Projektowanie zwinne

wg.Robert C. Martin, Micah Martin
Agile
Programowanie zwinne
Zasady, wzorce i praktyki zwinnego wytwarzania oprogramowania w C#

Struktura prezentacji

1. Symptomy złego projektu
2. Wprowadzenie do zarządzania projektowaniem oprogramowania metodą iteracyjno-rozwojową
3. Zasady projektowania zwinnego
4. Przykłady metodyk

Struktura prezentacji

1. **Symptomy złego projektu**

(1) Symptomy złego projektu

- Sztywność

  Bardzo wiele zmienianych modułów w celu wprowadzenia nawet najdrobniejszych zmian

- Wrażliwość

  Tendencja do ulegania uszkodzeniom lub usterkom w wielu miejscach wskutek wprowadzenia nawet najprostszych zmian. Często usterki pojawiają się w miejscach, gdzie wydają się być nie związane z dokonanymi zmianami.

- Nieelastyczność

  Projekt zawiera elementy, które mogłyby być wykorzystane w innych systemach, jednak nie mogą być oderwane od oryginalnego systemu.

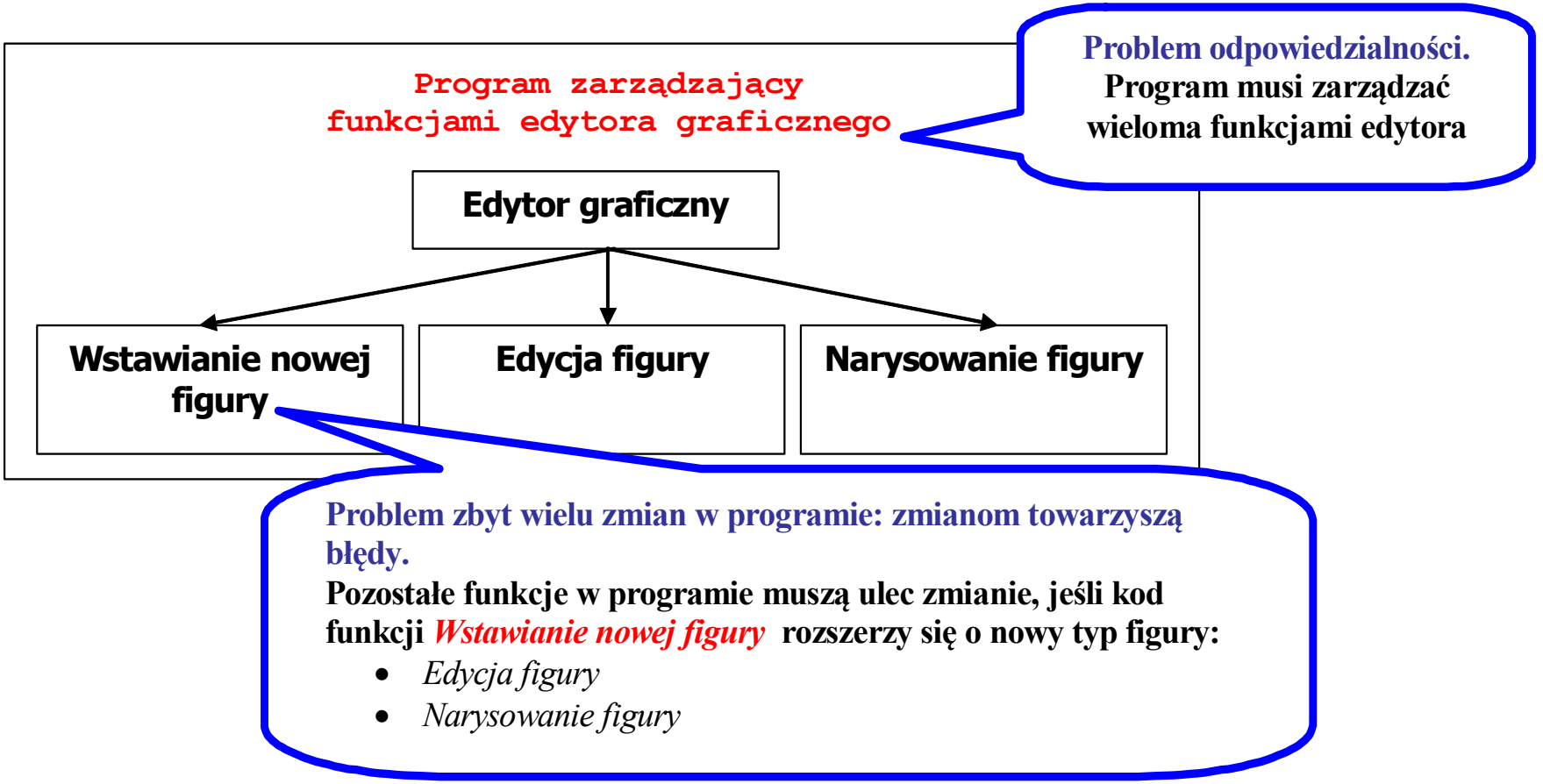
(2) Symptomy złego projektu

- Niedostosowanie do rzeczywistości
  - Niedostosowanie oprogramowania do zmian (zmiany można realizować na wiele sposobów – sposoby prostsze są sprzeczne z projektem)
  - Niedostosowanie środowiska (długi czas kompilacji, zbyt długa kontrola wersji itd.)
- Nadmierna złożoność
  - Program zawiera elementy, które są zbędne. Wynikają one z przedwczesnego przystosowania kodu do ewentualnych zmian, jednak skutek jest odwrotny – część udogodnień może być nigdy nie wykorzystana.
