



Návrh a realizace softwarových systémů

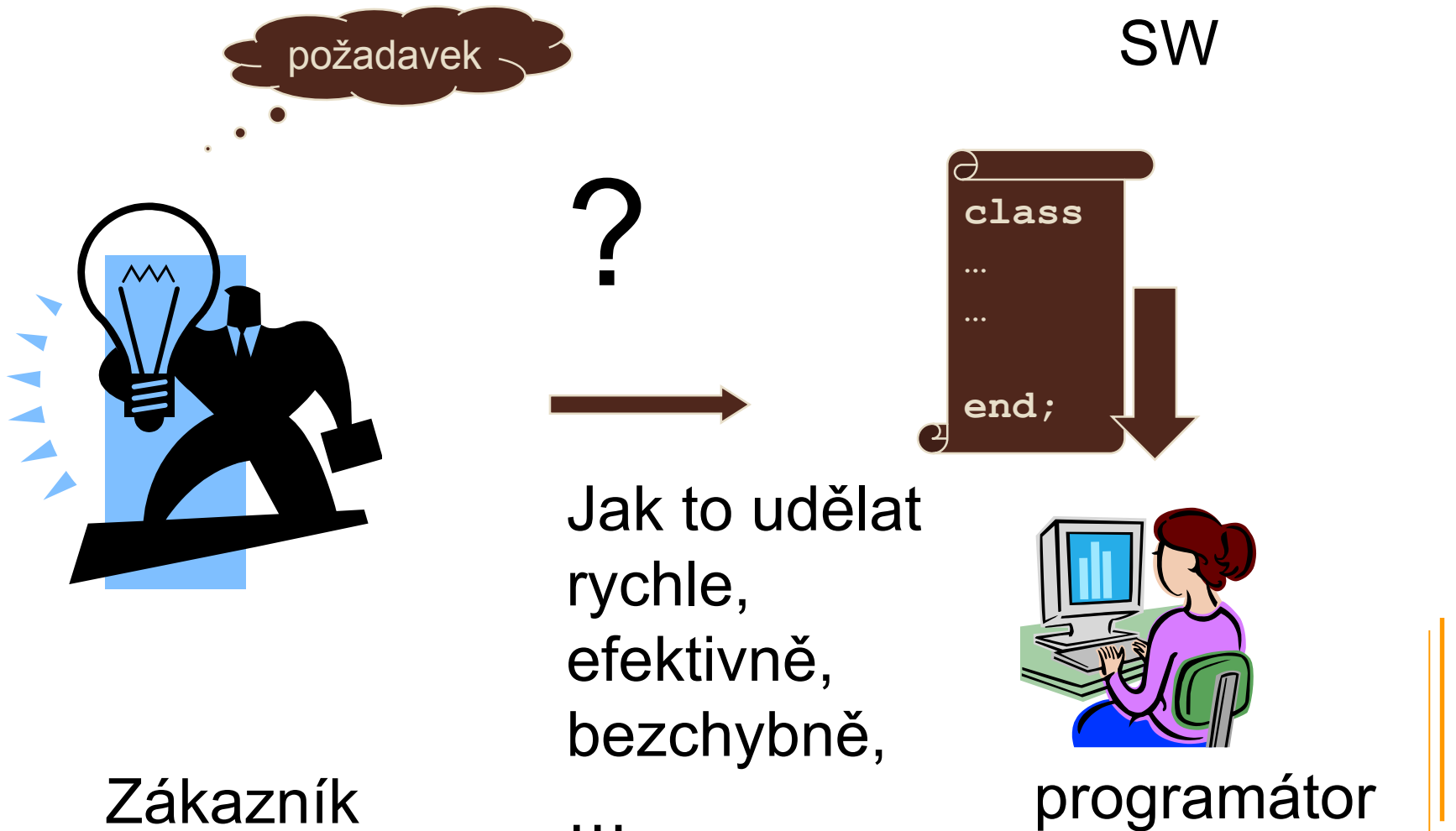
Projektování programových systémů

Jan Kelnar

Smysl a cíle předmětu

- Předmět by měl navázat na znalosti analýzy, UML modelování a věnovat se návrhu (nejen) architektury softwarové aplikace jako takové
- Vyzkoušení si vývoje reálné LOB aplikace ve výukovém prostředí
- Příprava na roli technického vedoucího softwarového týmu
- Ukončení předmětu:
 1. Projekt – 100 bodů, potřeba získat alespoň 75 bodů.(více jak 90 bodů – 5 bodů bonus ke zkoušce, více jak 95 bodů – 10 bodů ke zkoušce)
 2. Písemná zkouška – alespoň 25 bodů z 50 možných

