Technologia Programowania 2017/2018
Wykład 1

Jakub Lemiesz

Informacje o kursie

- Zasady zaliczenia, listy zadań, konsultacje:

  ki.pwr.edu.pl/lemiesz

- Slajdy z wykładu (zapisz ten adres):

  ki.pwr.edu.pl/lemiesz/TP2017

Cel kursu

- Znacie język programowania, umiecie tworzyć niewielkie aplikacje, ale nie macie doświadczenie w analizowaniu, projektowaniu i implementacji większych systemów
- Uczymy się systematycznego podejścia do analizowania i projektowania systemów (*technologii programowania*)
- Dlaczego systematyczne podejście jest istotne?
  1. złożoność domeny i systemu (*złożony - mający wiele aspektów*)
  2. częste zmiany wymagań i koncepcji

Cel kursu: metody walki ze złożonością

1. Złożoność dziedziny
   - ▷ programowanie obiektowe (*rzeczywitość → obiekty*)
   - ▷ OOA/D - Object Oriented Analysis and Design (*usystematyzwane podejście do anlalizy i projektowania*)

2. Złożoność projektu (*'extreme software flexibility'*)
   $\Longrightarrow$ wzorce projektowe

3. Złożoność systemu - przekracza zdolności pojmowania jednej osoby (*np. naprawienie błędu powoduje inne*)
   - ▷ testowanie na bieżąco (*np. JUnit*)
   - ▷ systemy kontroli wersji (*np. SVN,Git,...*)
   - ▷ dobre metodyki (*metodyki zwinne*)

Cel kursu: metody walki ze zmianami

- Przyczyny pojawiania się zmian w projekcie:
  - niepełna analiza, nieuświadomione wymagania
  - zmienne wymagania (*np. zmiany organizacyjne*)
  - czynniki zewnętrzne: nowe technologie, prawo, ...
  - warto spojrzeć na raport NIK o Edukacji.CL
- Im później następuje zmiana tym problem i koszt większy (*"entropia" systemu wzrasta z czasem*)
- Aktualne podejście: zwiększyć elastyczność wykorzystując metodyki zwinne (*agile programming, np. RUP, Scrum*)

Plan wykładu

- OOA/D (*analiza i poprawne modelowanie obiektowe*)
- Unified Modeling Language (*UML*)
- wzorce projektowe
- popularne narzędzia i technologie
  (*JUnit, Mockito, SVN, Git, Akka, Play, Hibernate, Android, XML, regex,...*)

Źródła wiedzy

[1] *Robert C. Martin*
Zwinne wytwarzanie oprogramowania. Najlepsze zasady, wzorce i praktyki. (Helion 2014)

[2] *Craig Larman*
UML i wzorce projektowe. Analiza i projektowanie obiektowe oraz iteracyjny model wytwarzania aplikacji.

[3] *Erich Gamma et al.*
Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku

OOA/D - Object Oriented Analysis and Design

Przychodzi do nas Klient...

*"Napiszcie mi system, który umożliwałby prowadzenie **sprzedaży** w sieci moich **sklepów**. Przede wszystkim powinien on wspierać obsługę **kas** i umożliwiać przeprowadzanie różnego rodzaju **płatności**. Musi również archiwizować transakcje i sprzedane **produkty**. I ma być wygodny w obsłudze!."*

Specyfikacji klienta jest:

1. obiektowa - opisuje obiekty i zdarzenia z rzeczywistości
2. nieprecyzyjna (*doprecycowanie przez przypadki użycia*)
3. zawiera wymagania funkcjonalne i niefunkcjonalne

Specyfikacja: kolejność powstawania artefaktów

diagram przypadków użycia

↓

przypadki użycia

(*scenariusz główny, alternatywne*)

↓

diagramy sekwencji

↓

operacje systemowe

operacje systemowe
model dziedziny
} → kontrakty operacji

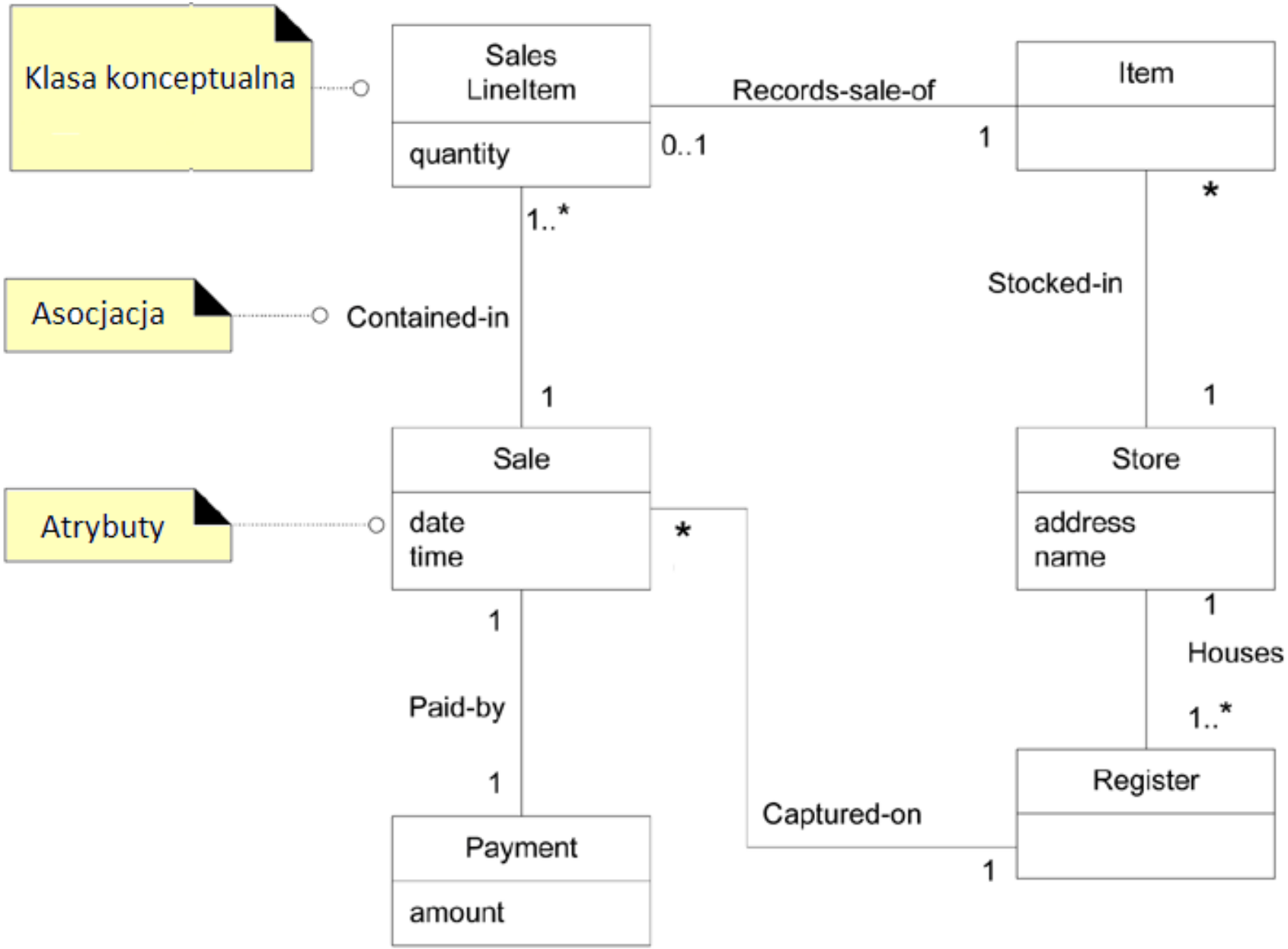
↓

projektowanie systemu...

