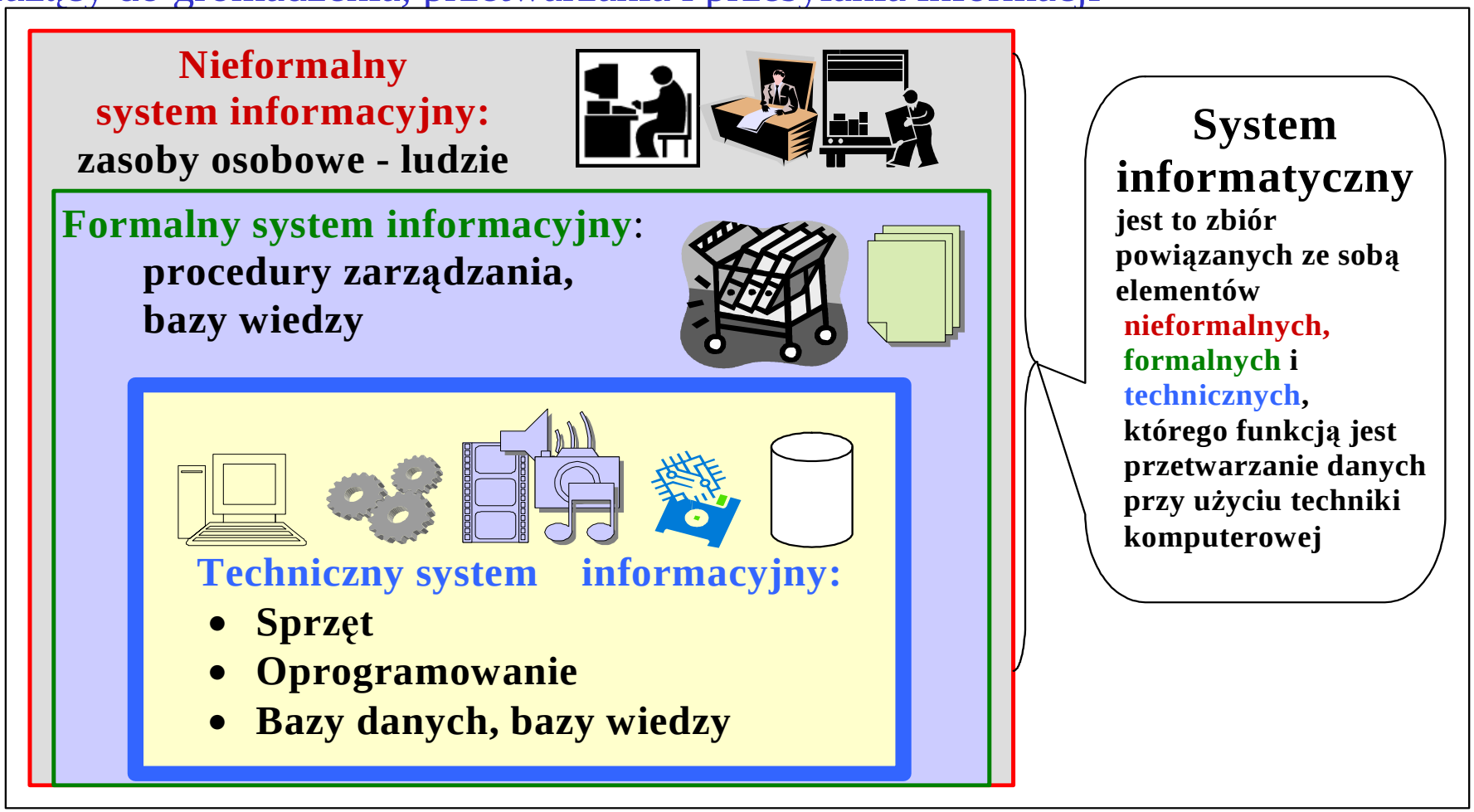
Wykład 1 – część pierwsza

Wstęp do inżynierii oprogramowania.