Hlavní problém tvorby software



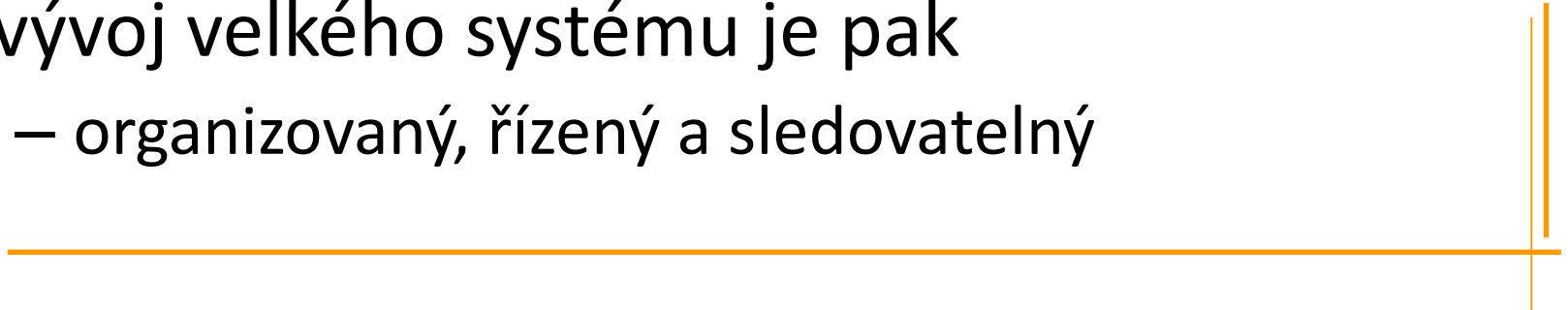
A co praxe?

- Hra E.T. – rok 1982
- Atari 2600
 - procesor 1.19Mhz,
 - 128 bajtů RAM,
 - cartridge 4kB ROM
 - 5 týdnů na celý projekt – sebevražda – nikdy neberte takový projekt, i kdyby Vám nabídli vilu v Karibiku a doživotní rentu
 - Ztráta 536 miliónů dolarů
 - "novinky"
- Ariane 5, let číslo 501 (1996)
 - starší verze SW-modulu byla použita bez dostatečného testování
 - Při odlišné dráze došlo k přetečení z 64-bitového double na 16-bitový integer se znaménkem
 - 370 milionů dolarů



Metodiky vývoje software

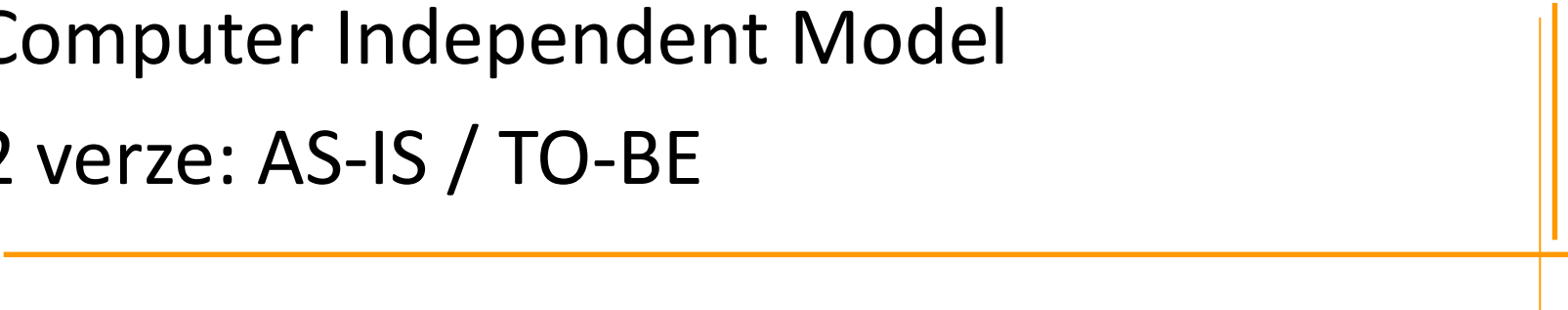
- doporučený postup při přechodu od mentálního modelu k řešení
- člení postupu na kroky
 - výstupem jednotlivých kroků - vybrané artefakty (model, komponenta, zdroj.kód)
- vývoj velkého systému je pak
 - organizovaný, řízený a sledovatelný



