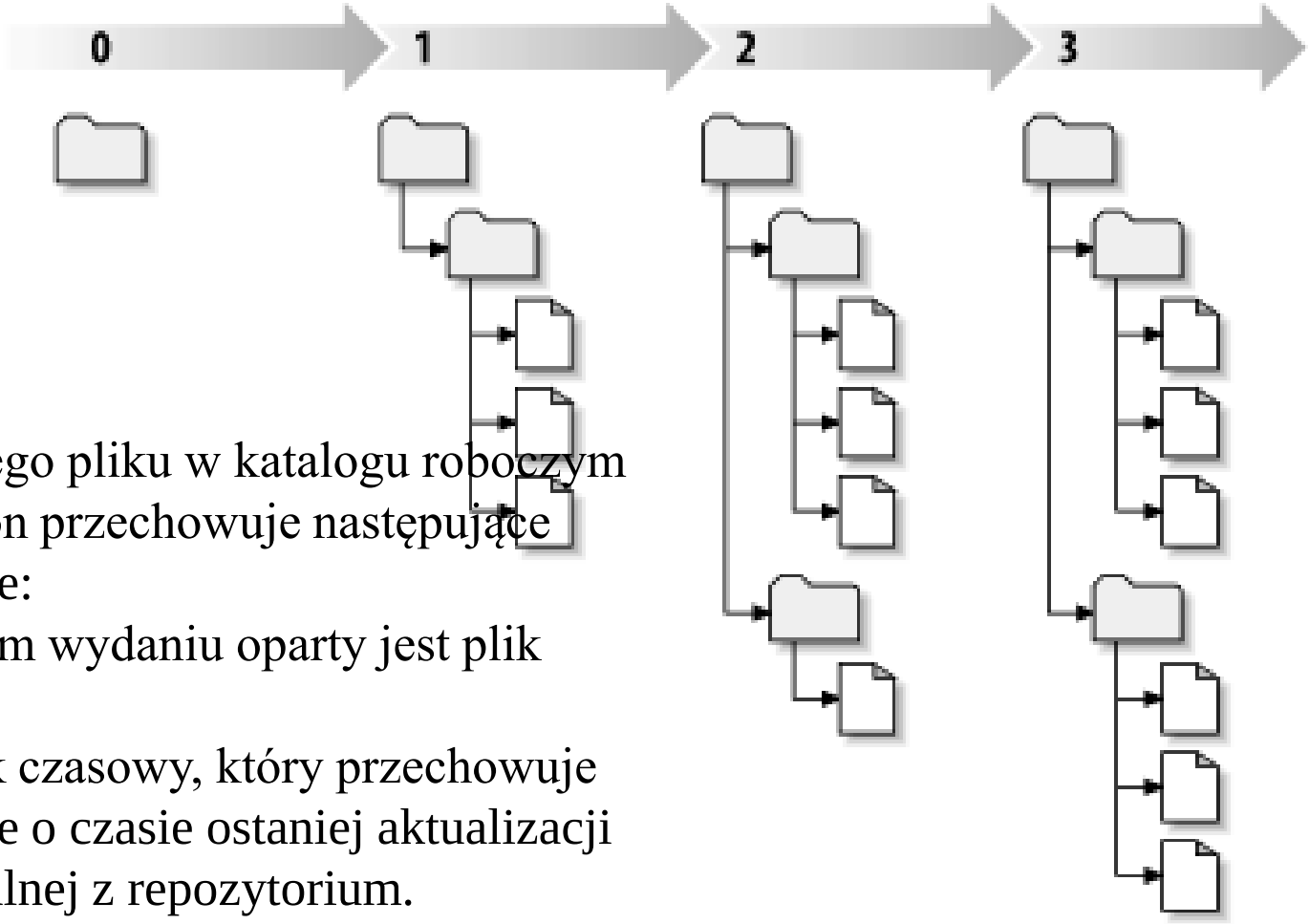
Funkcje narzędzi wersjonowania *(cd)*

- Cofanie dowolnej liczby zmian
- Dokonywanie przeglądów – sprawdzanie kompletności i spójności komponentów
- Ustalenie odpowiedzialności pracowników – kto dopisał tę linię kodu, która powoduje błąd?
- Raportowanie:
  - historii zmian
  - różnic między wersjami
  - aktywności pracowników
  - statusu całego projektu i jego komponentów
- Oszczędność zasobów (czasu, pracy…), bezpieczeństwo

Zarządzanie wersjami – podstawowe operacje w CVS



Przykład struktury repozytorium (Subversion)



Dla każdego pliku w katalogu roboczym Subversion przechowuje następujące informacje:

- Na którym wydaniu oparty jest plik roboczy,
- Znacznik czasowy, który przechowuje informacje o czasie ostaniej aktualizacji kopii lokalnej z repozytorium.

Zarządzanie konfiguracją – błędy

- nie ma pewności, która wersja elementu jest wersją bieżącą (strata czasu, dodatkowy nakład pracy)
- niezgodności między poszczególnymi elementami konfiguracji (np. specyfikacja programu, kod i specyfikacja testów są niezgodne)
- „wydawanie” produktów nie jest kontrolowane
- niekontrolowane zmiany w środowisku oprogramowania lub sprzętu –bez analizy ich skutków
- w przypadku „katastrofy” - nie do odtworzenia bieżący stan przedsięwzięcia

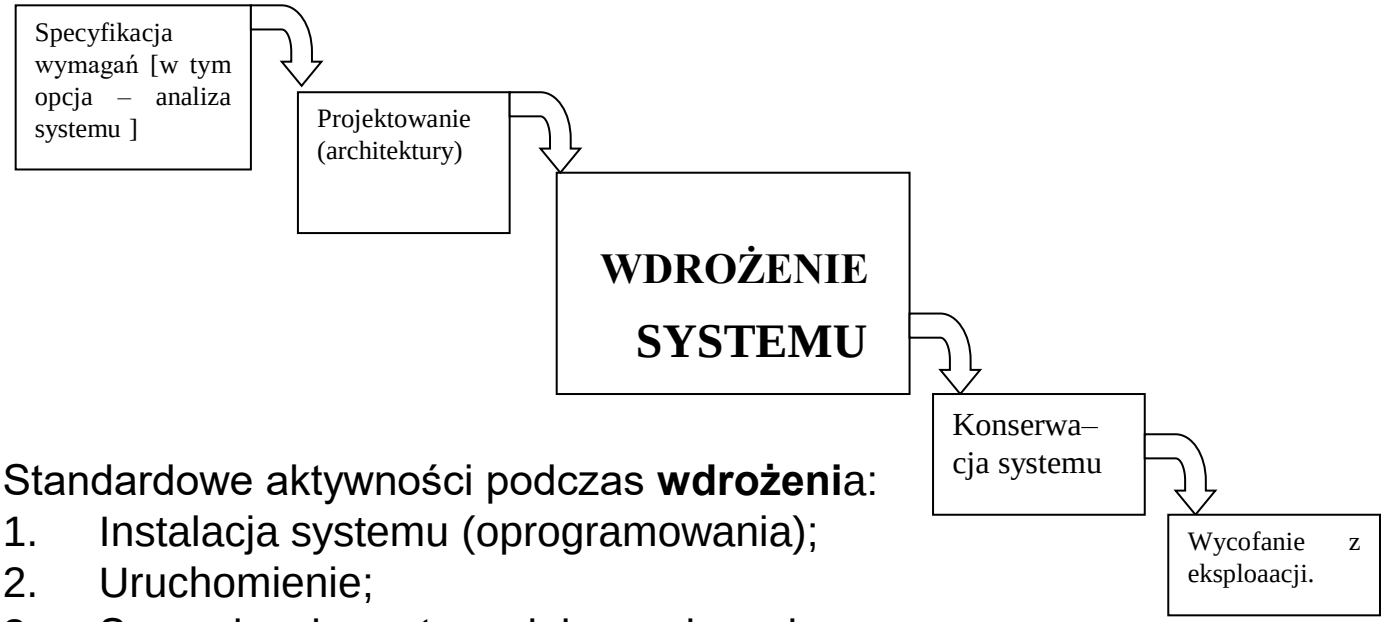
Wdrożenie systemu informatycznego

- **wdrożenie jest procesem samooceny, analizy, poważnej zmiany procesów biznesowych i uczenia się',**