- Niepotrzebne powtórzenia
  - Skutek zastosowania wycinania i wklejania fragmentów kodu przez różnych uczestników tworzenia programu
- Nieprzejrzystość
  - Niezrozumiały, trudny do odczytania kod – programista nie wczuwa się w rolę innego programisty, który też powinien zrozumieć dany kod w celu jego rozwijania (brak również komentarzy)

Przykład 1: Konsekwencje podejścia nieobiektowego opartego na dekompozycji funkcjonalnej przy zmianach wymagań

Centralizacja odpowiedzialności (sztywność, wrażliwość, nielestyczność)

Edytor graficzny



Przykład 2: Konsekwencje podejścia nieobiektowego opartego na dekompozycji funkcjonalnej przy zmianach wymagań

Centralizacja odpowiedzialności (sztywność, wrażliwość, nielestyczność)

Sporządzanie rachunków

1. Rachunek:
   - oblicza cenę brutto zakupionego produktu (wg atrybutów: cena netto, podatek, promocja)
   - zna liczbę zakupionych produktów
2. Rachunek podaje swoją wartość: sumuje wartość zakupu każdego produktu obliczając cenę brutto na podstawie atrybutów danego produktu i mnożąc cenę brutto przez liczbę zakupionego produktu.

Wnioski:

- Duża odpowiedzialność rachunku w obliczaniu wartości rachunku oraz duża wrażliwość na zmiany algorytmów obliczania wartości zakupu produktu
- Zmiany działania rachunku w odniesieniu jednego produktu mogą wpływać w przypadkowy sposób na obsługę innych produktów i powodować błędy.