Model dziedziny

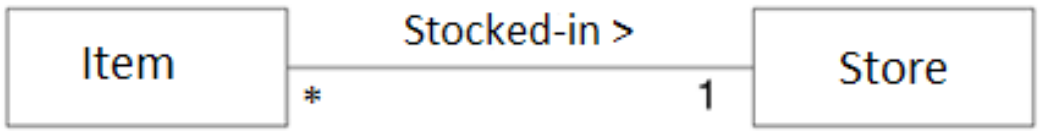
- Identyfikacja i graficzna reprezentacja obiektów - nie tylko materialnych (*tzw. klasy konceptualne*), powiązań miedzy nimi (*asocjacji*) i własności (*atrybutów*)
- Zmniejsza rozdźwięk między naszym obrazem rzeczywistością a modelem projektowym (*łatwiej się myśli*)
- Inspiracja dla obiektów programistycznych (*to nie jest diagram klas!*)

Model dziedziny - przykład (*książka Larmana*)



Model dziedziny – asocjacje

Asocjacja to relacja między instancjami klas opisująca ważne powiązanie

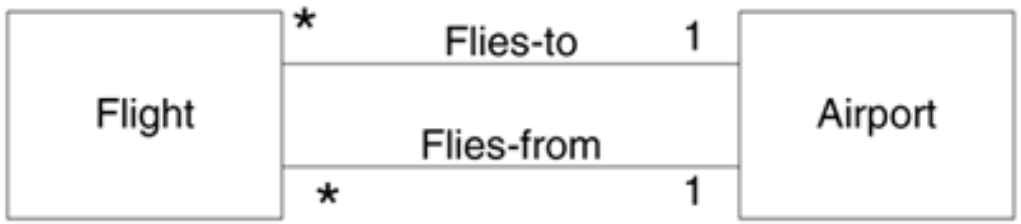


1. Dlaczego tylko ważne powiązania?
   (*ile krawędzi ma graf o n wierzchołkach?*)
2. Asocjacja jest relacją dwukierunkową (*a strzałka?*)
3. Posiada oznaczenia określające liczbę instancji pozostających w danej relacji

Model dziedziny – krotność asocjacji

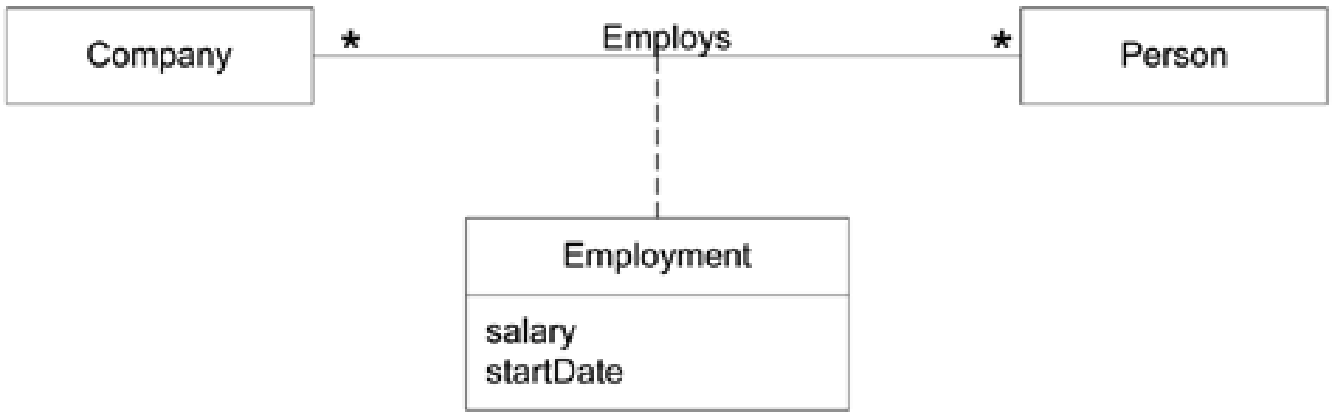
Wartość krotności informuje ile instancji klasy może być powiązanych z inną w jednym momencie

Model dziedziny – wielokrotne asocjacje



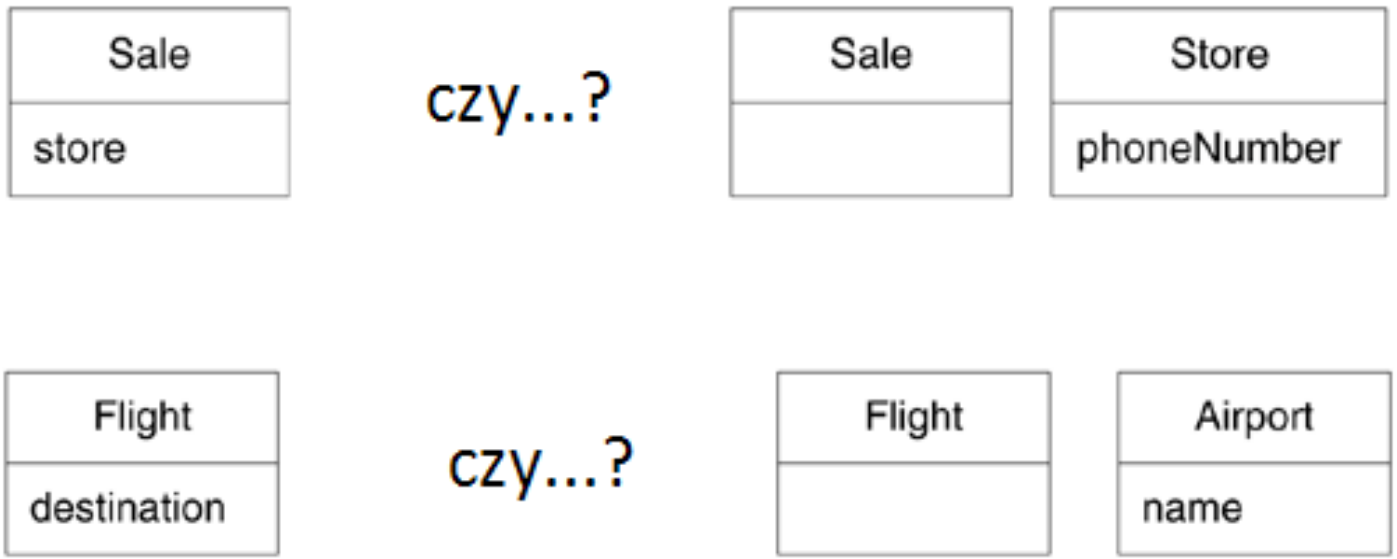
Dwie klasy mogą być powiązane kilkoma różnymi asocjacjami