Metodiky

- Většina klasických metodik obsahuje pracovní aktivity:
 - Modelování organizace (Business Modeling)
 - Sběr požadavků (Requirements)
 - Analýza (Analysis)
 - Návrh (Design)
 - Implementace (Implementation)
 - Testování (Testing)
 - Nasazení (Deployment)
-

Business Modelování

- Pochopit fungování organizace
 - Porozumět pojmům dané domény
 - Modelují se
 - Procesy – dynamický model (Business Process Model/ Business Use Case Model)
 - Pojmy – statický model (Business Object Model)
 - NEjedná se o model budoucího systému
 - Computer Independent Model
 - 2 verze: AS-IS / TO-BE
- 

BPM

Modelování business procesů

- Strukturovaný text
- UML diagramy aktivit
- Erikssonovy-Penkerovy Diagramy
- BPMN



BOM

- Modelování Business Objektů (Pojmy dané domény)
 - Používají se UML Class Diagramy
 - Budoucí analytický Class Diagram je podmnožinou tohoto



Sběr požadavků

- získat od zákazníka požadavky na budoucí systém
- hlavní činnost – interview se zákazníkem a budoucími uživateli systému
- vytvářené artefakty mají většinou podobu strukturovaného textu – musí být srozumitelný zákazníkovi
- Katalog požadavků (funkční/nefunkční)
- Use Case Model



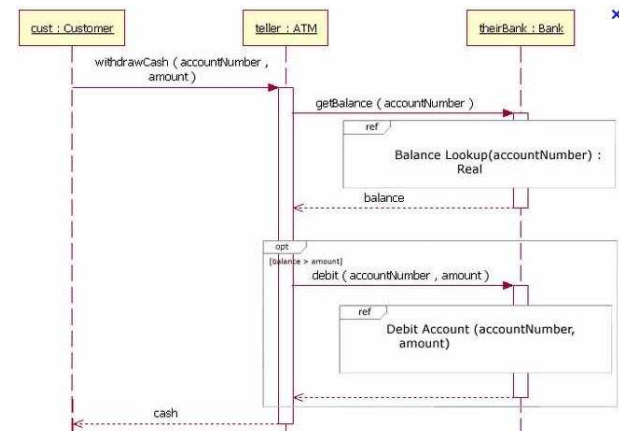
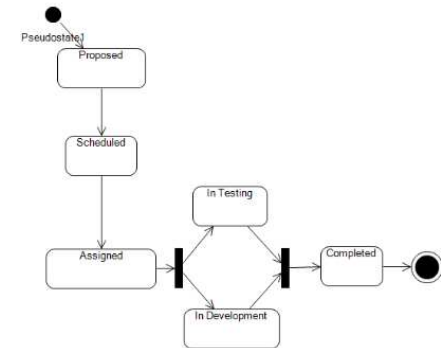
Analýza

- Zkoumání problému (nikoliv hledání způsobu jeho řešení)
- Zpracování a formalizace požadavků
- Vyjasnění pojmů, souvislostí
- Platformě nezávislý model systému (Platform Independent Model)
- Analýza je nezávislá na technologii !!!



Analýza

- Platform Independent Model
 - Class Diagram + Package Diagram
 - State machines
 - Průchod Use Cases – Activity diagrams
 - Realizace Use Caseů – sequence diagrams



Návrh

- Hledá řešení **JAK** splnit požadavky definované v analýze
- Zvolit vhodné technologie a architekturu
- Převod analytického modelu do konkrétní technologie
- Cílem nalézt vhodné softwarové objekty a ukázat, jak vzájemnou spoluprací realizují požadavky
- Návrh je již platformě specifický (Platform Specific Model)



Návrh

- Platform Specific Model
 - Class Diagram pro určitou technologii
 - Datový model (databáze)
 - Package diagram (logická architektura)
 - Sequence diagrams (pro konkrétní technologii)
 - Návrh GUI