- wymaga ono zwykle istotnych zmian w strukturze i organizacji pracy całej firmy np. przy wprowadzaniu systemów informatycznych klasy ERP *(ang. Enterprise Resource Planning)*

- lepiej mówić o **informatyzacji firmy** rozumiejąc ją jako projekt informatyczny, której jednym z elementów jest wdrożenie systemu informatycznego.

Wdrożenie systemu informatycznego



Standardowe aktywności podczas **wdrożeni**a:

1. Instalacja systemu (oprogramowania);
2. Uruchomienie;
3. Sprawdzenie systemu lub rozwiązania;
4. Przekazanie do eksploatacji.

Sukces wdrożenia:
Czy zmieszczono się w harmonogramie?
Czy zmieszczono się w budżecie?
Czy zrealizowana całość projektu?
Jakie są parametry eksploatacyjne?
Czy wdrożony system jest zgodny z ustalonymi wcześniej celami?
Czy wdrożenie przyniosło MIERZALNE EFEKTY?

Wdrożenie systemu informatycznego -***strategie***

krokowy (*ang. step-by step*) - konwencjonalne; wdrażanie sekwencyjne kolejnych podsystemów wg ustalonych priorytetów; wadami takiego podejścia są długi okres trwania i poczucie tymczasowości rozwiązań, natomiast zalety – to rozłożenie kosztów w czasie i nieangażowanie wielu pracowników we wdrożenie; w Polsce blisko 60-70% , w świecie około 25-30% systemów jest wdrażanych w ten sposób;

kompleksowy (*ang. big bang*) – wdrażanie całego systemu od razu; wadami takiego podejścia są angażowanie wielu pracowników i dyrekcji we wdrożenie oraz poniesienie dużych kosztów w krótkim czasie, natomiast zalety – to krótki okres trwania wdrożenia i szybko widoczne efekty we wszystkich działach firmy; w Polsce blisko zera a w świecie około 25-30% systemów jest wdrażanych w ten sposób;

główne podsystemy/moduły jednocześnie (*ang. middle-size big bang*) wdrażanie jednoczesne tych modułów, którym przyznano najwyższe priorytety; zaletą takiego wdrożenia jest stosunkowo szybkie uzyskanie efektów w krótkim czasie w najważniejszych obszarach funkcjonowania firmy; w Polsce około 30-35% a w świecie około 50% systemów jest wdrażanych w ten sposób.

Wdrożenie systemu informatycznego - rodzaje

W ramach metod przeprowadzania konwersji przyjmuje się cztery podstawowe schematy działania:

1. **konwersja bezpośrednia** — polega na natychmiastowym wprowadzeniu systemu w miejsce dotychczas użytkowanego, niesie to największe ryzyko spośród wszystkich metod, gdyż w przypadku błędnego działania systemu, użytkownicy nie mają alternatywy,
2. **konwersja równoległa** — dotychczasowy i nowy system funkcjonują równolegle, aż do momentu osiągnięcia stanu pełnej niezawodności i stabilności nowego systemu; procedura jest bezpieczna, ale kosztowna i uciążliwa — stosowanie ma sens w krótkim okresie,
3. **konwersja pilotowa** — tylko część użytkowników wykorzystuje nowy system, testując jego działanie przed wprowadzeniem do ogólnego stosowania; metoda jest skuteczna i tania, ale długotrwała,
4. **konwersja fazowa** — polega na etapowym wprowadzaniu nowego systemu poprzez sukcesywne instalowanie poszczególnych modułów, zastępujących dotychczas użytkowane; głównym problemem tej metody jest czasochłonność i integracja danych między modułami różnych systemów — dotychczasowego nowego.

Konserwacja oprogramowania

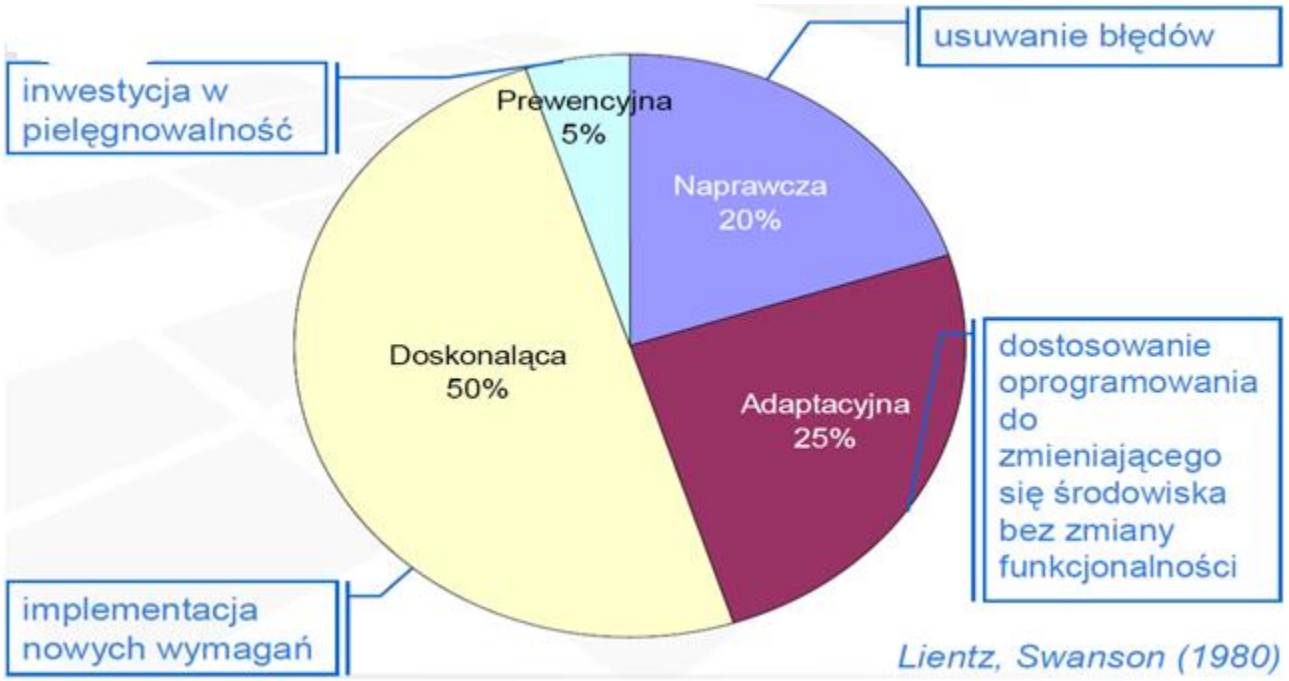
Typy konserwacji/pielęgnacji:

- korekcyjna — usuwanie błędów niewykrytych podczas testowania
- adaptacyjna — modyfikacja systemu dopasowująca go do zmian w środowisku
- perfekcyjna — rozszerzenie systemu do rozwiązywania nowych problemów lub uzyskania korzyści z nowych okoliczności
- prewencyjna — zabezpieczenie systemu przed przyszłymi problemami

Konserwacja oprogramowania obejmuje 4 aktywności:

- otrzymanie wymagań konserwacyjnych
- transformacje wymagań w zmiany
- zaprojektowanie zmian
- zaimplementowanie zmian

Konserwacja oprogramowania - koszty



- Koszt pielęgnacji zwykle jest **wyższy** od kosztu stworzenia oprogramowania
  - koszt linii kodu w Boeingu: $30, pielęgnacji: $4000 (Boehm, 1985)
- Koszt pielęgnacji **rośnie** wraz z okresem pielęgnacji

[*B.Walter, Uczelniaonline*]

Konserwacja oprogramowania

Powód prowadzenia modyfikacji (przykłady)

1. adaptacyjnej (dostosowującej):

- zmiany wymagań użytkowników
- zmiany przepisów prawnych,
- zmiany organizacyjne w firmie klienta
- zmiany sprzętu/oprogramowania systemowego

2. perfekcyjnej (ulepszającej):

- poprawa wydajności pewnych funkcji
- poprawa ergonomii interfejsu
- poprawa przejrzystości raportów

Przyczynami prowadzenia modyfikacji są na ogół żądania użytkowników

Konserwacja oprogramowania

Elementy kosztów konserwacji/pielęgnacji:

- defekty systemu — liczba nieznanych defektów po zainstalowaniu
- klienci konserwantów — liczba różnych klientów, których grupa musi obsłużyć
- dokumentacja aktualna, kompletna, dla różnych wersji systemu
- personel — zabezpieczenie systemu przed przyszłymi problemami
- narzędzia — niezawodne, dobrze rozpoznane, nowoczesne
- struktura oprogramowania — rozszerzenie systemu do rozwiązywania nowych problemów lub uzyskania korzyści z nowych sytuacji

- KOSZTY KONSERWACJI MOGĄ BYĆ DUŻE (mogą przewyższać koszty wytwarzania oprogramowania)

Konserwacja oprogramowania

Obiektywne czynniki kosztów konserwacji/pielęgnacji:

- stabilność środowiska pracy systemu
- stabilność platformy sprzętowej i oprogramowania systemowego
- czas użytkowania systemu

Redukcja kosztów dzięki:

- znajomości dziedziny problemu
- wysokiej jakości modelu i projektu
- wysokiej jakości dokumentacji technicznej (możliwość stosowania inżynierii odwrotnej)
- stabilność personelu
- środowisko implementacji
- niezawodność oprogramowania
- zarządzanie wersjami

Konserwacja oprogramowania – kategorie programów Lehmana

Program typu E (osadzony w rzEczywistości)

- **założenia**: system funkcjonuje w rzeczywistym świecie
- **kryterium jakości**: subiektywna ocena użytkownika
- **ewolucja**: nieunikniona, program i jego środowisko nieustannie oddziałują na siebie

Program typu P (rozwiązujący Problem)

- **założenia**: system w przybliżeniu odtwarza rzeczywistość
- **kryterium jakości**: akceptowalne rozwiązanie problemu
- **ewolucja**: prawdopodobna – poprawa programu, ewolucja środowiska

Program typu S (oparty na Specyfikacji)

- **założenia**: dostępna jest pełna specyfikacja systemu
- **kryterium jakości**: zgodność ze specyfikacją
- **ewolucja**: brak (modyfikacja → nowy problem → nowy program)

[*B.Walter, Uczelniaonline*]

Konserwacja oprogramowania – prawa Lehmana

"Prawa" Lehmana

1. Prawo nieustannej zmiany
2. Prawo wzrastającej złożoności
3. Prawo samoregulacji
4. Prawo organizacyjnej stabilności
5. Prawo zachowania przyzwyczajeń
6. Prawo ciągłego wzrostu
7. Prawo spadku jakości
8. Prawo przyrostowego rozwoju

- Oparte na obserwacjach rozwoju kilku dużych systemów (m.in. IBM OS/360)
- Dotyczą dużych systemów (typu E) tworzonych na zamówienie w dużych organizacjach
- Obserwacje częściowo potwierdzone przez wyniki projektów z serii FEAST

- Niesprawdzone w innych typach oprogramowania (S i P)
- Raczej obserwacje lub hipotezy niż prawa

[*B.Walter, Uczelniaonline*]

Konserwacja oprogramowania - wnioski

- Oprogramowanie, aby pozostało użyteczne, musi ewoluować
- Jakość oprogramowania (zdolność do ewolucji) pogarsza się z upływem czasu
- Rosnąca złożoność oprogramowania w pewnym momencie znacznie utrudnia dalszy rozwój systemu
- **Tempo rozwoju oprogramowania jest w najlepszym przypadku stałe i nie zależy od sposobu zarządzania**

[*B.Walter, Uczelniaonline*]

Konserwacja oprogramowania - modele

- ostrożne planowanie i zarządzanie
- wyczerpująca weryfikacja wersji
- powolna ewolucja

- żywiołowy, wielokierunkowy rozwój
- duża liczba poprawek
- okresowa refaktoryzacja

Zatem można wyróżnić także inną klasyfikację modeli pielęgnacji oprogramowania, opartą na obserwacjach E. Raymonda dotyczących metod open source:

- **styl budowy katedry**: szczegółowo zaplanowany, nie ulegający łatwo zmianom i ewolucji, oraz
- **styl bazaru**, żywiołowo rozwijającego się w wielu kierunkach, charakteryzujący się wieloma wydaniami i koniecznością okresowej restrukturyzacji

elniaonline]

Konserwacja-podsumowanie

- Istnieją cztery zasadnicze rodzaje pielęgnacji oprogramowania: korekcyjna, adaptacyjna, ulepszająca, prewencyjna.
- Dla dużych systemów można sformułować zależności (‘prawa Lehmana), które charakteryzują ewolucję systemu programowego. Zdefiniowano je na podstawie obserwacji i są wskazówkami dotyczącymi sposobów zarządzania procesem pielęgnacji.
- Koszt pielęgnacji (w tym zmiany) oprogramowania zwykle przekracza koszt jego budowy. Firmy utrzymują coraz większa liczbę systemów odziedziczonych, i ponoszą coraz większe koszty na pielęgnację systemów odziedziczonych.
- Wysokie koszty pielęgnacji wynikają z braku stabilności personelu; ze sposobu wytwarzania, które nie zachęcają do tworzenia kodu zdatnego do pielęgnacji; z niedostatecznych umiejętności niezbędnych do pielęgnowania systemu.