Model dziedziny – klasy asocjacyjne



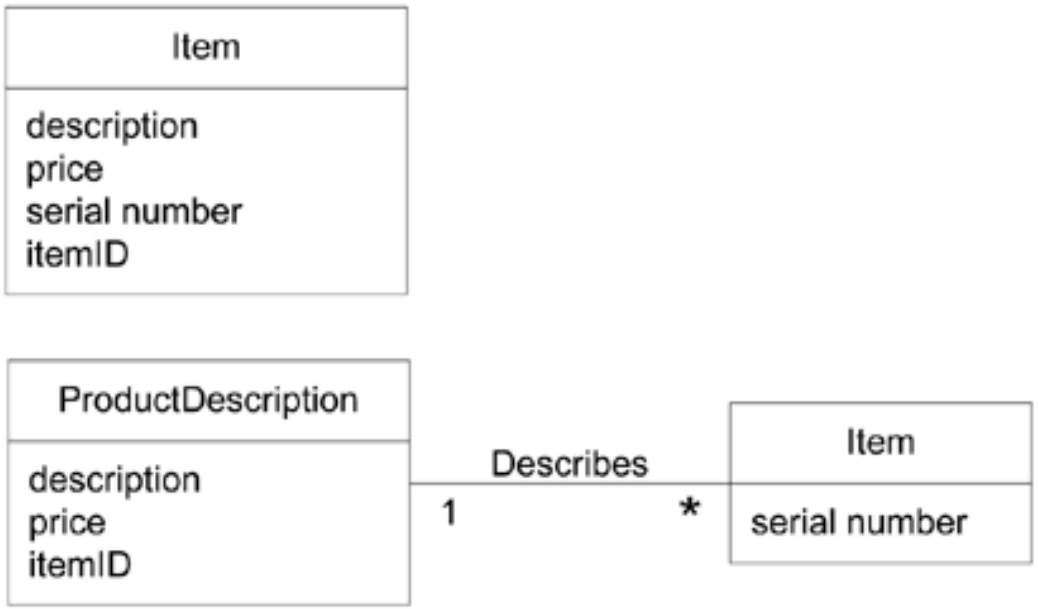
- Klasa asocjacyjna – asocjacja jako klasa z atrybutami
- Kiedy używać? Gdy zachodzi relacja wiele-do-wielu, a każde powiązanie wymaga przechowywania osobnych informacji

Model dziedziny – czy to atrybut czy klasa?



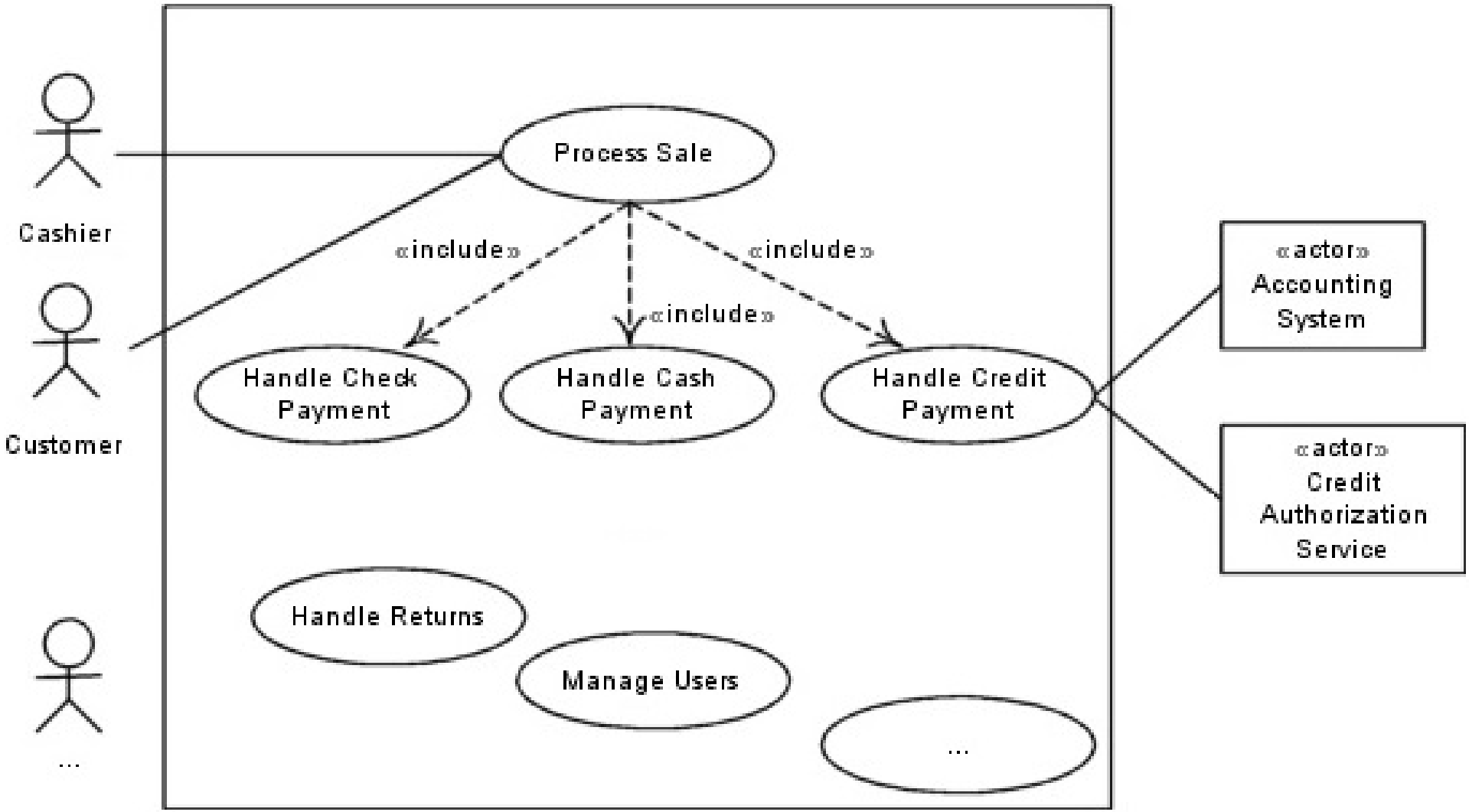
- Częsty błąd: atrybutem jest coś, co powinno być klasą
- Zazwyczaj atrybut to krótki tekst lub liczba z którą nie są powiązane dodatkowe informacje

Model dziedziny – klasy opisowe



- Wzorzec *Item – Descriptor* - klasa opisowa, która zawiera informacje opisujące inną klasę
- Kiedy? Potrzebny opis niezależny od istnienia instancji (*np. usunięcie ostatniej instancji grozi utratą informacji*)

Diagram przypadków użycia - aktorzy i funkcjonalność



(*Źródło: http://brasil.cel.agh.edu.pl/ 09sbfraczek*)

Przypadek użycia (*ang. use case*)

1. Przypadek użycia to lista kroków opisująca interakcje między systemem a aktorem w dążeniu do istotnego celu.
2. Jest często tworzony z udziałem przyszłych użytkowników, w celu odkrywania i dokumentowania wymagań.
3. Opisuje zachowanie systemu obserwowane z zewnątrz, uwzględnia różne scenariusze, w tym nietypowe.
4. Diagram przypadków użycia pełni rolę pomocniczą, obrazuje w przyjazny sposób aktorów i funkcjonalność.

