System zaprojektowany w sposób obiektowy, to zbiór obiektów współpracujących ze sobą. W wyniku współpracy tych obiektów system realizuje pewne nałożone mu zadania.

Obiekt - nie ma negat co ma być obiektem, obiekt - warty uwagi, podmiot z dziedziny problemu, w ilości pojedynczej

Metoda - zbiór pewnych rad i wskazówek, z których jeśli będziemy korzystać, to mamy pewne szanse, że uda nam się projekt zrobić, nie mamy jednak na to żadnej gwarancji. Warto się stosować do tych wskazówek, ale to czy uda się zaprojektować system nie zależy od metody, tylko od ludzi pracujących nad projektem.

Jeżeli projektujemy system w sposób obiektowy musimy zacząć od identyfikacji obiektu

System bankowy (ważne nazwanie w dziedzinie) - konto, klient, kasjer, pożyczka

Po zidentyfikowaniu obiektów, należy się zastanowić nad identyfikacją objektów. Należy zastanowić się nad interakcjami między tymi objektami, nad scenariu-

System zaprojektowany w sposób obiektowy, to zbiór obiektów współpracujących ze sobą. W wyniku współpracy tych obiektów system realizuje pewne nałożone mu zadania.

Obiekt - nie ma negat co ma być obiektem, obiekt - warimy nazwanie, podmiot z dziedziny problemu, w liczbie pojedynczej

Metoda - zbiór pewnych rad i wskazówek, z których jeśli będziemy korzystać, to mamy pewne szanse, że uda nam się projekt zrobić, nie mamy jednak na to żadnej gwarancji. Warto się stosować do tych wskazówek, ale to czy uda się zaprojektować system nie zależy od metody, tylko od ludzi pracujących nad projektem.

Jeżeli projektujemy system w sposób obiektowy musimy zacząć od identyfikacji obiektów

System bankowy (ważne nazwanie w dziedzinie) - konto, klient, kasjer, pożyczka

Po zidentyfikowaniu obiektów, należy się zastanowić nad identyfikacją objektów. Należy zastanowić się nad interakcjami między tymi objektami, nad scenariu-

sami współpracy tych obiektów. Co dany obiekt powinien udostępniać innym obiektom. Uzyskamy pewne usługi, interfejsy, które pewien obiekt realizuje, a więc trzeba będzie je zdefiniować. Część usług będzie wykonywanych na żądanie, część na potrzeby obiektu.

Typy obiektów:

- pasywne - potrzebne po to, aby przechowywać jakieś informacje np. konto (przechowujemy informacje o stanie konta)
- aktywne - np. kasjer, pracownik czy klient, działają na innych obiektach.
- odzwierciedlają obiekty istniejące w świecie rzeczywistym np. system do prognozy pogody, czujniki będą odzwierciedlały rzeczywiście istniejące wskaźniki pomiaru
- obiekty wymyślone przez projektanta
- obiekty działające cały czas podczas funkcjonowania systemu, zostaną utworzone przy starcie systemu
- obiekty chwilowe, potrzebne przez chwilę, po czym zostają usuwane

- obiekty prywatne - obiekty wewnętrzne jakiś obiektów
- obiekty publiczne - mają charakter ogólnie dostępny

- obiekty typu całość - samochód
- obiekty typu części - silnik, skrzynia biegów

Organizacja obiektów:

Zastanawiamy się nad cechami wspólnymi i szczególnymi obiektów: np. obiekt osoba może zawierać w sobie obiekty klient, kasjer, które mają wspólny pesel, nip, nazwisko, imię, adres ...

Obiekty mogą dziedziczyć cechy innych obiektów, czyli np: klient może odziedziczyć po obiekcie kasjer takie cechy jak nip, pesel, data wodrenia, adres ...

Obiekty w systemie współpracują ze sobą, ale nie każdy z każdym, tylko współpracują ze sobą w takich mniejszych grupach. Należy się zastanowić które obiekty mają ze sobą współpracować.