Metodyka - definicje

- Metodyka = zestaw pojęć, modeli, technik i sposobów postępowania służący do analizy dziedziny problemu oraz projektowania pojęciowego, logicznego i fizycznego [Subieta].
- „Metodyka wytwarzania oprogramowania to wszystko, co jest regularnie robione w celu wytworzenia oprogramowania (np. dobór pracowników, zadań, zasady współpracy, procedury) →

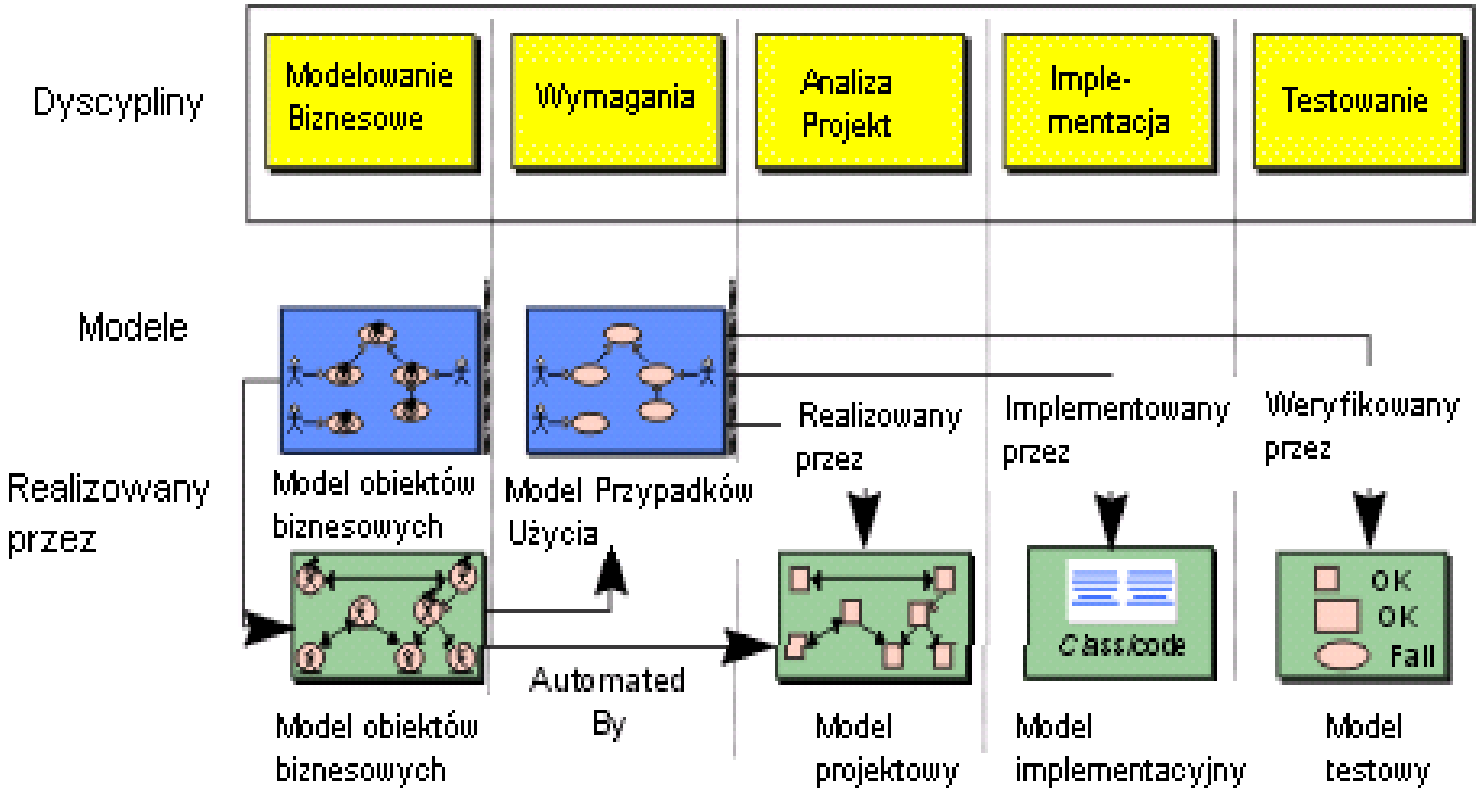
każda organizacja ma swoją własną, niepowtarzalną metodykę” [Cocburn].

Metodyka wytwarzania oprogramowania

- to wszystko, co jest systematycznie wykonywane w celu wytworzenia oprogramowania (np. dobór pracowników, zadań, zasady współpracy, procedury postępowania);

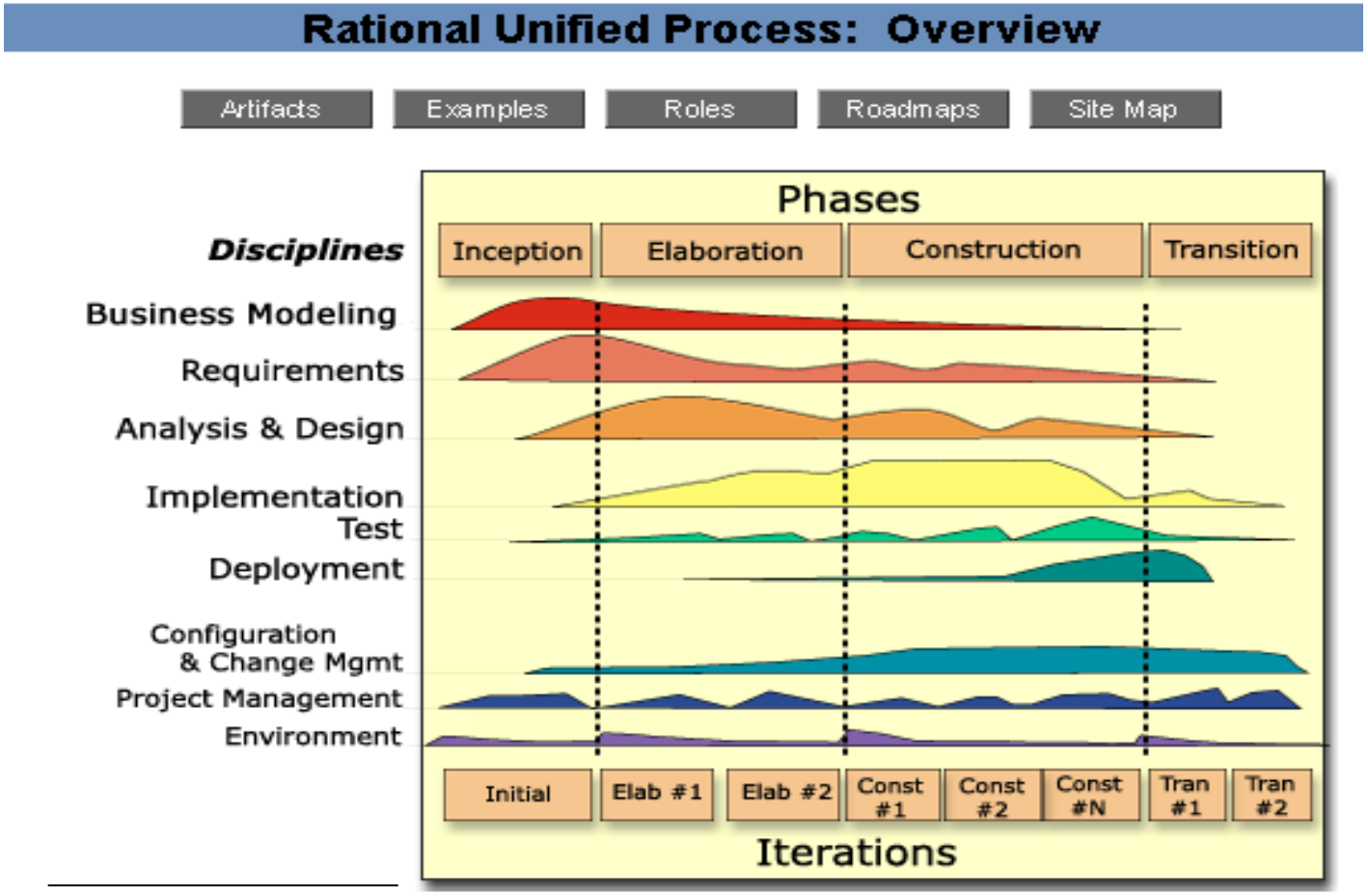
- każda firma wytwórcza ma swoją własną, niepowtarzalną metodykę.