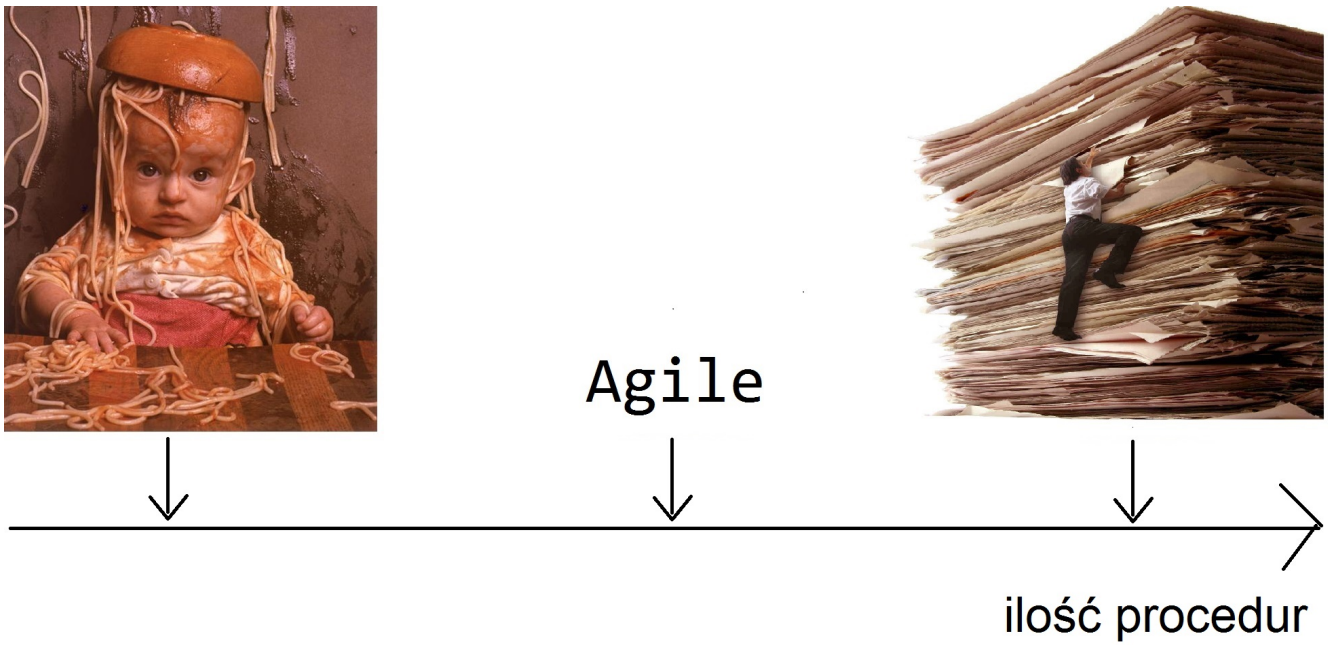
Przykład przypadku użycia (*więcej: książka Larmana*)

Name: Process sale , **Actors:** Cashier, Customer
Preconditions: Cashier is identified and authenticated
Postconditions: Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Receipt is generated.

Main Success Scenario:
1. Customer arrives at a POS checkout with items to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records item and presents total.
5. Cashier repeats steps 3-4 for all items.
6. System presents total with taxes calculated. 7. ...

Alternate Flows:
3a. Invalid identifier:
   1. System signals error and rejects entry.
3b. There are multiple of same item category:
   1. Cashier can enter item category identifier and the quantity.

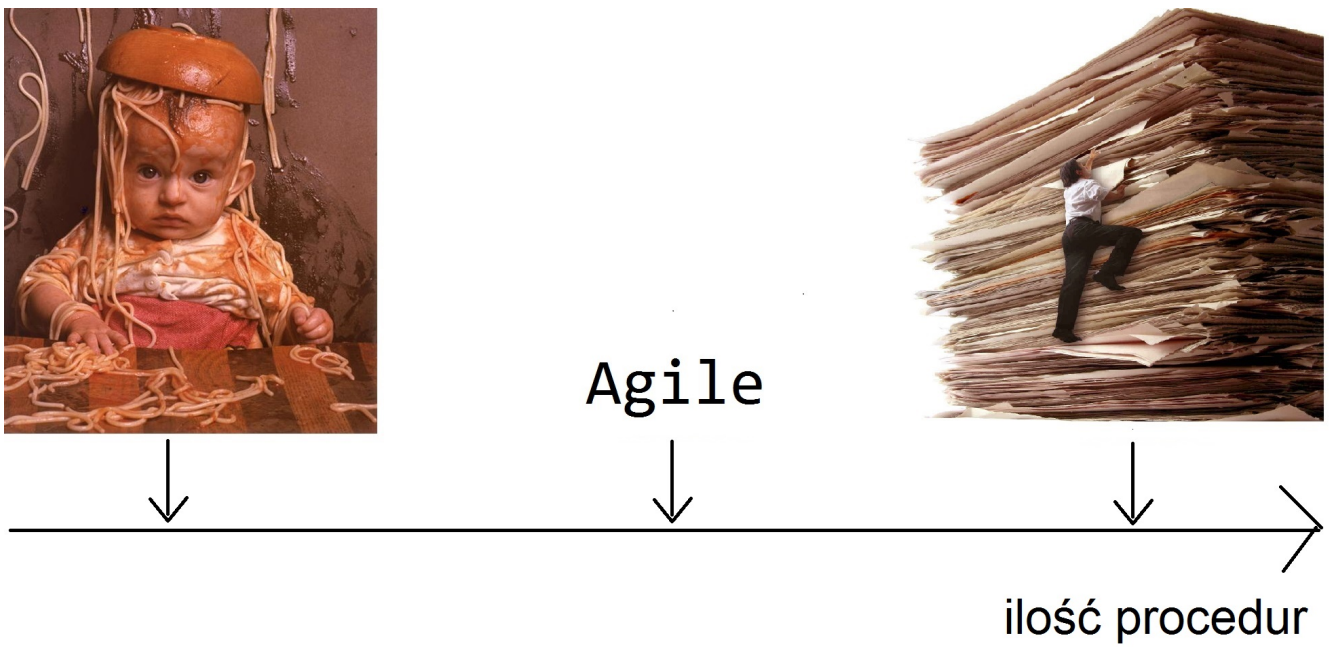
Metodyka - czy metodyka jest potrzebna?



Metodyka to usystematyzowanie procesu mające ułatwić planowanie i monitorowanie postępu