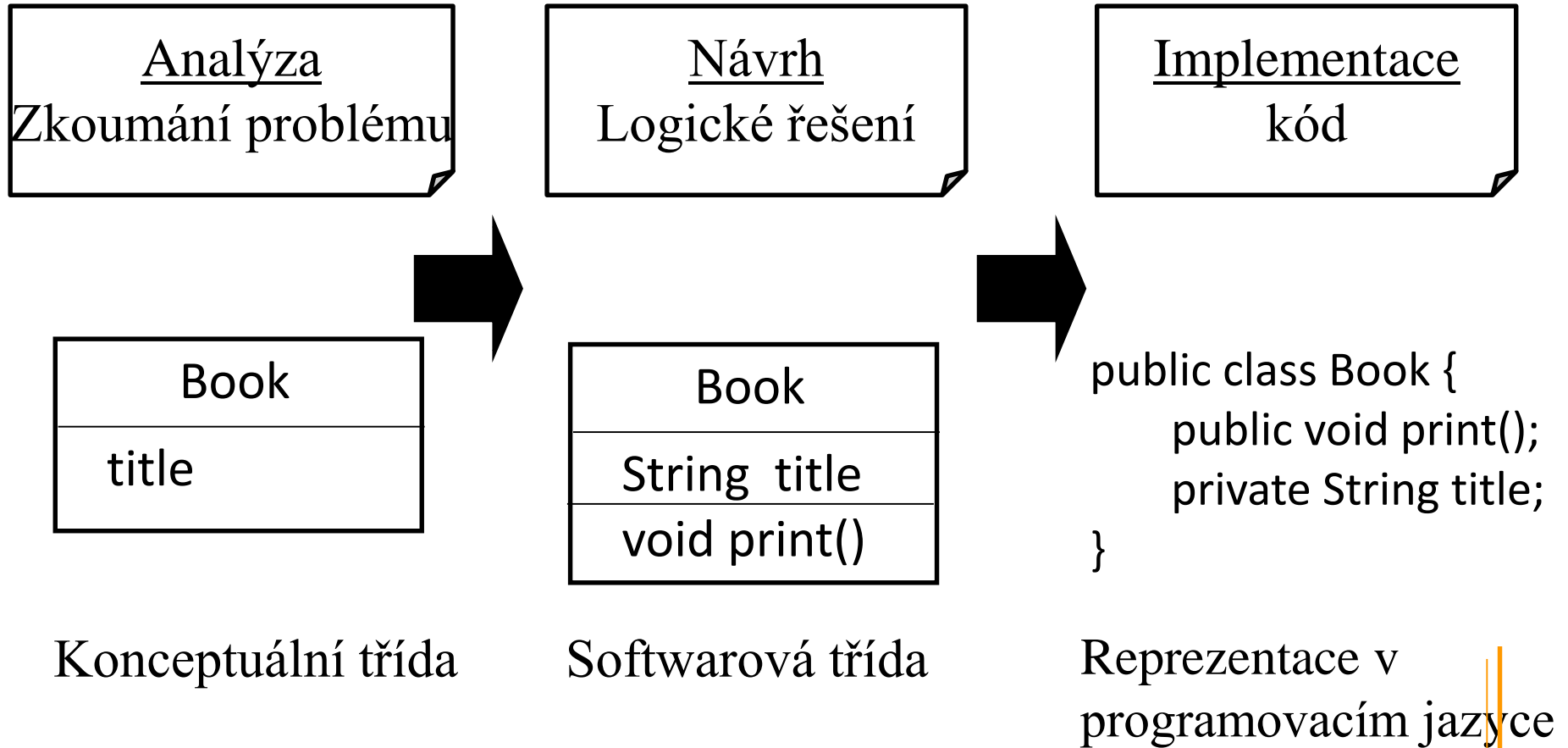


Implementace

- Programová realizace řešení navrženého návrhu
- Vygenerování základu kódu z návrhových tříd (MDA)
- Dopsání funkcionality popsané v návrhu