Cykle rozwoju oprogramowania-iteracyjno-rozwojowy cykl oprogramowania

System Informacyjny =Techniczny SI

- zorganizowany zespół środków technicznych (komputerów, oprogramowania, urządzeń teletransmisyjnych itp.)
- służący do gromadzenia, przetwarzania i przesyłania informacji



I. Obszar inżynierii oprogramowania

Charakterystyka kryzysu oprogramowania:

1. Przekraczanie terminów

1.1. brak właściwych technik budowy oprogramowania

1.2. brak właściwych języków programowania umożliwiających specyfikacje oprogramowania i tworzenie kodu źródłowego

1.3. brak doświadczeń w tworzeniu zespołów specjalistów, zajmujących się tworzeniem programów

1.4. nieumiejętne kierowanie przedsięwzięciem programistycznym

2. Przerywanie prac z powodu utraty aktualności przez realizowany projekt

2.1. wydłużony czas tworzenia oprogramowania,

2.2. szybki rozwój sprzętu

3. Tworzenie programów niezgodnych z wymaganiami klienta

3.1. brak właściwego sposobu porozumiewania się klienta z zespołem informatyków

3.2. brak odpowiednich norm jakości oprogramowania

3.3. niska niezawodność sprzętu i oprogramowania

Źródła powstania **inżynierii oprogramowania -** działu informatyki:

- metody opanowania kryzysu oprogramowania, trwającego od połowy lat sześćdziesiątych
- tworzenie oprogramowania na skalę produkcyjną.

Inżynieria oprogramowania jest wiedzą techniczną, która zajmuje się:

- procesem wytwarzania (produkcją) oprogramowania i jakością tego procesu
- budową oprogramowania i jakością oprogramowania (czyli uzyskanego produktu)

II. Zagadnienia inżynierii oprogramowania

1. **Zarządzanie przedsięwzięciem programistycznym obejmujące:**
   - **1.1.** techniki planowania, szacowania kosztów, harmonogramowania i monitorowania
   - **1.2**. sposoby przygotowania dokumentacji technicznej i użytkowej
   - **1.3.** techniki pracy zespołowej
   - **1.4.** określanie poziomu umiejętności specjalistów
   - **1.5.** zastosowanie narzędzi CASE (*Computer Aided System Engineering*)

2. **Metody analizy, projektowania i implementacji (programowania)**

3. **Pomiary oprogramowania**

3.1. Wyznaczanie i badanie atrybutów **wewnętrznych oprogramowania** obejmujących właściwości struktury oprogramowania $\Rightarrow$ metryki oprogramowania

3.2. Wyznaczanie i badanie **atrybutów zewnętrznych oprogramowania**:

3.2.1. jakości oprogramowania, obejmującej:

» niezawodność (testowalność)