Struktura prezentacji

1. Symptomy złego projektu
2. **Wprowadzenie do zarządzania projektowaniem oprogramowania metodą iteracyjno-rozwojową**

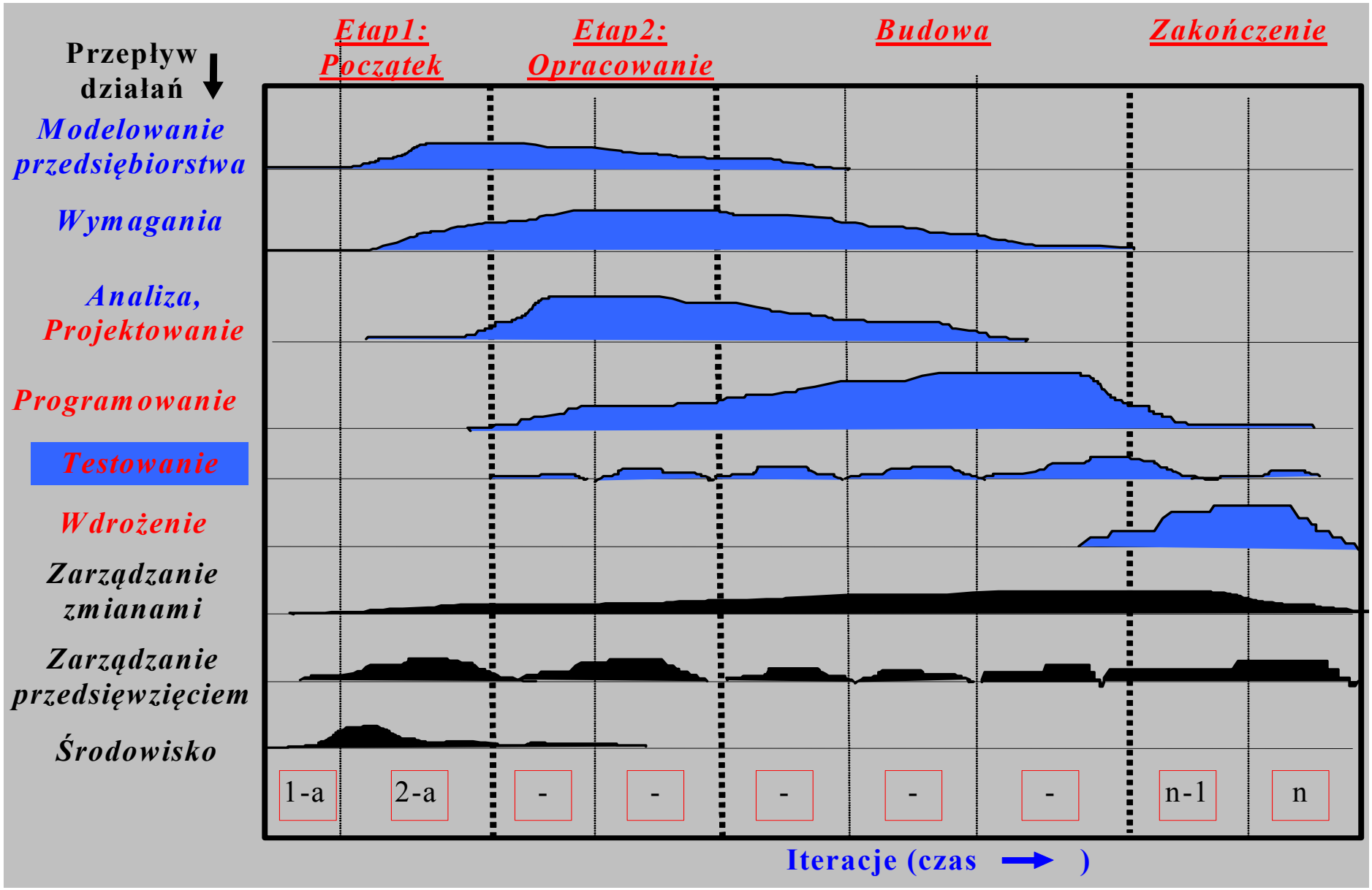
Model procesu wytwarzania oprogramowania - czyli model cyklu życia oprogramowania*

Tworzenie technicznego systemu informacyjnego jest powiązane z:

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

Modelowanie struktury i dynamiki systemu	**Implementacja systemu,**	**struktury i dynamiki generowanie kodu**
Perspektywa koncepcji *co należy wykonać?*	Perspektywa specyfikacji *jak należy używać?*	Perspektywa implementacji *jak należy wykonać?*
• model problemu np. Modelowanie przedsiębiorstwa • Wymagania • Analiza (model konceptualny) • Testowanie modelu	• Projektowanie (model projektowy: architektura sprzętu i oprogramowania; dostęp użytkownika; przechowywanie danych) • Testowanie projektu	• Programowanie (specyfikacja programu : deklaracje, definicje; dodatkowe struktury danych: struktury „pojemnikowe”, pliki, bazy danych) • Testowanie oprogramowania • Wdrożenie • Testowanie wdrażania

Zunifikowany iteracyjno- przyrostowy proces tworzenia oprogramowania – kiedy?



Przepływy działań (wg G.Booch, J. Rumbaugh, I.Jacobson)

- **Modelowanie przedsiębiorstwa** - opis dynamiki i struktury przedsiębiorstwa

---

- **Wymagania** - zapisanie wymagań stawianych programowi metodą opartą na przypadkach użycia
- **Analiza i projektowanie** - używanie różnych perspektyw architektonicznych
- **Implementacja** - tworzenie oprogramowania, scalanie systemu
- **Testowanie** - opisanie danych testowych, procedur i metryk poprawności, realizacja testów
- **Wdrożenie** - ustalenie konfiguracji gotowego systemu, usuwanie usterek
- **Zarządzanie zmianami** - panowanie nad zmianami i dbanie o spójność elementów systemu
- **Zarządzanie przedsięwzięciem** - opisane różnych strategii prowadzenia procesu iteracyjnego
- **Środowisko** – opisanie struktury niezbędnej do opracowania systemu