Od analýzy k implementaci



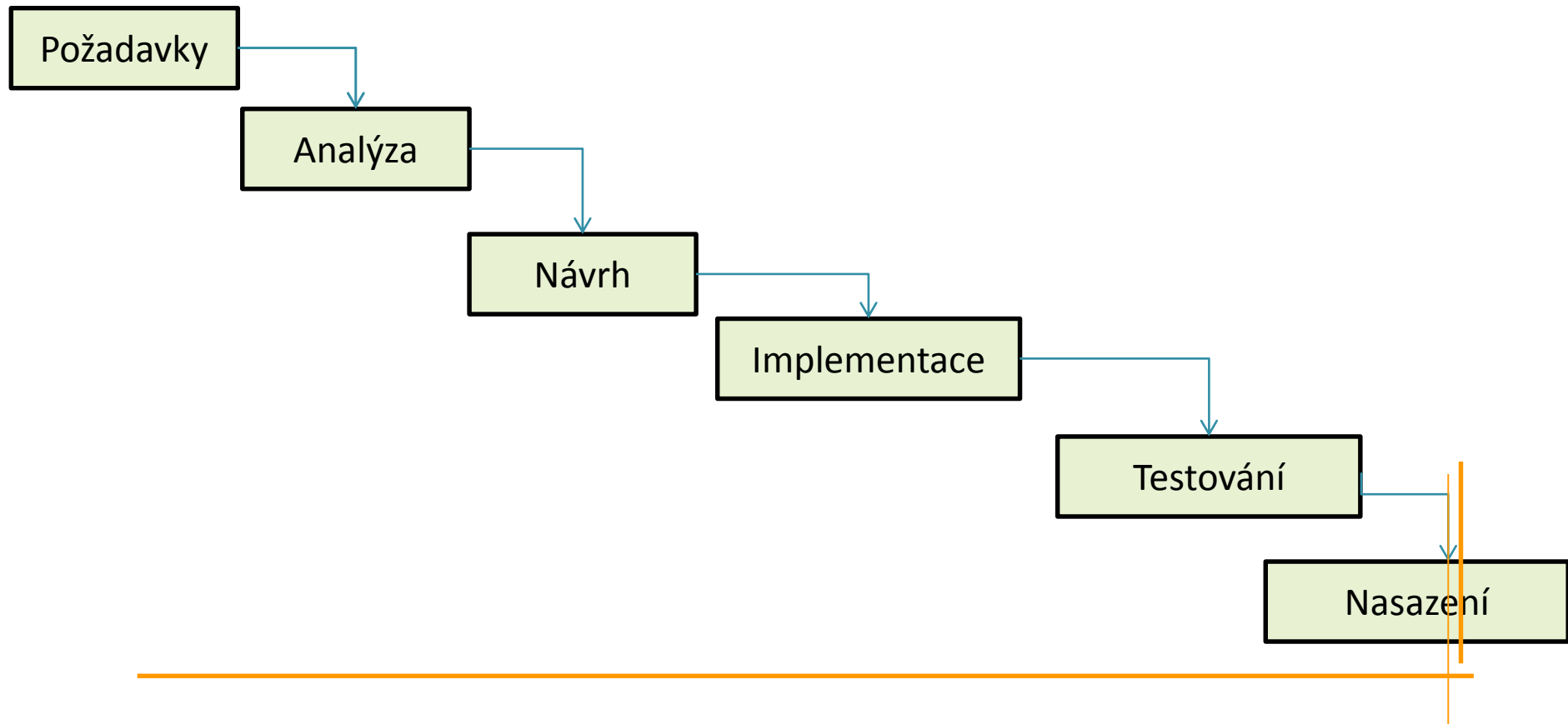
Styly vývoje

- Vodopád
- Klasické iterativní metodiky (RUP-like)
- Agilní metodiky



Vodopád (waterfall)

- fáze následují za sebou v kaskádě
 - další fáze začne tehdy, až předchozí skončí