Metodyki sterowane modelami



[RUP 2003]

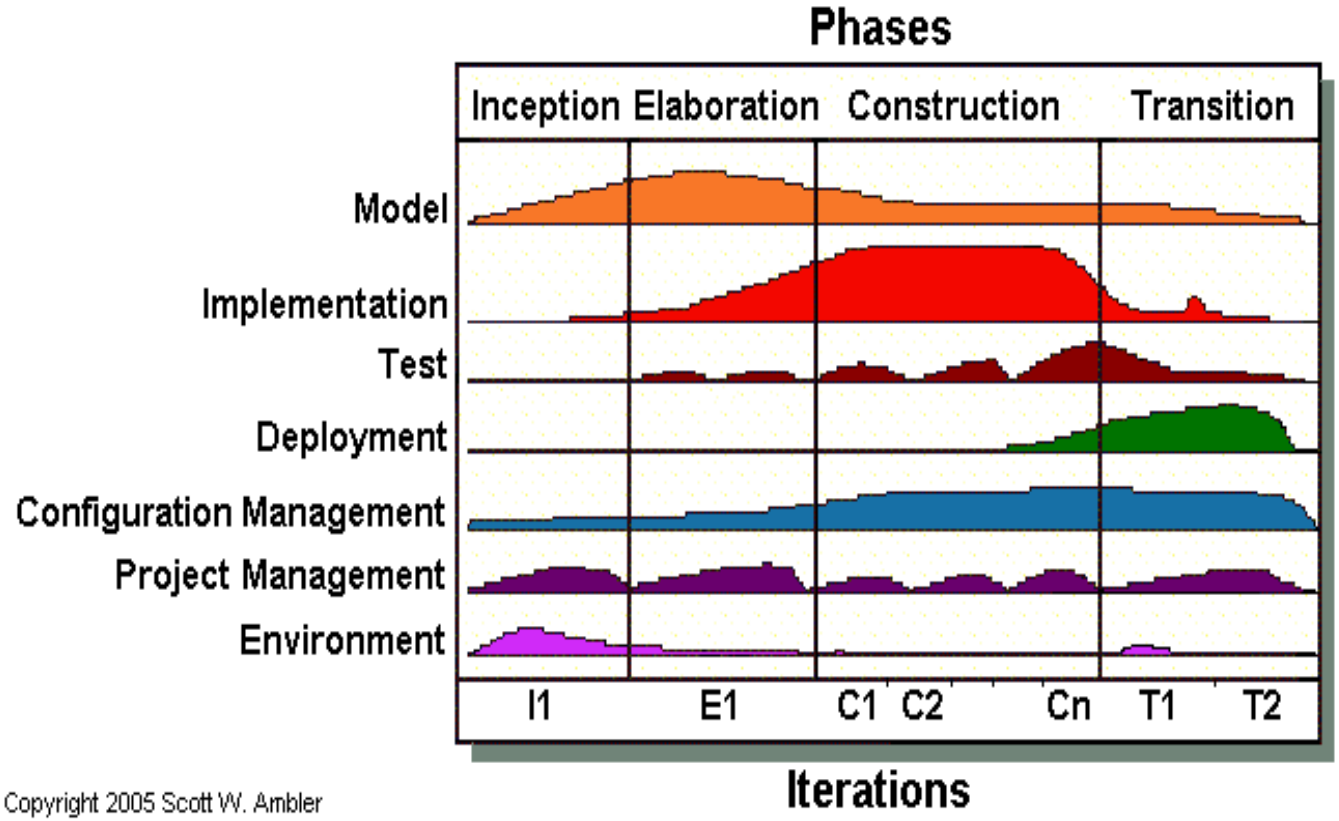
Metodyka RUP* – fazy i dyscypliny



*RUP- RationalUnified Process

[RUP,2003]

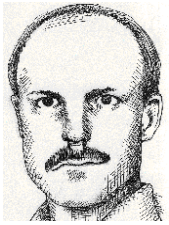
Metodyki AUP* v1.1 – fazy i dyscypliny



*AUP- Agile Unified Process; rodzina metodyk

A może inaczej zarządzać .....??.

Luty 2001, Snowbird, Utah, 17 osób



Kent Beck

(karty CRC, xUnit, XP)

Marin Fowler

(refaktoryzacja, UML Distilled)

Alistair Cockburn

(rodzina metodyk Crystal)

Jim Highsmith

(Adaptive Software Development)

Manifest zwinności (*Agile Manifesto*)

Manifest zwinności - **wartości**

ważniejsze

Jednostki i interakcje niż procesy i narzędzia

Działające oprogramowanie niż obszerna dokumentacja

Współpraca klienta niż negocjacja kontraktu

Nadążanie za zmianami niż trzymanie się planu

Metodyki zwinne

Crystal & Crystal Light

Scrum (broadest, variable end-point)

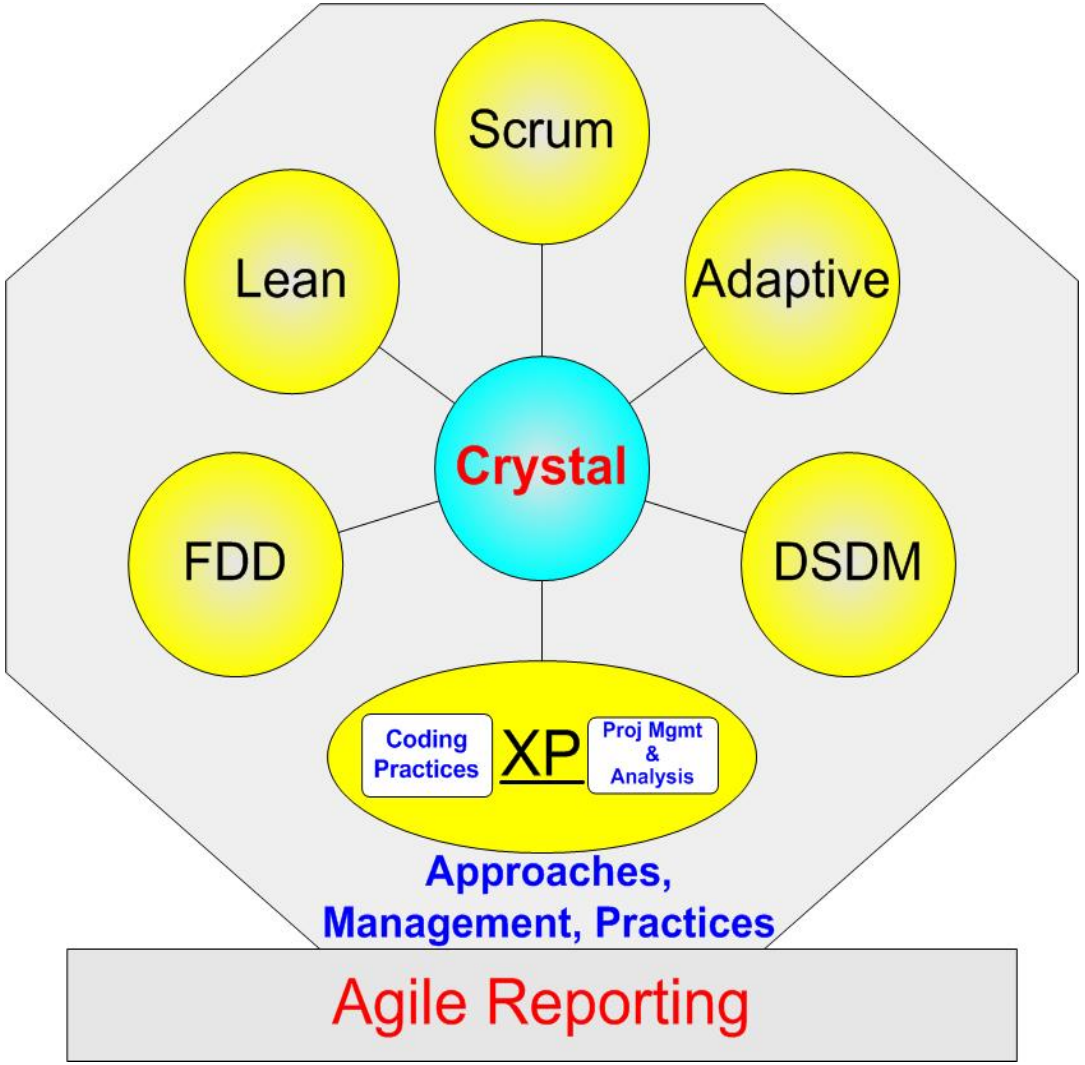
DSDM Model (construction oriented) Dynamic Systems Development Method Ltd.