» konserwowalność

» zrozumiałość

» wieloużywalność

» stopień osiągniętej abstrakcji

3.2.2. funkcjonalności

3.3.3. kosztu

4. **Kształtowanie jakości oprogramowania:**

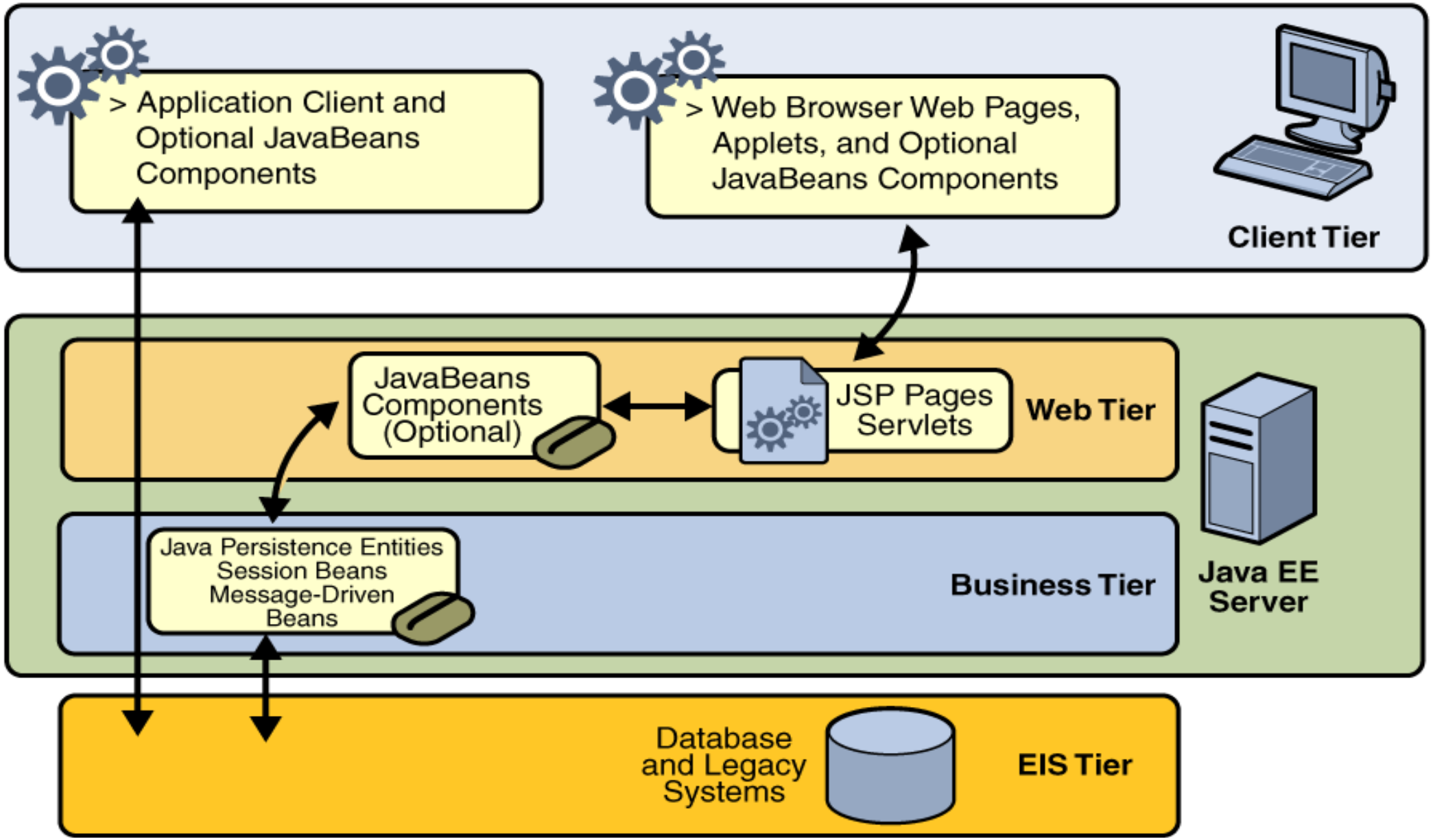
   4.1. sposoby poprawy niezawodności, konserwowalności, wielużywalności, zrozumiałości, stopnia osiągniętej abstrakcji

   4.2. sposoby testowania i walidacji systemów

   4.3. badanie zależności między atrybutami wewnętrznymi i jakością oprogramowania

5. **Rozwój środowisk i narzędzi programistycznych**

Warstwy aplikacji (Java EE)*



Pięciowarstwowy model logicznego rozdzielania zadań (wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe.)

III. Modele procesu wytwarzania oprogramowania - czyli modele cyklu życia oprogramowania

Tworzenie systemu informacyjnego jest powiązane z:

- budową oprogramowania: co i jak wykonać?
- zarządzaniem procesem tworzenia oprogramowania: kiedy wykonać?
- wdrażaniem oprogramowania

Modelowanie struktury i dynamiki systemu (diagramy UML)	**Implementacja struktury (diagramy,**	**i dynamiki systemu generowanie kodu UML)**
co należy wykonać?	*jak należy wykonać?*	
• model przedsiębiorstwa • wymagania • analiza (model konceptualny) • testy modelu	• projektowanie (model projektowy: architektura sprzętu i oprogramowania; dostęp użytkownika; przechowywanie danych) • testy projektu	• programowanie (specyfikacja programu : deklaracje, definicje; dodatkowe struktury danych: struktury „pojemnikowe”, pliki, bazy danych) • testy oprogramowania • wdrażanie • testy wdrażania

Co i jak wykonać? - perspektywy projektowania obiektowych systemów informacyjnych

(wg Alan Shalloway, James R.Trott)

- koncepcji

  (co obiekty powinny powinny robić?)