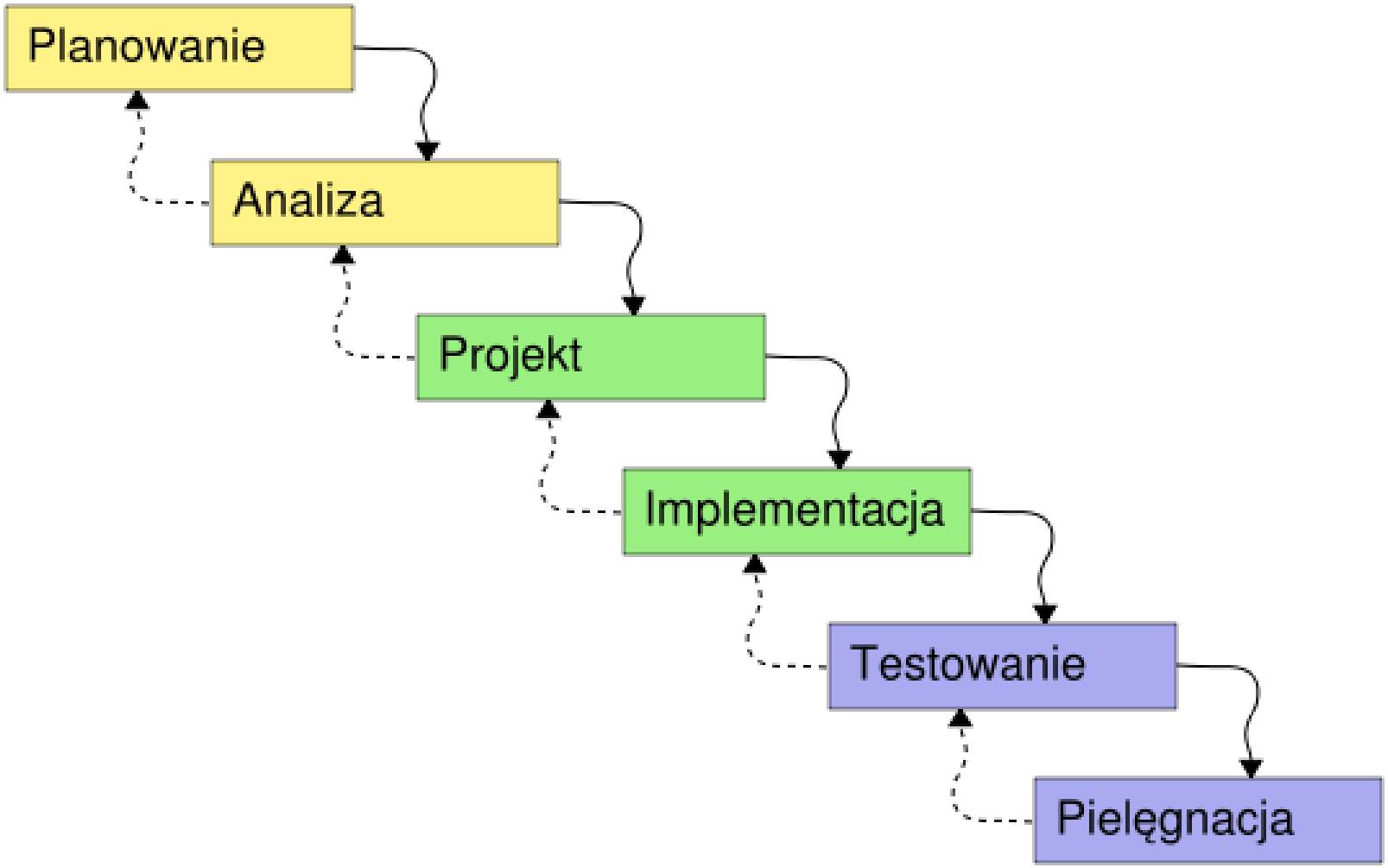
Np. podział na fazy: analiza, projektowanie, implementacja, testowanie; ustalenie ról członków zespołu, jaka dokumentacja powinna powstać, reguły zmiany fazy,...

Metodyka - czy metodyka jest potrzebna?



Ciekawa lektura: dodatek C w książce Martina

Model kaskadowy (waterfall model) ~1970



Źródło wykresu: Wikipedia

Model kaskadowy – krótkie podsumowanie

1. Gdy odkrywamy błędy wracamy do wcześniejszej fazy. Powtarzamy aż otrzymamy satysfakcjonujący produktu.

2. Główne wady:
   - kosztowne powtarzanie od początku wielu czynności (*złe wymagania wielokrotnie kosztowniejsze od błędu programistycznego*)
   - długa przerwa w kontaktach z klientem
   - nie można zmienic fazy przed zakończeniem aktualnej

3. Model kaskadowy może być użyty gdy wymagania i architektura systemu są precyzyjnie określone (*np. zespół zrobił już podobny system*)

Manifest Zwinnego Wytwarzania Oprogramowania

Agile Manifesto (2001, USA)

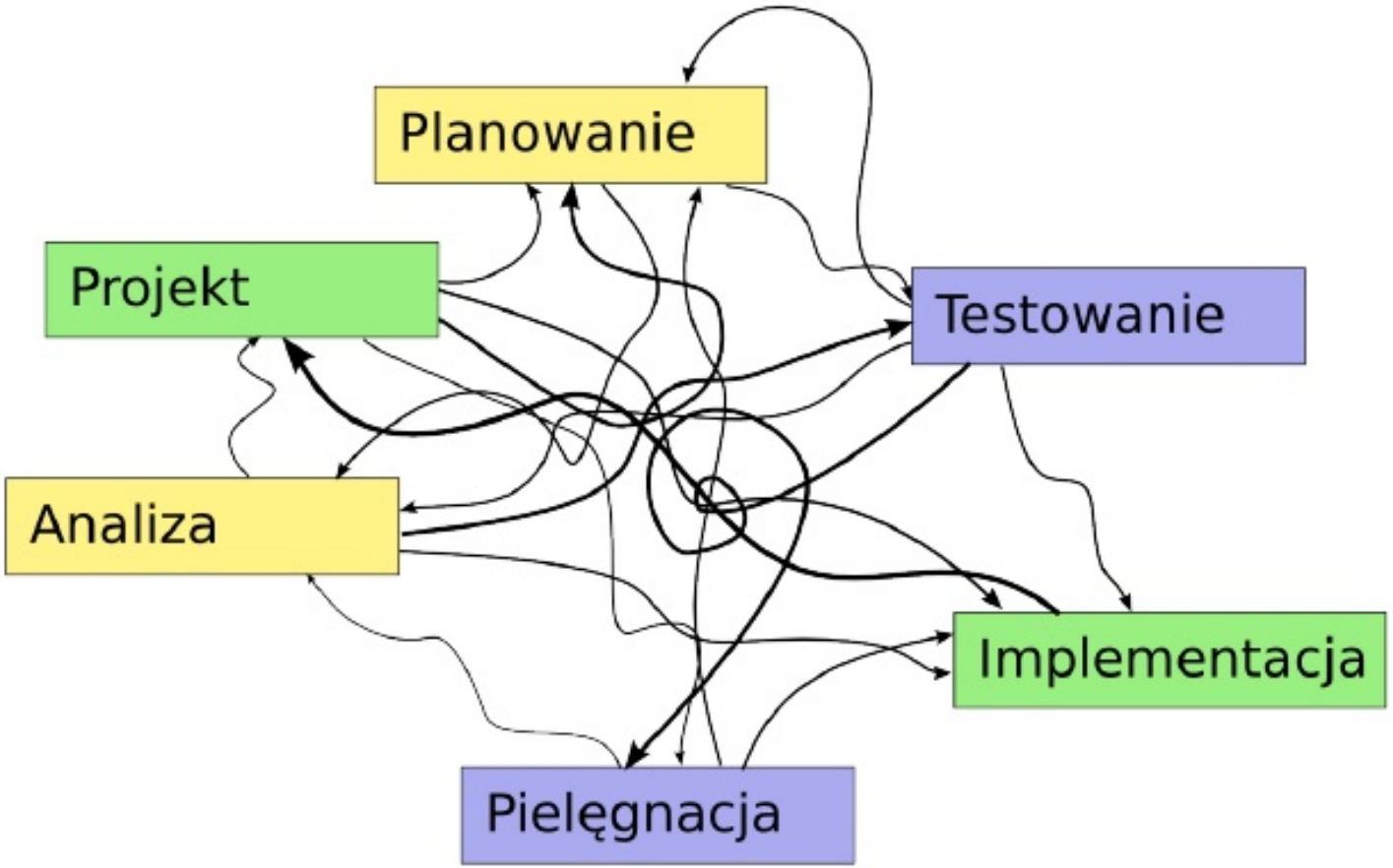
Przedkładamy:

- ludzi i interakcje ponad procesy i narzędzia,
- działające oprogramowanie ponad obszerną dokumentację,
- współpracę z klientem ponad formalne ustalenia,
- reagowanie na zmiany ponad podążanie za planem.