Obiekty są zależne jeden od drugiego, wtedy gdy np. obiekt A tworzy obiekt B, to obiekt B zależy od obiektu A

Relacje należy zaznaczać oczywiście do tego pewnych symboli graficznych.

Scenariusze - przykłady użycia systemu, które obiekty muszą ze sobą współpracować, żeby system działał.

Operacje obiektu - usługi świadczone przez jeden obiekt na rzecz innego obiektu, wykonywane na żądanie innego obiektu lub realizowane wewnętrznie przez obiekt.

Implementacja operacji które obiekty muszą realizować.

Standard w metodach - inżynieria oprogramowania wspomagana komputerowo, CASE. Stworzono zunifikowany język modelowania obiektowego, który został zatwierdzony w 1997 r. jako standard UML - zunifikowany język modelowania. UML wersja 1.5. wydał szybko inne metody obiektowe, bo może być stosowany w różnych systemach

UML - standard modelowania, język, który do wizualizacji projektu, za jego pomocą specyfikujemy projekt, konstruujemy projekt,

pozwala wygenerować dokumentację projektu.

UML - jest językiem graficznym, znaczenie poszczególnych symboli jest bardzo ściśle, pozwala na konstrukcję języków programowania, generuje pełne szkielety kodów.

Projektowanie systemów z wykorzystaniem notacji UML-owej - poziomy widzenia systemu; podczas projektowania systemu

1. Widzenie systemu poprzez USE CASE, symbolem use case jest elipsa ⬭, jest to przykład widzenia systemu, funkcji jakie ten system oferuje, w jaki sposób można ten system można wizualizować. Tworzymy diagramy use case, które mówią co system robi i kto pracuje w tym systemie
2. ~~System~~ Poziom logiczny - jak system realizuje funkcje, co jest potrzebne żeby system mógł działać, co znajduje się w systemie, tworzymy różne typy diagramów, mówimy jakie obiekty są w systemie, jak ze sobą współpracują.
3. Poziom implementacyjny - określamy z czego system korzysta, określamy architektonicznie system, co jest potrzebne

aby system mógł pracować.

(4) Poziom fizyczny - jak oprogramowanie jest rozmieszczone, gdzie są zainstalowane poszczególne komponenty, jak to jest rozmieszczone.

Model USE CASE - widzenie systemu poprzez przypadki użycia. Musimy widzieć system z punktu widzenia różnych użytkowników. Kto i co może pracować w systemie oraz co może zrobić jako usługę może wywołać. Określamy tu różne możliwości.

Diagramy USE CASE - pokazują pełne funkcje i usługi systemu.

Elementy diagramu use case:

Aktor - element zewnętrzny, kto lub co może w danym systemie pracować, może to być użytkownik, administrator, systemy wewnętrzne, zewnętrzne, można organizować aktorów, symbolem aktora jest ludzik 

Szukając use case szukamy czasowników, które określają czynności

przypadek użycia musi być przez nas nazwany, tak abyśmy wiedzieli jakąś funkcję on pełni

Diagram use case - łączymy aktora z use casem, natura use casa powinna określać czynność, która w tym systemie można temu aktorowi wykonać.

Linia ciągła (asocjacja, powiązanie)

- nieskierowana - asocjacja dwukierunkowa, pewne informacje mogą być przesyłane w jedną i w drugą stronę, aktor może wykonać funkcję systemu i może do niego dotrzeć informacja zwrotna, np. czy wywołanie funkcji systemu się powiodło.