- specyfikacji interfejsów

  (jak używać obiektow?)

- implementacji

  (w jaki sposób zaimplementować interfejs ?)

- tworzenia i zarządzania

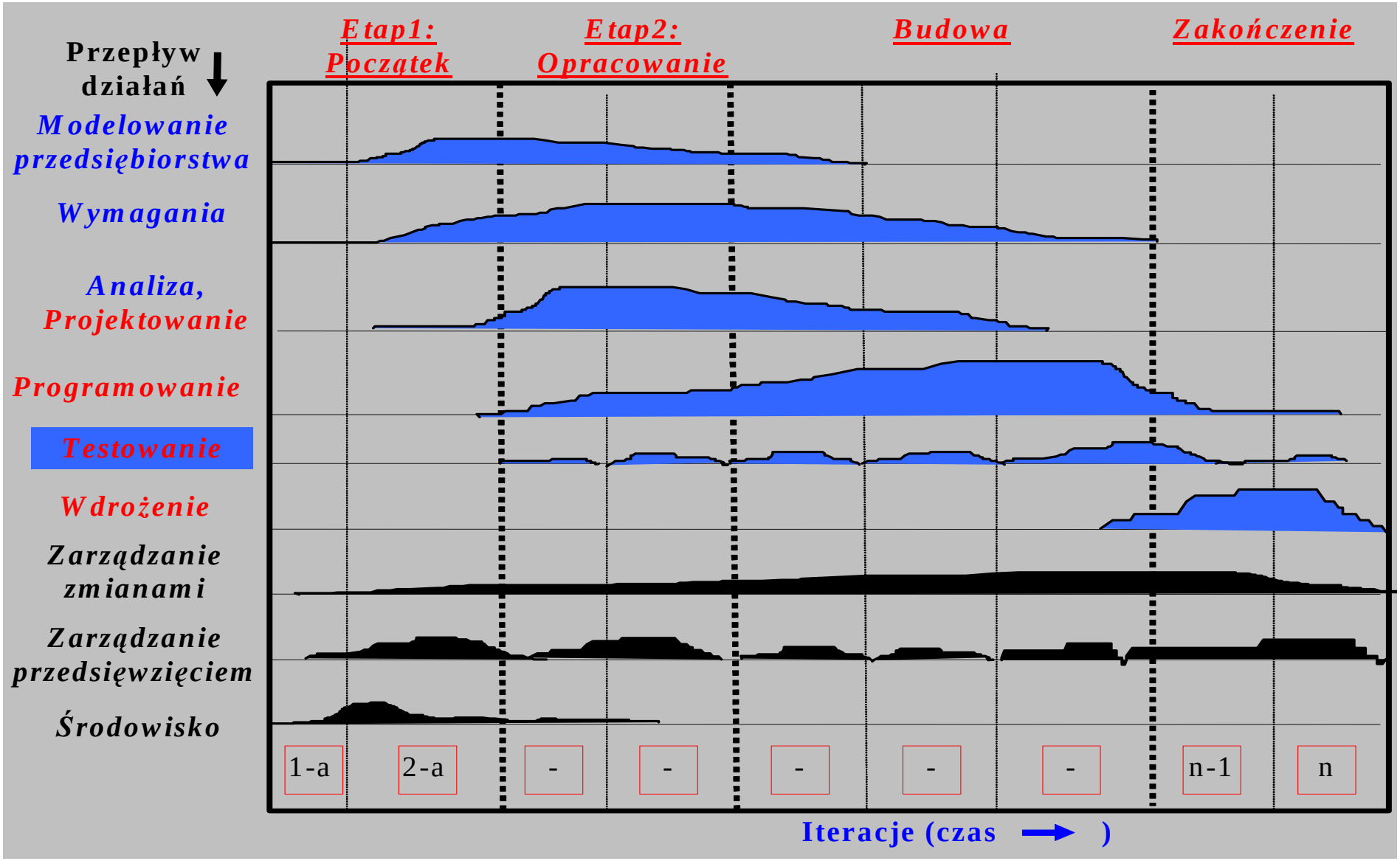
  (*obiekt A tworzy i zarządza B)*

- zastosowania

  (*obiekt A tylko stosuje obiekt B;*

  zabronione jest, aby *obiekt A tworzył i używał B*)

Zunifikowany iteracyjno- przyrostowy proces tworzenia oprogramowania – kiedy?