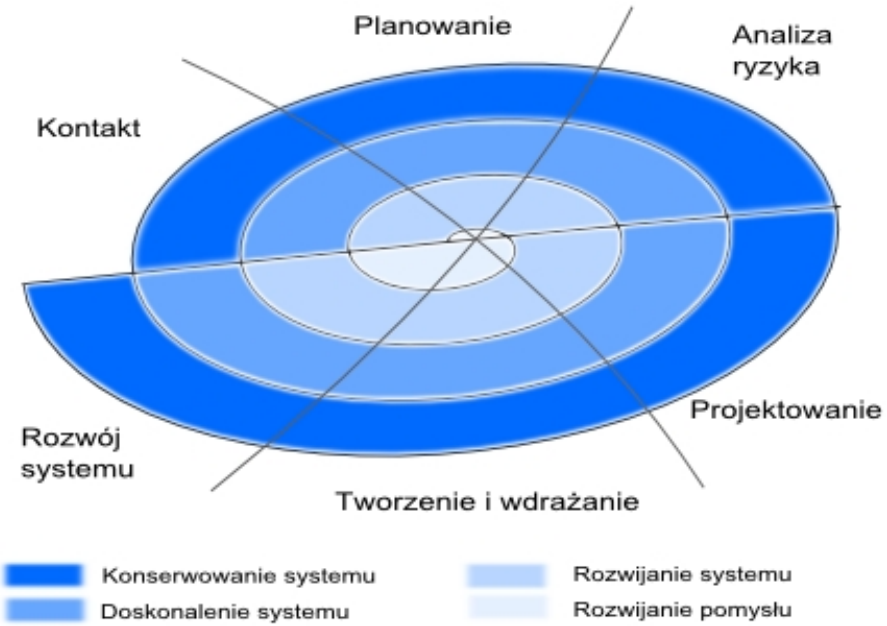
Doceniamy to, co po prawej stronie, bardziej cenimy to, co po lewej.

(*ang. agile* $\approx$ *zwinny*)

Agile???