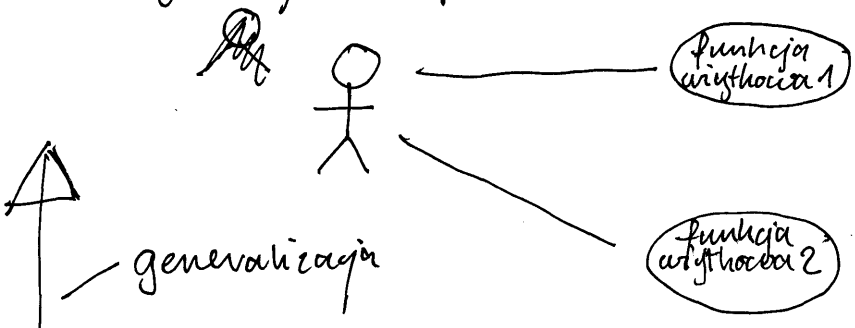
99,999) % przypadków, kiedy łączymy aktora z use casem to połączenia dwukierunkowe, pomiędzy jeśli użytkownik wykonuje jakąś funkcję systemu, to chce wiedzieć czy się ona powiodła

- skierowana - asocjacja jednokierunkowa → gwiazdka oznacza, w którą

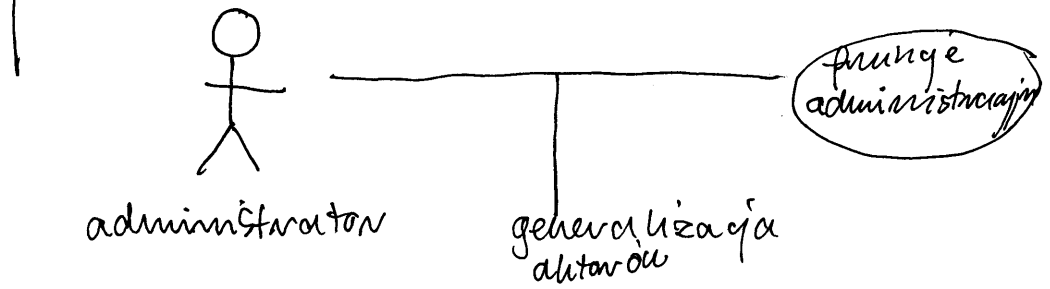
strony można przesłać informację, ale nie otrzymujemy informacji zwrotnej.

43:07 Aby zmienić asocjację ~~dwukierunkową~~ jednokierunkową na dwukierunkową należy zmienić nawigację zaznaczając nawigable.

Aktorów można organizować. Mamy fragment systemu, w którym jest wirtualnik mający uprawnienia do wirtualizacji systemu, może korzystać z funkcji wirtualnej 1 i funkcji wirtualnej 2, powiązany z nimi asocjacją dwukierunkową



W systemie mamy również administratora, który również może wykonywać funkcje

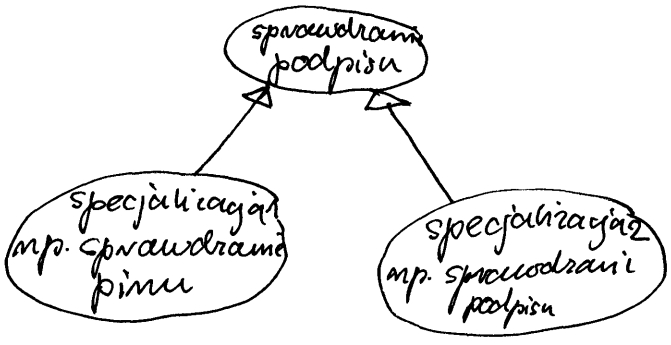


Administrator dziedziczy funkcje, które może realizować zwykły wirtualnik.

Diagram use case musi być bardzo czytelny. Uszczegółowienie
Dziedziczenie pokazujemy rysując strzałkę
oznacza ona generalizację, administrator
jest pewną specjalizacją zwykłego użytkownika.