UML – język wspierający zunifikowany iteracyjno - przyrostowy proces tworzenia oprogramowania

Diagramy UML modelowania strukturalnego

- **Diagramy pakietów**
- **Diagramy klas**
- **Diagramy obiektów**
- **Diagramy mieszane**
- **Diagramy komponentów**
- **Diagramy wdrożenia**

Diagramy UML modelowania zachowania

- **Diagramy przypadków użycia**
- **Diagramy aktywności**
- **Diagramy stanów**
- **Diagramy komunikacji**
- **Diagramy sekwencji**
- **Diagramy czasu**
- **Diagramy interakcji**

Rola diagramów UML 2

- praca zespołowa
- pokonanie złożoności projektu
- formalne, precyzyjne prezentowanie projektu
- tworzenie wzorca projektu
- możliwość testowania oprogramowania we
- wczesnym stadium jego tworzenia

Działanie 1: Wymagania

Czynności	Produkty wyjściowe	Opis produktu wyjściowego
lista kandydujących wymagań	lista znamionowa	status, szacowany koszt, priorytety, poziom ryzyka implementacji itp.
zrozumienie kontekstu systemu	***Model dziedziny*** (*domain model*)-najważniejsze obiekty systemu: „rzeczy” lub zdarzenia podawane przez ekspertów	**diagram najważniejszych klas dziedziny** (*domain classes*) z niewielką ilością operacji-metod (około 10-50 w notacji UML), reszta przewidywanych klas w glosariuszu (*glossary*);
	Model biznesowy (*business model*) -wewnętrzny model procesu biznesowego organizacji, wyszczególniany przez klientów systemu (*customers)*	**„business use case” :** **a) opis „uses cases” i „actors”** odpowiadających procesowi biznesowemu (*the business*) oraz klientom (*customers*) procesu biznesowego **b) biznesowy model obiektowy** (*business object model*) składający się z wykonawców (*workers*), encji biznesowych (*business entities*), jednostek pracy (*work units*) realizujących „use case”,

funkcjonalne wymagania	***„Use-case" model*** (identyfikacja „use cases" z modelu biznesowego)	proces reprezentowania wymagań jako „use cases" oraz innych produktów specyfikowany za pośrednictwem języka UML: **1) model „use cases" zawierający „actors" i „use cases" oraz powiązania** (np. dziedziczenia) między nimi (klasy zawierające atrybuty i operacje) oraz: **dodatkowo diagramy** (*diagrams*): stanów(*statecharts*), czynności (*activity*), sekwencji akcji (*sequence*) oraz współpracy (*collaboration*), **przepływ zdarzeń** (*flow events*)-opis tekstowy realizujących zachowanie systemu przy działaniu poszczególnych „use case" lub „actors" i „use case" - czyli opis sekwencji akcji odpowiednich do modyfikacji, przeglądu, projektowania i testowania, **specjalne wymagania** (*spevial requirements*) zawierające niefunkcjonalne wymagania (*nonfunctional requirements*) w postaci opisu tekstowego, **2) opis architektury** (*architecture description*) „use cases" , **3) glosariusz** (*glossary*) - definicje ważnych termów wyprowadzanych z modelu dziedziny (*domain model*) lub modelu biznesowego (*business model*), **4) prototyp interfejsu użytkownika** (*user-interface prototype*)- interakcje między „actors" - ludźmi i oprogramowaniem
niefunkcjonalne wymagania	uzupełniające wymagania lub (*supplementary requirements*)- indywidualne wymagania	ograniczenia środowiska i implementacji (np. typ komputera, typ plików, rodzaj systemu operacyjnego, typ oprogramowania Internetu), zależności, konserwacja, zdolność do poszerzania,