Lean Development (domain, 80/20 approach)

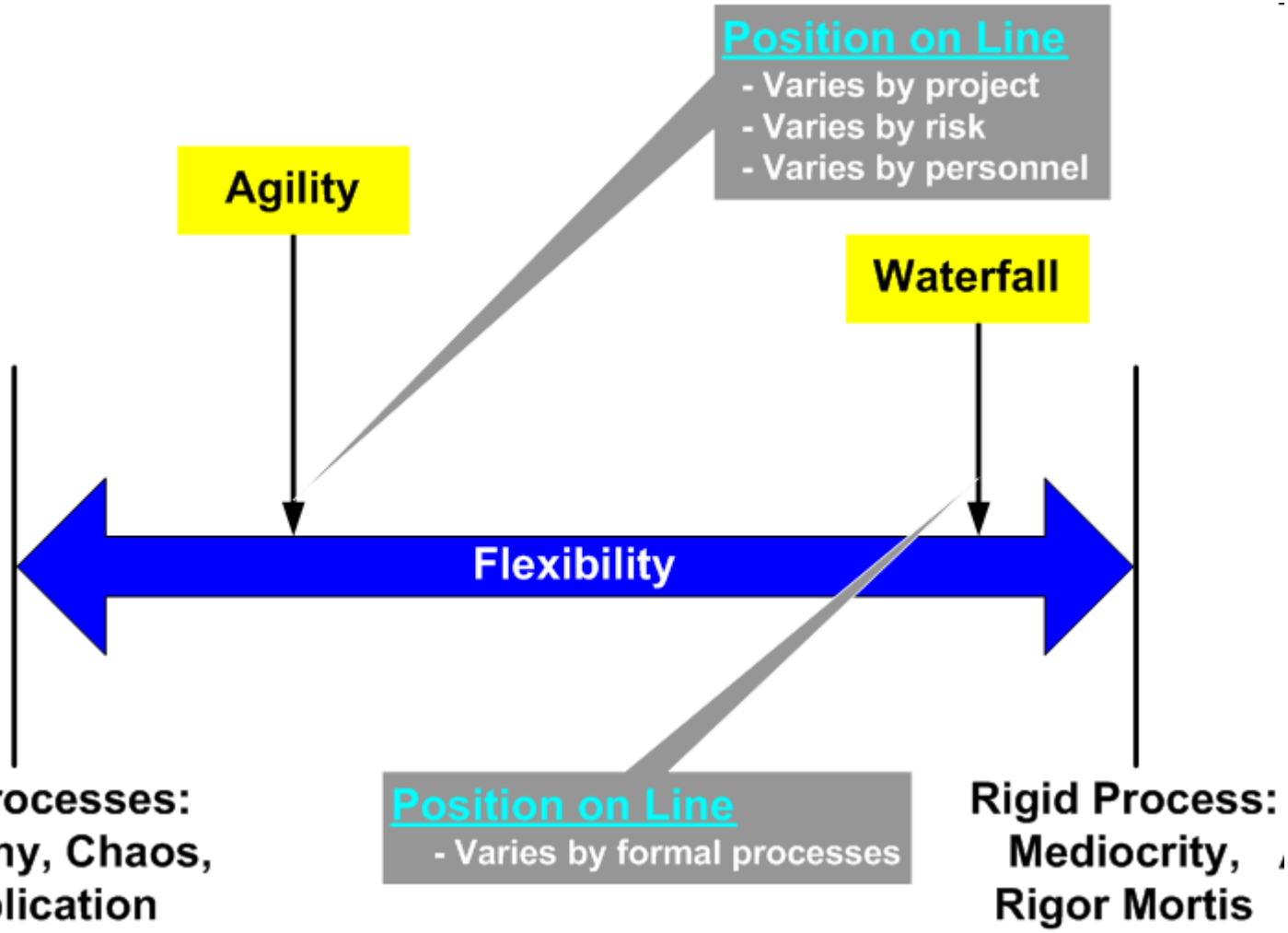
Feature-Driven Development - FDD (most structured)