Wniosek

- W środowiskach niezwinnych projekty ulegają degradacji wskutek zmian wymagań jako konsekwencja zastosowania złych praktyk
- W praktykach zwinnych dzięki kolejnym iteracjom można eliminować negatywny wpływ zmian w wymaganiach

Struktura prezentacji

1. Symptomy złego projektu
2. Wprowadzenie do zarządzania projektowaniem oprogramowania metodą iteracyjno-rozwojową
3. **Zasady projektowania zwinnego - opartego na metodzie iteracyjno rozwojowej**

(1) Zasady projektowania zwinnego

- **Zasada pojedynczej odpowiedzialności (Single-Responsibility Principle - SRP)**

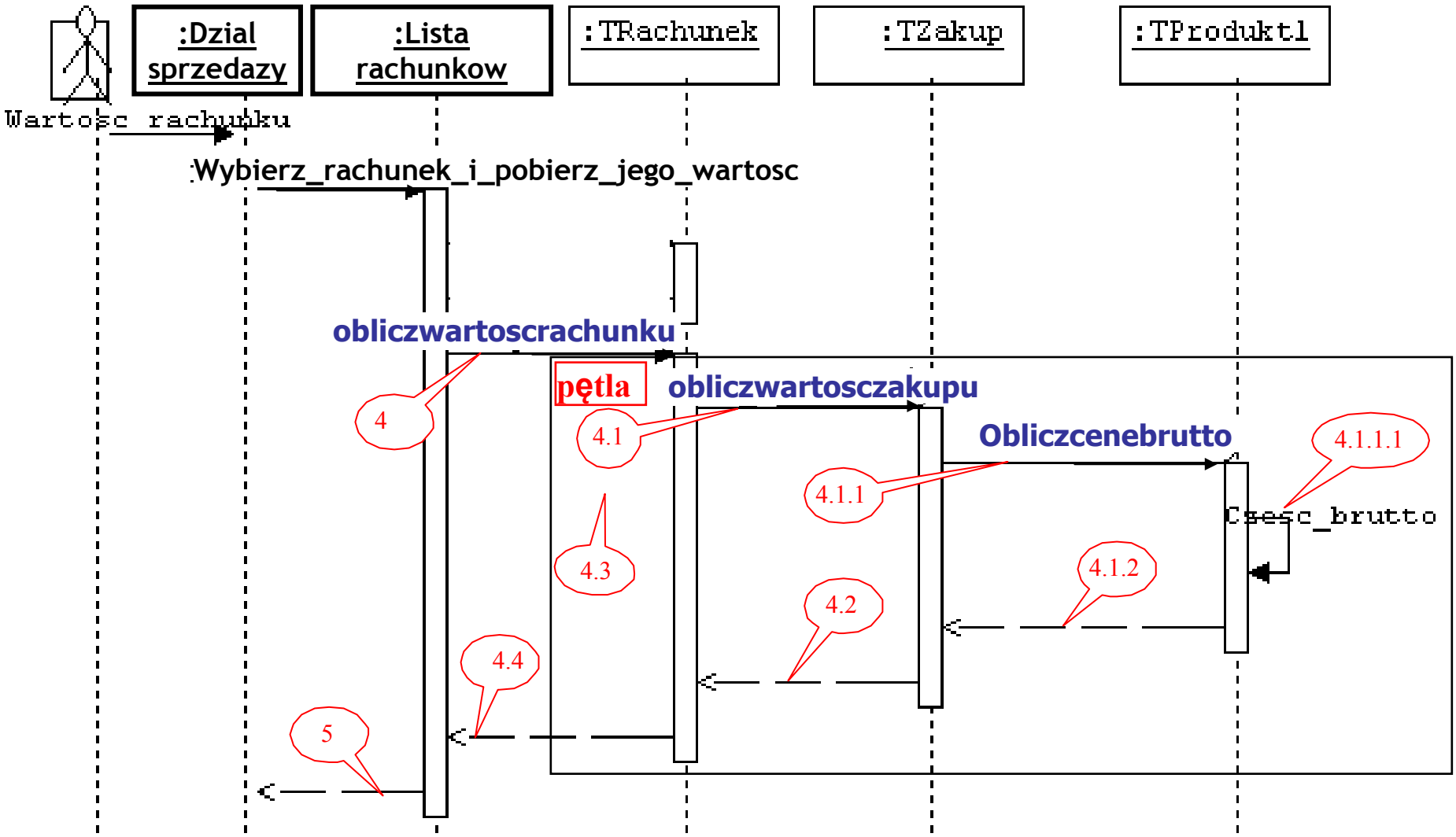
  Żadna klasa nie może być modyfikowana z więcej niż jednego powodu.