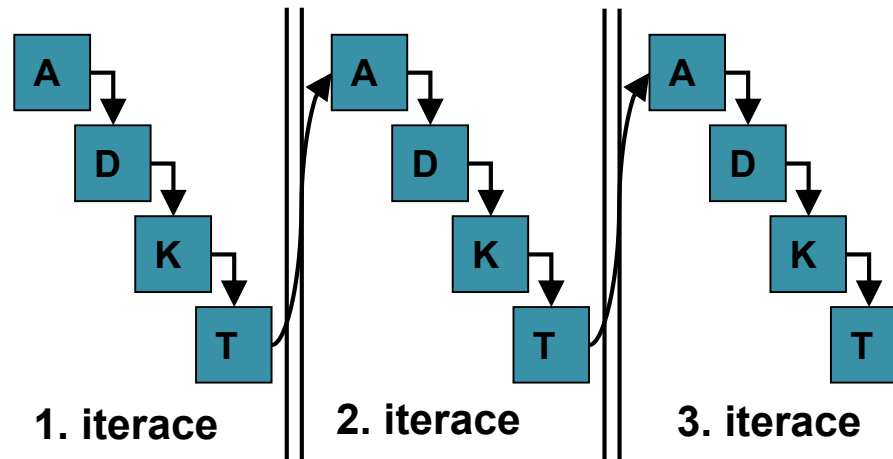


- Pro jednoduché systémy by bylo prověřitelné sekvenční (tzv. vodopádové) řešení
- Pro složitější systémy je vodopádový postup nerealistický (mnoho neúspěšných projektů jako důkaz)



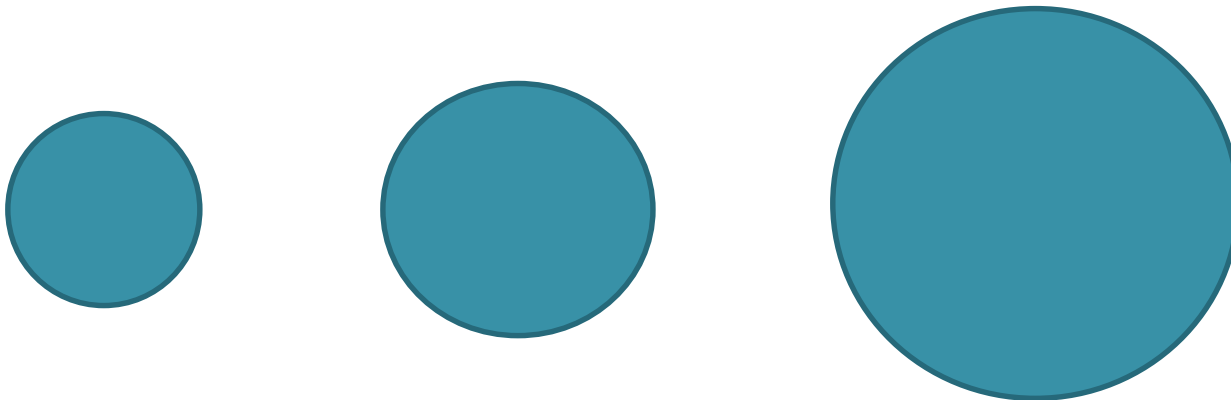
Iterativní vývoj

- Vývoj je organizován do krátkých mini projektů s pevnou dobou trvání zvaných **iterace**.
- Každá iterace představuje kompletní vývojový cyklus, obsahuje vlastní: sběr požadavků, analýzu, návrh, implementaci a testování.



Iterativní vývoj

- Iterativní vývojový proces je založen na postupném rozšiřování a vylepšování systému násobnými iteracemi spolu se zpětnou vazbou a adaptací.
- Každá iterace zpracovává nové požadavky a inkrementálně rozšiřuje systém.
- Systém roste přírůstkově iteraci za iterací.



Iterativní vývoj

- Zákazníci často mění své požadavky.
- Díky iteracím je rychlá zpětná vazba a šance zabudovat připomínky uživatelů při dalších iteracích.
- Iterace může příležitostně přepracovávat již hotovou část a vylepšit ji (zabudování připomínek od klientů).



Iterativní vývoj

- Výstup iterace **není** experimentální prototyp ale funkční část výsledného systému.
- Výstup každé iterace je testován a zintegrován – výsledkem je spustitelný systém.



Iterativní vývoj

- doporučují se krátké iterace pro dosáhnutí rychlé zpětné vazby a možnosti rychlého zapracování připomínek (UP doporučuje iterace o délce 2-6 týdnů.)
- Dlouhé iterace zvyšují riziko projektu.
- Iterace mají pevnou dobu trvání (**timeboxed**). Pokud by hrozilo nestihnutí v termínu, některé požadavky se z iterace odeberou a přesunou do další iterace.