Adaptive

Extreme Programming – XP

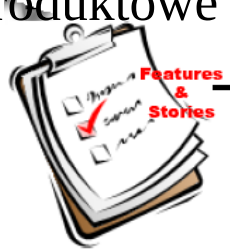


Metodyki zwinne vs. formalne

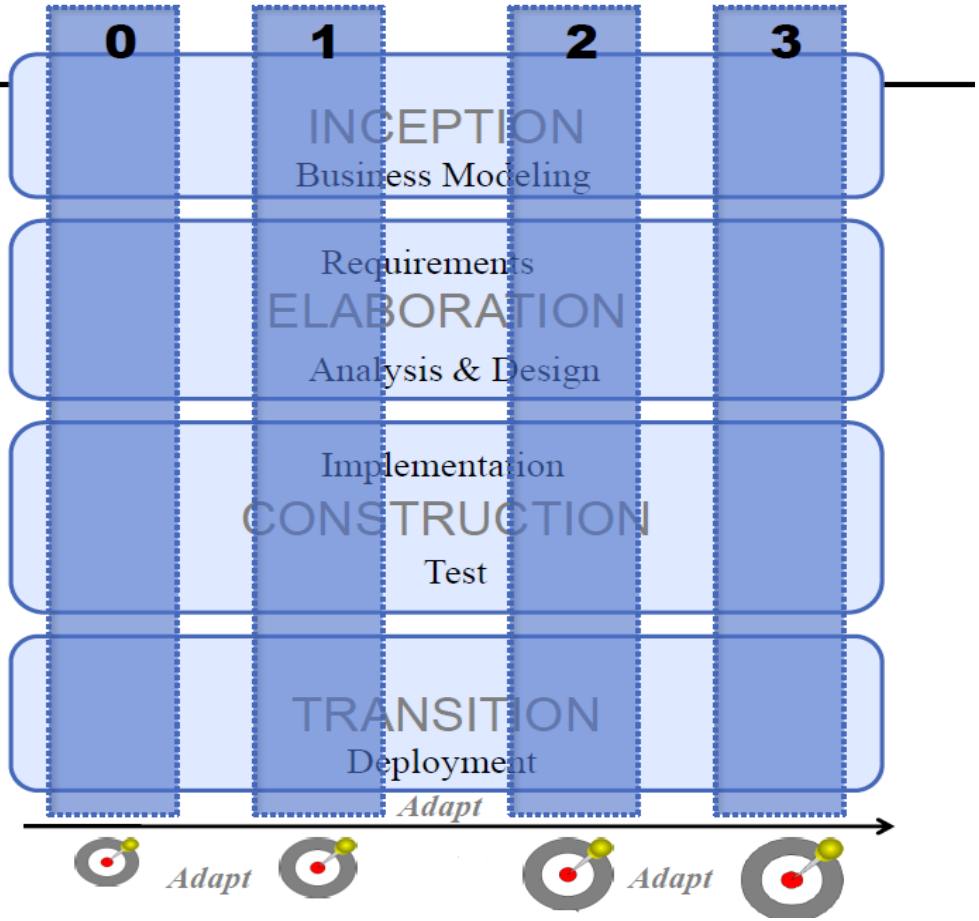


Agile vs. Plan-driven

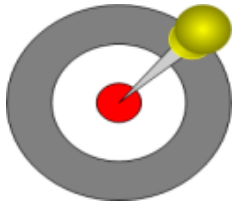
Zamówienia produktowe



Wg wartości bizneso wych



Realizacja celu klienta

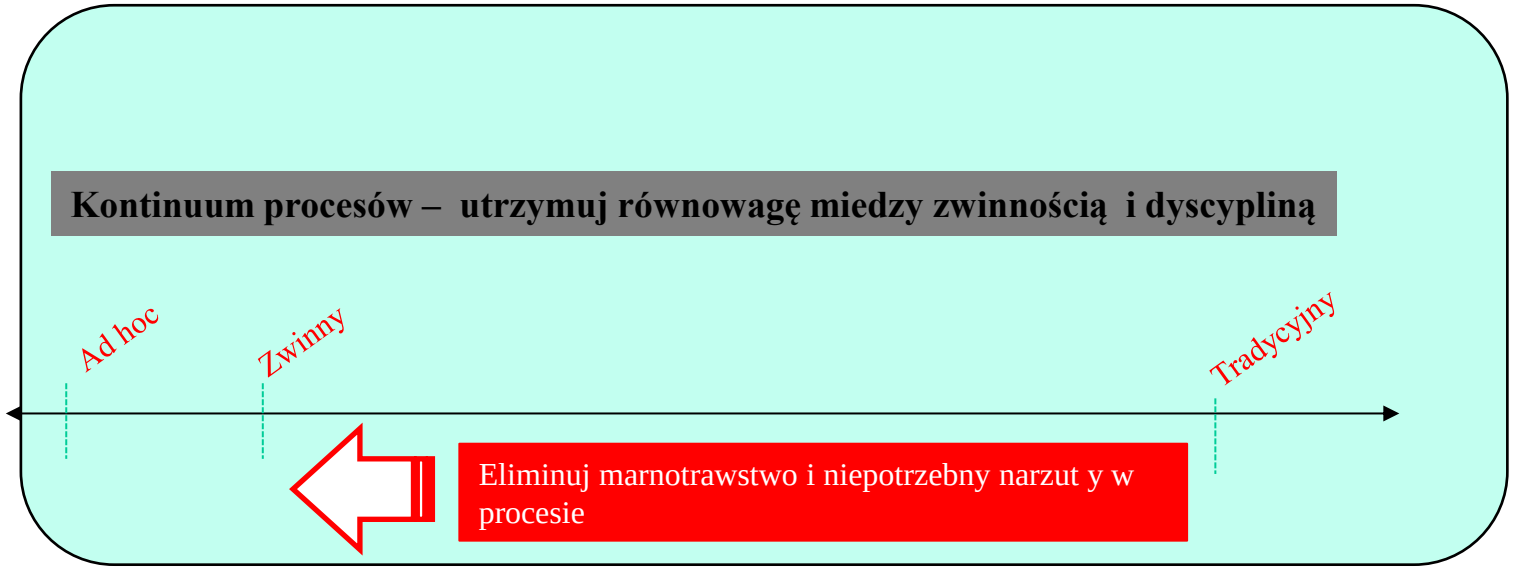


Produkt

Ciągła integracja

[Turgeon]

Dobór metodyki /procesu wytwórczego (1)



Przesuwaj się w kierunku 'pełnej zwinności' bez zwiększania ryzyka projektu do nieakceptowalnego poziomu

? Jak dobierać

Dobór metodyki /procesu wytwórczego (2)

5 czynników istotnych dla wyboru metodyki:

Krytyczność: jaki będzi koszt błędu w końcowym produkcie?
(skala: od brak komforu pracu do strat ludzkich)

Personel: poziom umiejętności członków zespołu projektowego

Rozmiar: liczba zaangażowanych osób.

Dynamika: liczba zmian wymagań /miesiąc (poziom jakości zdefiniowanych prze ludzi wymagań)

Kultura: czy członkowie zespołu stosuje zarządzanie zmianą, występują z inicjatywą , czy też zwykli słuchać nakazów, oczekiwać na zaplanowanie edla nich prac

Boehm and Turner "Balancing Agility and Discipline: A Guide for the Perplexed"

Metodyka a praktyka

Metodyki rekomendują różne zestawy tzw. najlepszych (dobrych?) praktyk (ang. best practices), które:

- mają zredukować nakład prac potrzebny do wykonania systemu informatycznego,
- dać produkt możliwie najwyższej jakości.

Praktyka – definicja ISO/IEC 15504:

Podstawowa praktyka (base practice) – aktywność, która wspiera cele poszczególnego procesu.

- Konsekwentne wykonywanie praktyk podstawowych powiązanych z procesem będzie pomagać w stałym osiąganiu tego celu.
- Spójny zbiór praktyk podstawowych jest powiązany z każdym procesem w obszarze procesu.
- Podstawowe praktyki są opisane na poziomie abstrakcyjnym, określając "co" powinno być zrobione, bez określania "jak" to osiągnąć.
- Implementacja praktyk podstawowych procesu powinna skutkować osiągnięciem podstawowych produktów pracy, odzwierciedlających cel procesu.

Praktyka – definicja (cd)

- Termin **najlepsza praktyka** jest definiowany jako "technika, metoda, proces, aktywność, motywacja lub nagroda, którą uważa się za bardziej efektywną w dostarczaniu określonego rezultatu, niż inną technikę, metodę, proces, etc., jeżeli jest ona stosowana w określonych warunkach" *[wikipedia]*.

- **Praktyka** jest podejściem stosowanym do rozwiązania jednego lub więcej powszechnie występujących problemów. Praktyki są rozumiane jako „kawałki/fragmenty" procesu, stosowane w celu jego adaptacji, rozwijania i konfiguracji *[openUP]*.