  1) Klasa obsługująca reguły biznesowe nie powinna zarządzać trwałością

  2) Oddzielanie wzajemnie powiązanych odpowiedzialności- np.. obiektowa idea rachunku:

  – Zmiana promowania ze względu na producenta produktu - tylko modyfikacja kodu produktów
  – Zmiana promowania ze względu na ilość zakupionego produtu - tylko modyfikacja kodu zakupów
  – Sposób wyznaczania wartości rachunku np. ze względu na grupy podatkowe – zmiana kodu rachunku

(z wykładu 1 (5)) Diagram sekwencji UML– obiektowy sposób przedstawienia scenariusza obliczania rachunku



(2) Zasady projektowania zwinnego

- **Zasada otwarte-zamknięte (Open/Closed Principle - OCP)**

  Składniki oprogramowania (klasy, moduły, funkcje itp.) powinny być otwarte na rozbudowę, ale zamknięte dla modyfikacji.

  1) Stosowanie dziedziczenia, polimorfizmu, implementacji interfejsów tylko w takich przypadkach, gdy istnieje możliwość zmian

  2) Obiektowa idea rachunku:

  – zmiana promowania ze względu na producenta produktu - tylko modyfikacja kodu produktów przez polimorfizm i dziedziczenie (TProdukt1, TProdukt2)

  – zmiana promowania ze względu na ilość zakupionego produtu - tylko modyfikacja kodu zakupów przez dziedziczenie i polimorfizm (TZakup1, TZakup2, itp.)

  – Sposób wyznaczania wartości rachunku np. ze względu na grupy podatkowe – zmiana kodu rachunku przy sumowania wartości zakupów przez dziedziczenie i polimorfizm (TRachunek1, TRachunek1…)

(3) Zasady projektowania zwinnego

- **Zasada podstawiania Liskov**

  Musi istnieć możliwość zastępowania typów bazowych ich podtypami.

  1) Klasa potomna nie może mieć mniejszej funkcjonalności niż jej klasa bazowa. Podstawianie klasy potomnej powinno następować automatycznie, bez potrzeby rozpoznawania typu obiektu

  2) Obiektowa idea rachunku:

  – Zakup obliczając cene nie musi znać typu produktu – zawsze otrzymuje cenę brutto (wynikającą z podatku lub/i promocji). Naruszeniem zasady byłoby stosowanie instrukcji if w celu rozpoznania typu produktu, aby pobrać różnie nazwane metody do obliczenia ceny brutto.

(4) Zasady projektowania zwinnego

- **Zasada odwracania zależności**

  A. Moduły wysokopoziomowe nie powinny zależeć od modułów niskopoziomowych. Obie grupy modułów powinny zależeć od abstrakcji

  B. Abstrakcje nie powinny zależeć od szczegółowych rozwiązań. To szczegółowe rozwiązania powinny zależeć od abstrakcji.

  Strategia programu nie powinna zależeć od szczegółowych rozwiązań w zakresie implementacji:
  1) Implementacja klasy JApplet
  2) Implementacja interfejsów jako słuchaczy zdarzeń w pakiecie Swing
  3) Implementacja obiektów typu wątki

(5) Zasady projektowania zwinnego

- **Zasada segregacji interfejsów (Interface Segregation Principle - ISP)**

  Klasa implementujaca nie powinna być zmuszana do zależności od metod, których nie używa.

  1) Separacja przez implementowanie wielu interfejsów
  2) Dziedziczenie wielobazowe

Manifest Zwinnego Tworzenia Oprogramowania
(http://agilemanifesto.org/iso/pl/)

Wytwarzając oprogramowanie i pomagając innym w tym zakresie, odkrywamy lepsze sposoby wykonywania tej pracy.
W wyniku tych doświadczeń 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 wymieniono po prawej stronie, jednak bardziej cenimy to, co po lewej.