Výhody iterativního vývoje

- Minimalizovat rizika
 - Rizika jsou identifikována včas, je dobře vidět, jak se systém vyvíjí.
- Dosáhnout robustní architektury
 - Architektura může být nastavena a vylepšena včas
- Zpracovávat vyvíjející se požadavky
- Uživatelé se mohou seznamovat se systémem již v průběhu

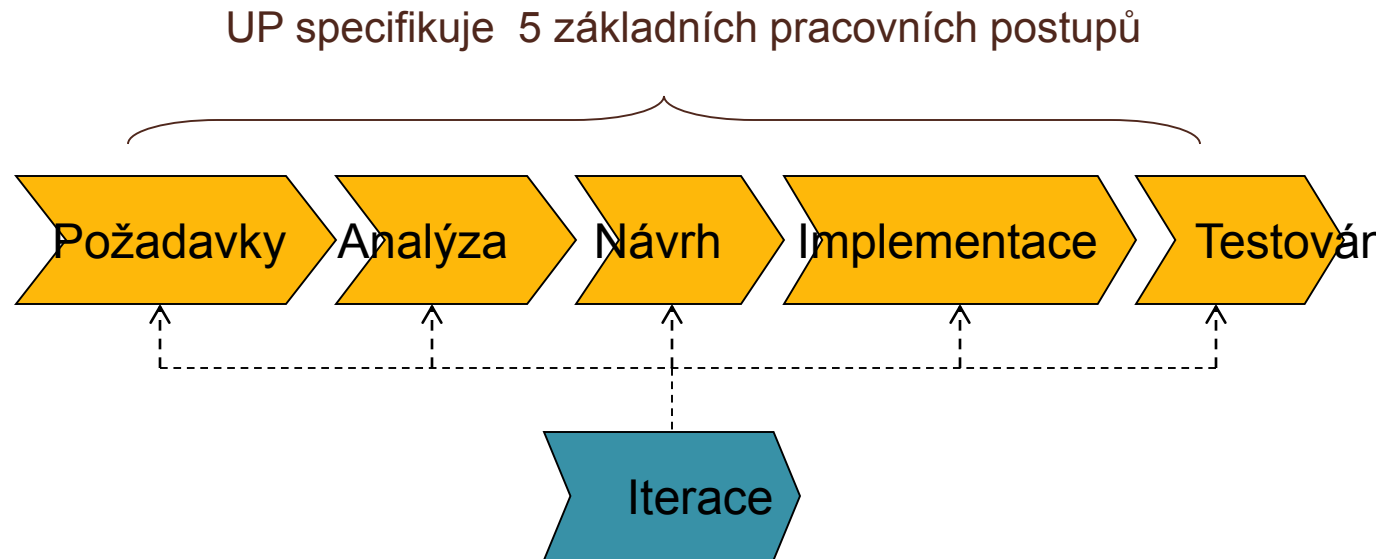


Metodika Unified Process

- Popsána v knize Jacobsona: *Unified Software Development Process* (1999).
- paralelismus
- Vychází z metodiky RUP (Rational Unified Process), ale představuje otevřený standard.
- Použití UML
- Metodika UP představuje principy, které je nutno upravit – je nutno počítat s tím, že přizpůsobení zabere určitý čas a bude vyžadovat určité prostředky (nástroje, šablony, standardy, modifikace živ. cyklu,...).
- Zaměření na zákazníka, průběžné řízení změn, plánování jednotlivých fází, používání komponent a vytváření znovupoužitelného kódu
- Pro každou organizaci i pro každý jednotlivý projekt je nutno vytvořit novou instanci.