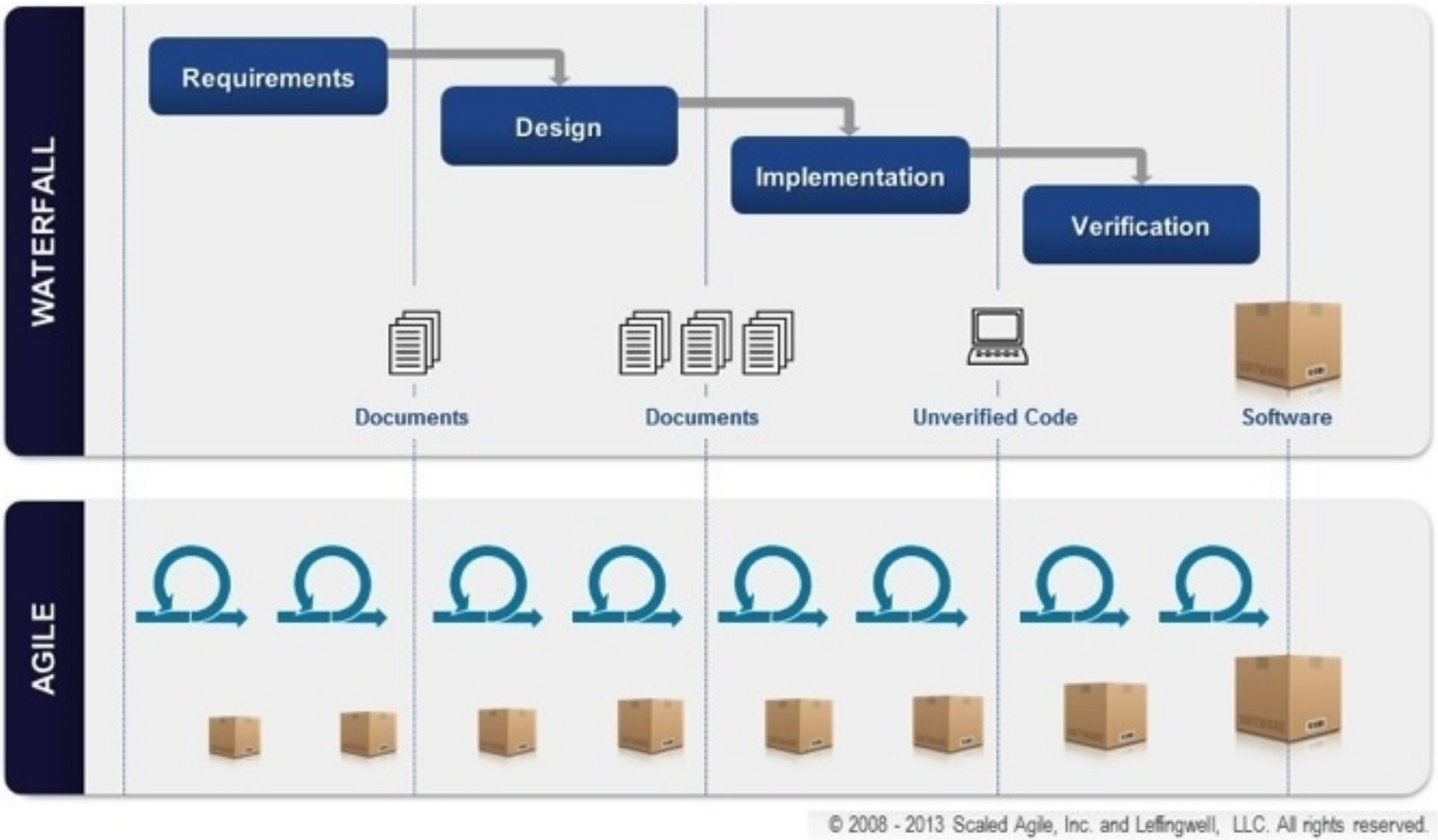

Planowanie
Projekt
Testowanie
Analiza
Implementacja
Pielęgnacja

Agile w praktyce - podejście iteracyjne i przyrostowe



Źródło: webhosting.pl

Agile - przyspiesza dostarczanie produktu

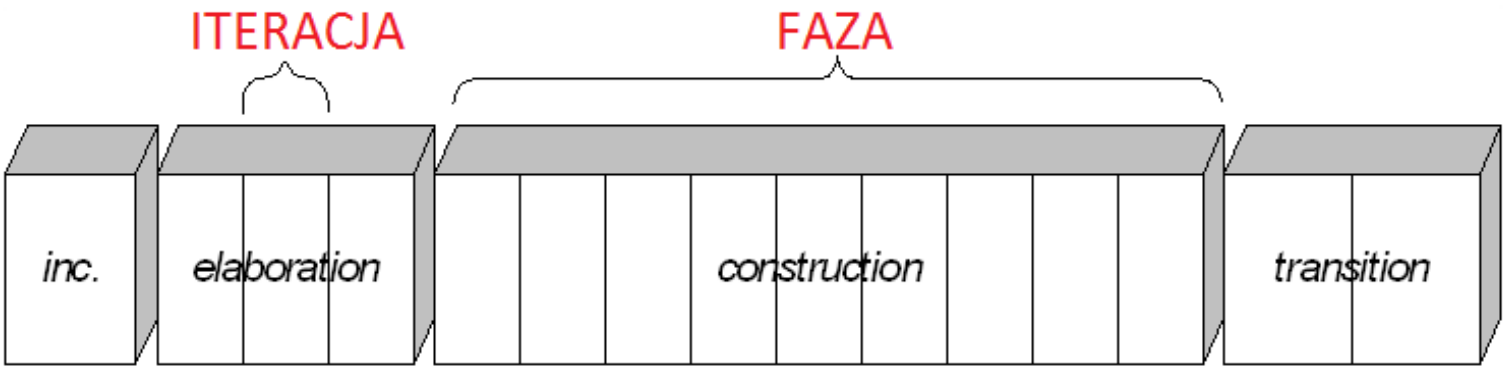


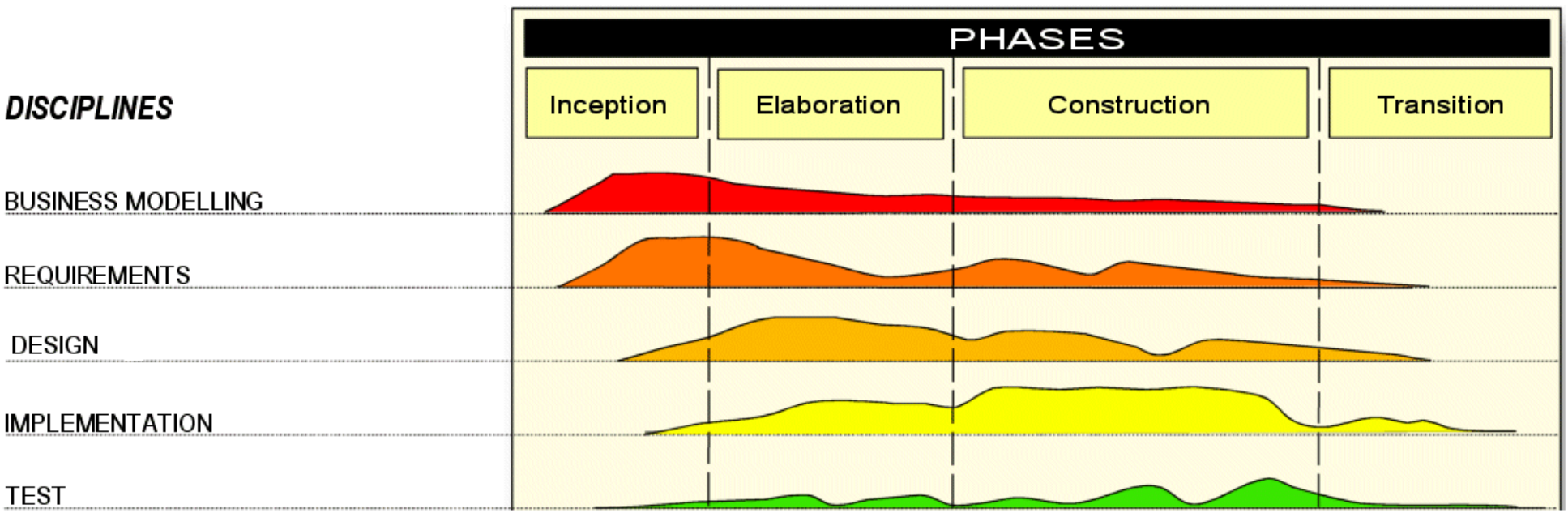
Agile - co jest najważniejsze?

1. Adaptacja do zmieniających się wymagań (*zwinność*)
2. Krótkie iteracje zakończone dodaniem istotnej funkcjonalności (*zauważalne postępy*)
3. Bliska współpraca z klientem (*"codzienna"*)
4. Szybkie wytwarzanie oprogramowania wysokiej jakości (*brak prototypowania*)
5. Małe zespoły i bezpośredni kontakt, zamiast rozbudowanej dokumentacji
6. Projekt uwzględnia informacje zwrotne od deweloperów

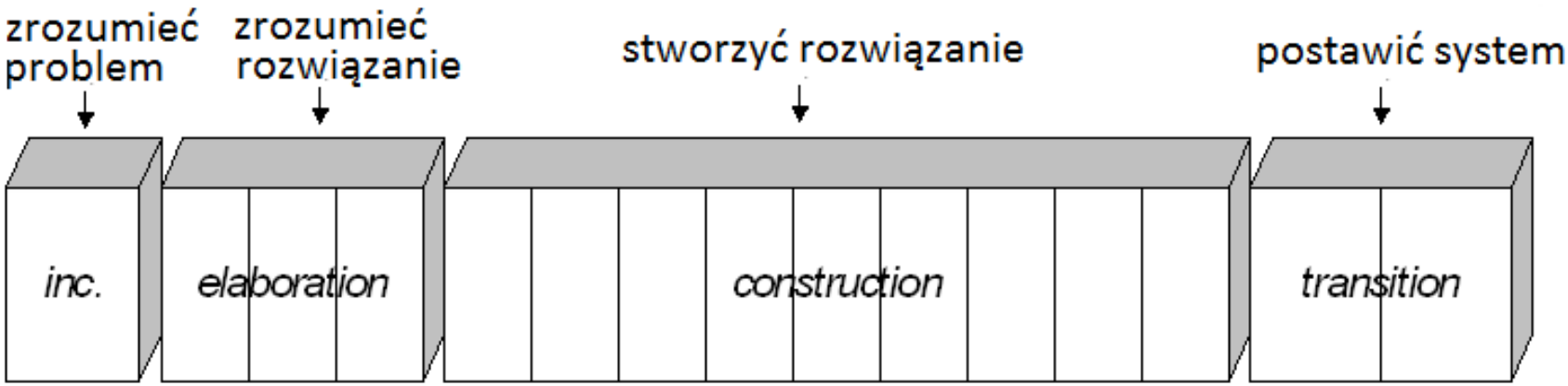
Konkretne metodologie w duch Agile: RUP, Scrum, eXtreme Programming,...