Można również organizować use case na
3 sposoby:

1. Generalizacją - pewien use case jest ogólny,
a inny stanowi jego specjalizacje.



Generalizacja jest dozwolona, ale rzadko
stosowana

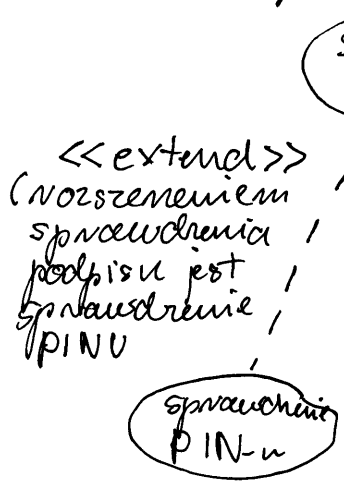
2. <<include>> - włączanie

<< >> - steretypy UML, pewien zbiór steretypów
jest wbudowany w UML-u, inne można tworzyć
samemu

<<include>> - jeden USE CASE jest włączany
w inny.

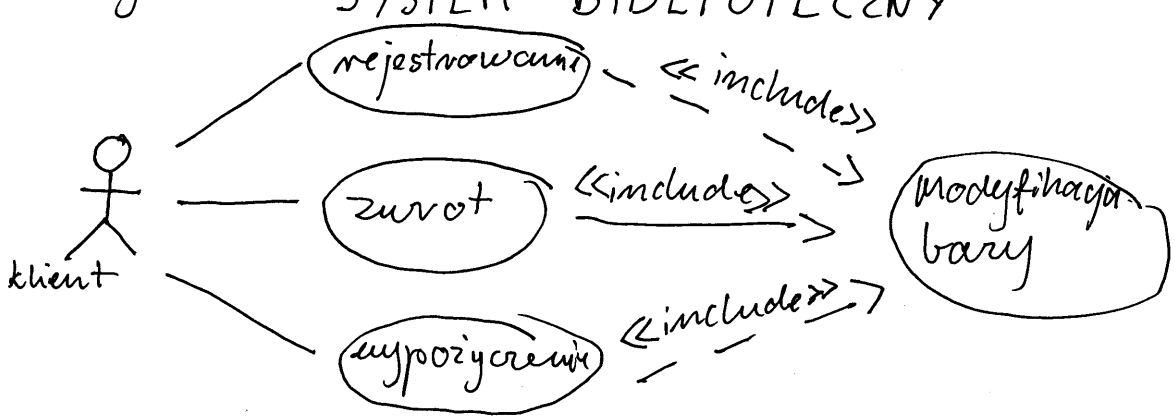
3. <<extend>> - rozszerzanie właściwości jednego
USE CASE za pomocą innego

Za pomocą stereotypów <<include>>, <<extend>> korzysta się z linii przerywanych - - - - -, nazywanych liniami zależności, grot linii pokazuje kierunek zależności, jeśli nie ma grotu to jest to zależność dwukierunkowa.



Jeśli chcemy rozszerzyć działanie pewnych opcji USE CASE albo pokazać zależności to należy użyć stereotypu <<extend>>.

Stereotypu <<include>> używamy, jeżeli chcemy pokazać, że jakieś funkcje muszą być używane



Diagramy USE CASE pokazuje co można danemu aktorowi zrobić, ale nie pokazuje w jakiej kolejności może on to zrobić.

Kolejność realizujemy implementując USE CASE. Na diagramie nie da się tego zrobić

<< include >> - włącza coś do bazy, ustala przebieg, wiązań USE CASE w modyfikacjach, wyodrębniamy wspólny przebieg dla poszczególnych USE CASE

Kiedyś zależności pokazuje co jest rozszerzane przy użyciu stereotypu << extend >> lub co jest włączane przy użyciu stereotypu << include >>

Aktorem może być system zewnętrzny np. sieć telekomunikacyjna, która może wymagać systemu np. poprzez odebranie połączenia oraz jest niezbędna do realizacji połączenia. Czasami aktor jest niezbędny do realizacji USE CASE

Opisywanie USE CASE poprzez <u>sekwencje zdarzeń</u>, która realizuje tego USE CASE, tworzymy scenariusz zdarzeń.

Sekwencja zdarzeń na przykładzie podjęcia pieniędzy z systemu bankomatowego.

- znak zachęty na bankomacie
- klient podaje PIN
- klient potwierdza
- system sprawdza poprawność