- **Praktyki** są konkretnymi aktywnościami i i ich produktami i są częścią frameworku metodyki. *[Ambler]*

Wytwórcze praktyki inżynierskie w metodykach Agile

- Test Driven Development (TDD)
- **Continuous Integration**
- Team Rooms
- Pair Programming
- **Small Releases**
- **Refactoring**
- Collective Code Ownership

Metodyka OpenUP

Programowanie ekstremalne (1)

Programowanie Ekstremalne (XP) to lekka, zwinna (*ang. agile*) metodyka tworzenia oprogramowania

Podstawowe zasady XP:

- Najważniejsza komunikacja ustna.
- Jedyne artefakty: kod + testy
- IEEE/ANSI standard 830/1993? Zbędny !!!
  (IEEE Recommended Practice for Software Requirements Specifications)
- Inspekcje kodu? Zbędne !!!
- Punkty funkcyjne? Zbędne !!!
- Żadnych nadgodzin!

Programowanie ekstremalne (2)

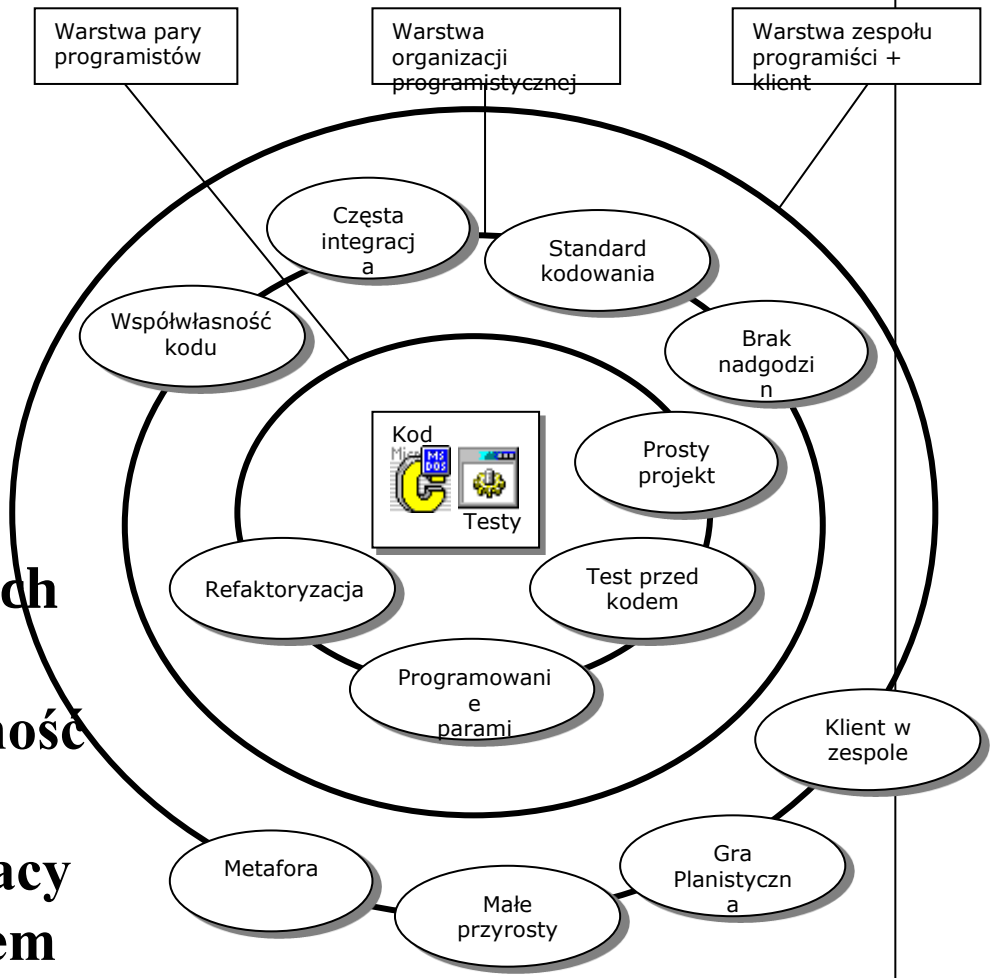
12 praktyk XP wg Kenta Becka:

- **Planowanie**
- **Małe wydania**
- **Metafora systemu**
- **Prosty projekt**
- **Ciągłe testowanie**
- **Przerabianie**
- **Programowanie w parach**
- **Standard kodowania**
- **Wspólna odpowiedzialność**
- **Ciągłe łączenie**
- **40-godzinny tydzień pracy**
- **Ciągły kontakt z klientem**

Metodyka XP

12 praktyk wg Kenta Becka:

- **Planowanie**
- **Małe wydania**
- **Metafora systemu**
- **Prosty projekt**
- **Ciągłe testowanie**
- **Przerabianie**
- **Programowanie w parach**
- **Standard kodowania**
- **Wspólna odpowiedzialność**
- **Ciągłe łączenie**
- **40-godzinny tydzień pracy**
- **Ciągły kontakt z klientem**



Metodyka XP

- praktyki projektowe: 4 praktyki (Whole Team, Planning Game, Small Releases, Customer Tests),
- praktyki dla programisty: 4 praktyki (simple design, Pair Programming, Test-Driven Development, Design Improvement)
- praktyki dla zespołu: 5 praktyk (Continuous Integration, Collective Code Ownership, Coding Standards, Metaphor, Sustainable Pace)

Programowanie ekstremalne (6)

Zespół w XP:

- zespół jest grupą ludzi, którzy razem dążą do celu; są odpowiedzialni za projekt i dbają się o niego; darzą się wzajemnie szacunkiem i mają silne poczucie więzi.
- W zespole XP poza **programistami** są inne role, odpowiedzialne za zarządzanie, oraz pomagające rozwiązywać szczególnie trudne problemy. Są to:
  - ***trener,***
  - ***zarządca,***
  - **tester,**
  - ***konsultant,***
  - **szef**
  - **klient.**
- Wszystkie one mają bardzo ściśle określone kompetencje.
- Wszyscy są odpowiedzialni za proces powstawania aplikacji.

Programowanie ekstremalne (7)

Zalety XP:

- niewykonywanie pracy która nie ma znaczenia, bo zgodnie z zasadą minimalizacji rozwiązania wykonuje się tylko to co jest potrzebne,
- nie ma anulowania projektów, bo ciągły kontakt z klientem zapewnia spełnienie jego wymagań, a jego udział w procesie projektowania zapewnia dobranie właściwego harmonogramu prac
- wykonywaniu pracy, z której nie jest się zadowolonym, bo tworzony kod jest najwyższej jakości, a cały zespół pracuje nad osiągnięciem perfekcyjnego produktu.

Programowanie ekstremalne (8)

Różnice między XP a innymi metodykami :

- wczesne, konkretne i ciągłe informacje zwrotne w krótkich cyklach programowania;
- planowanie przyrostowe, które szybko owocuje planem ogólnym rozwijającym się w czasie trwania projektu;
- zdolność do elastycznego planowania implementacji funkcjonalności, zależnie od zmieniających się potrzeb.

Programowanie ekstremalne (9)

Współtwórca XP Kent Beck:
„XP jest proste w szczegółach, ale trudne w realizacji.”

Słabości XP:

- brak dokumentacji
- klient „na miejscu” i tylko jeden
- zbyt krótka perspektywa planowania

Jak rozwiązać te problemy i zachować zwinność?