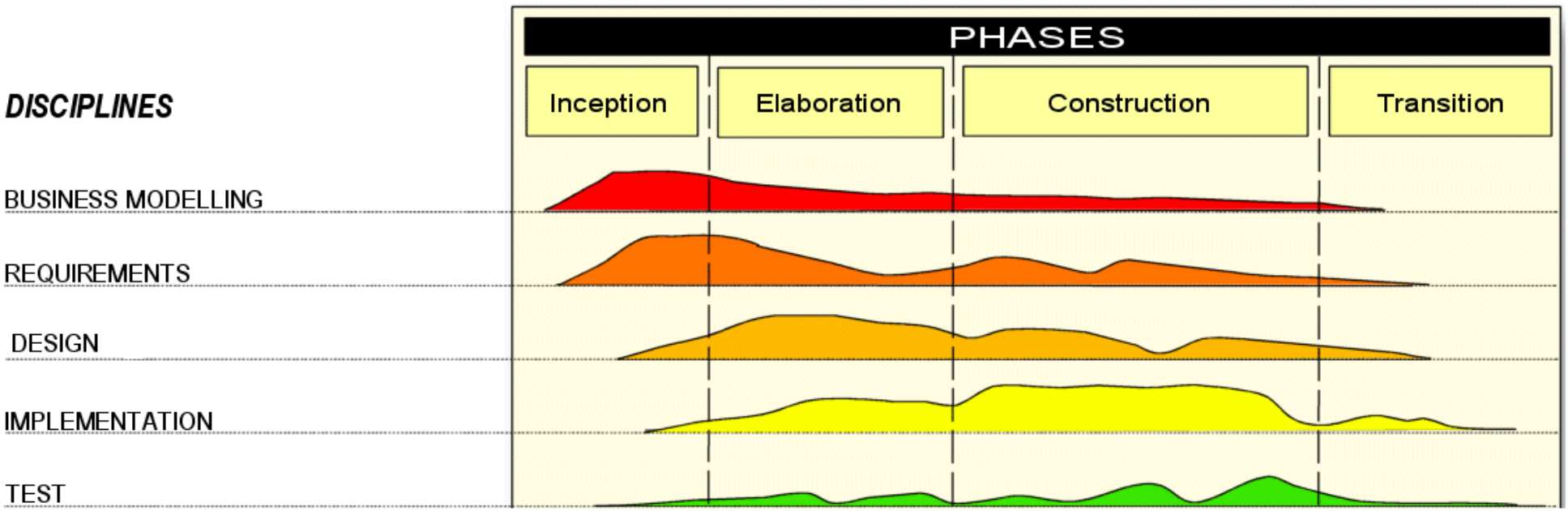
Rational Unified Process (*źródło: C.Larman*)



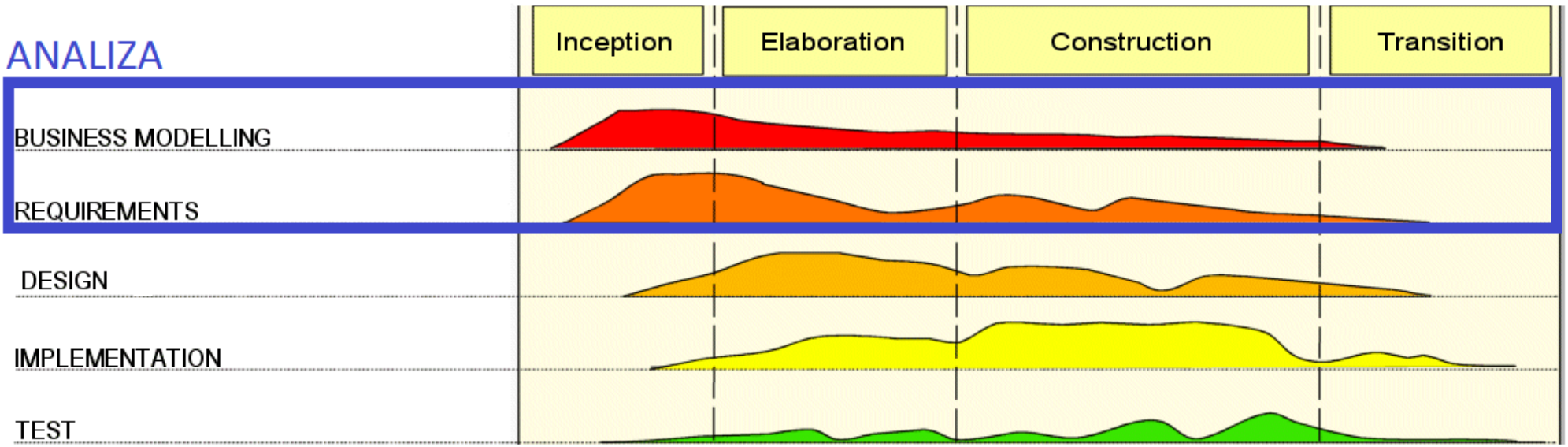


Rational Unified Process (*źródło: C.Larman*)



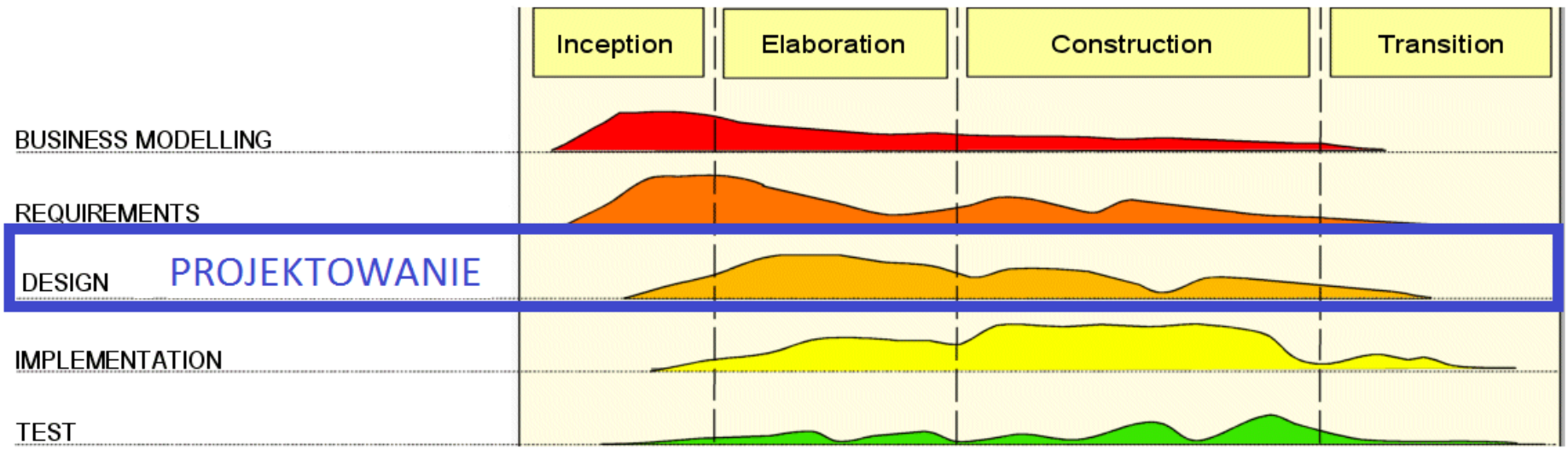


Analiza wymagań w metodykach zwinnych



- Business Modeling (słownik , model dziedziny,...)
- Requirements, wymagania funkcjonalne (przypadki użycia) i niefunkcjonalne (np. wydajność, bezpieczeństwo)
- To robimy na pierwszych ćwiczeniach

Projektowanie w metodykach zwinnych



- Projektowanie obiektów programistycznych i architektury systemu (diagramy klas, wzorce projektowe, ...)
- Najbliższe wykłady i ćwiczenia

Agile - eXtreme Programming

XP - skrajny przykład metodyki zwinnej:

- tworzenie niewielkich projektów wysokiego ryzyka
  (*gdy nie do końca wiadomo co się robi i jak to zrobić*)
- ciągłe modyfikacje architektury
- *test driven development*
  (*testy jednostkowe na początku każdej iteracji*)
- programowanie parami
- jeśli kontakt z klientem jest dobry można się obyć bez formalnej specyfikacji