Kent Beck Mike Beedle Arie van Bennekum Alistair Cockburn	Ward Cunningham Martin Fowler James Grenning Jim Highsmith	Andrew Hunt Ron Jeffries Jon Kern Brian Marick

Zasady kryjące się za Manifestem Zwinnego Wytwarzania Oprogramowania

http://agilemanifesto.org/iso/pl/principles.html

Kierujemy się następującymi zasadami:

1. *Najważniejsze dla nas jest zadowolenie Klienta wynikające z wcześnie rozpoczętego i ciągłego dostarczania wartościowego oprogramowania.*
2. *Bądź otwarty na zmieniające się wymagania nawet na zaawansowanym etapie projektu. Zwinne procesy wykorzystują zmiany dla uzyskania przewagi konkurencyjnej Klienta.*
3. *Często dostarczaj działające oprogramowanie od kilku tygodni do paru miesięcy, im krócej tym lepiej z preferencją krótszych terminów.*
4. *Współpraca między ludźmi biznesu i programistami musi odbywać się codziennie w trakcie trwania projektu.*
5. *Twórz projekty wokół zmotywowanych osób. Daj im środowisko i wsparcie, którego potrzebują i ufaj im, ze wykonają swoją pracę.*
6. *Najwydajniejszym i najskuteczniejszym sposobem przekazywania informacji do i w ramach zespołu jest rozmowa twarzą w twarz .*

7. *Podstawową i najważniejszą miarą postępu jest działające oprogramowanie.*
8. *Zwinne procesy tworzą środowisko do równomiernego rozwijania oprogramowania. Równomierne tempo powinno być nieustannie utrzymywane poprzez sponsorów, programistów oraz użytkowników.*
9. *Poprzez ciągłe skupienie na technicznej doskonałości i dobremu zaprojektowaniu oprogramowania zwiększa zwinność.*
10. *Prostota – sztuka maksymalizacji pracy niewykonanej – jest zasadnicza.*
11. *Najlepsze architektury, wymagania i projekty powstają w samoorganizujących się zespołach.*
12. *W regularnych odstępach czasu zespół zastanawia się jak poprawić swoją efektywność, dostosowuje lub zmienia swoje zachowanie.*

Agile Manifesto (wersja źródłowa)

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Struktura prezentacji

1. Symptomy złego projektu
2. Wprowadzenie do zarządzania projektowaniem oprogramowania metodą iteracyjno-rozwojową
3. Zasady projektowania zwinnego - opartego na metodzie iteracyjno rozwojowej
4. **Przykłady metodyk**

(1) Metodyka typu Agile

Scrum

- **sprint** (iteracja – (2-4 tygodnie),
- ***scrum meeting*** (spotkanie codziennie 15 min),
- ***sprint review*** *(podsumowanie sprintu)*)

Podstawowe zadanie metodyki:

- dostarczaniu kolejnych, coraz bardziej dopracowanych wyników projektu,
- włączaniu się przyszłych użytkowników w proces wytwórczy,
- samoorganizacji zespołu projektowego.

Role główne

- **Zespół** - 5-9 osób, zaangażowana w tworzenie oprogramowania
- **Właściciel produktu** (**Product owner**) - osoba reprezentująca klienta, która może należeć do zespołu
- **Kierujący zespołem (Scrum master)** – osoba ułatwiająca działania zespołu

(2) Metodyka typu Agile

Extreme Programming (XP)

Zalecenia:

- Iteracyjność
- Nie projektować z góry
- Testy jednostkowe (testy przed kodem)
- Ciągłe modyfikacje architektury
- Programowanie parami
- Stały kontakt z klientem

Kwestie kontrowersyjne

- Brak dokładnej specyfikacji.
- Stałe angażowanie strony klienta.
- Zbyt swobodne zmiany kodu

Aplikacje typu E-commerce

- **Business-to-customer (B2C)**
  Strona sklepu: strona internetowa, która jest udostępniana klientom, umożliwiając im zakup towarów za pośrednictwem Internetu. Dane z katalogu sklepu są zazwyczaj przechowywane w bazie danych, a strony udostępniające te dane są generowane dynamicznie.
  Konsola administracyjna: dostęp do funkcji aplikacji chroniony hasłem za pośrednictwem bezpiecznego połączenia. Używana przez personel sklepu do celów zarządzania typu online. Zazwyczaj dotyczy funkcji typu CRUD (tworzenie, odczyt, aktualizacja, usuwanie). Udostępnia: katalog sklepu, zarządzanie rabatami, transportem, formą płatności, oraz przeglądanie zamówień klientów.
- **Consumer-to-consumer (C2C):**
  Transakcje odbywają się między osobami, zwykle poprzez różne witryny, jak w przypadku aukcji internetowych.Typowym przykładem C2C jest system eBay.
- **Business-to-business (B2B):** handel występujący między przedsiębiorstwami, np. między sprzedawcą i hurtownią lub między hurtownią i producentem.
- **Business-to-government (B2G):** handel występujący między przedsiębiorstwami i agencjami rządowymi.