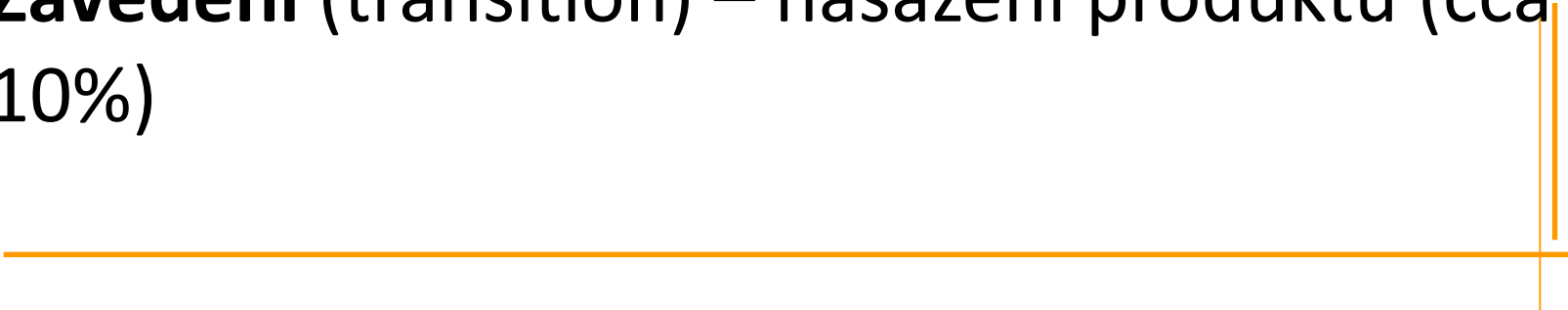
Pracovní postupy v iteraci

- Každá iterace **může** obsahovat všech 5 základních pracovních postupů, přičemž objem práce v jednotlivých se liší dle fáze

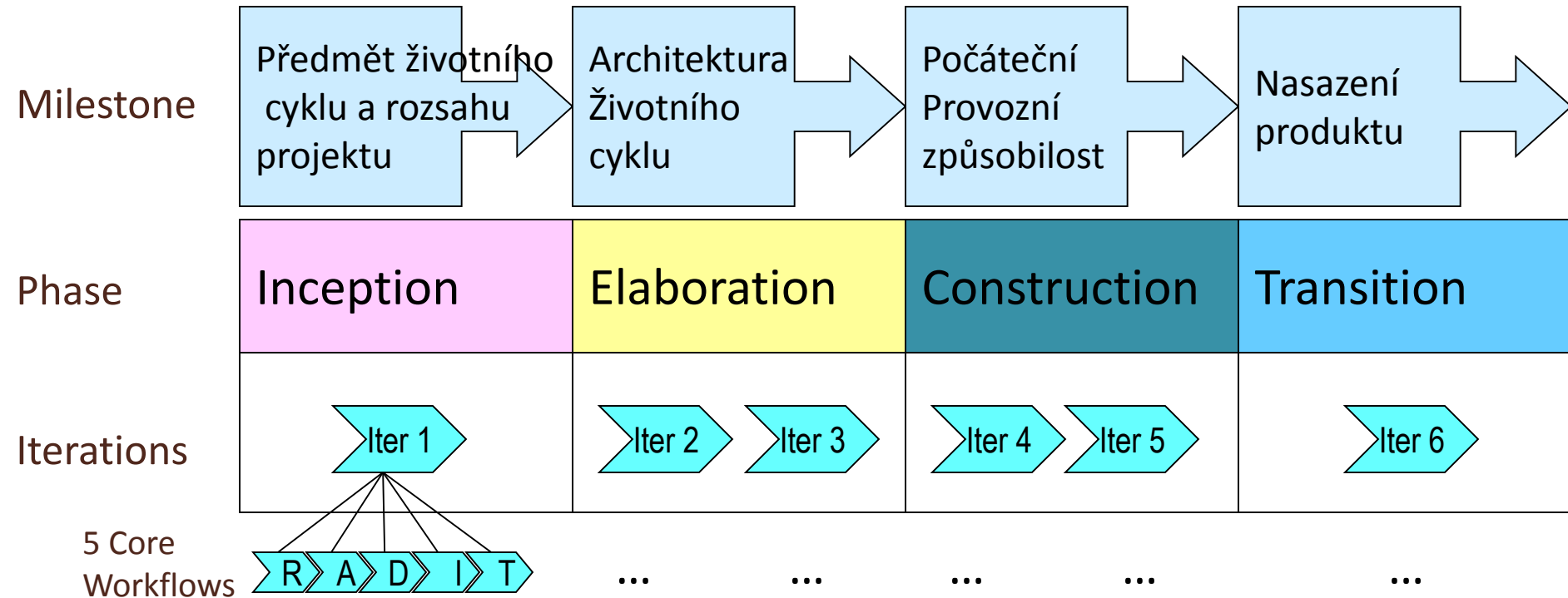


+plánování iterace

Struktura UP

- **Zahájení** (inception) – Rozsah projektu, business case (cca 10%), požadavky
 - **Rozpracování** (elaboration) – Požadavky, architektura (cca 30%), analytici
 - **Konstrukce** (construction) – tvorba (cca 50%)
– zde jsou iterace
 - **Zavedení** (transition) – nasazení produktu (cca 10%)
- 

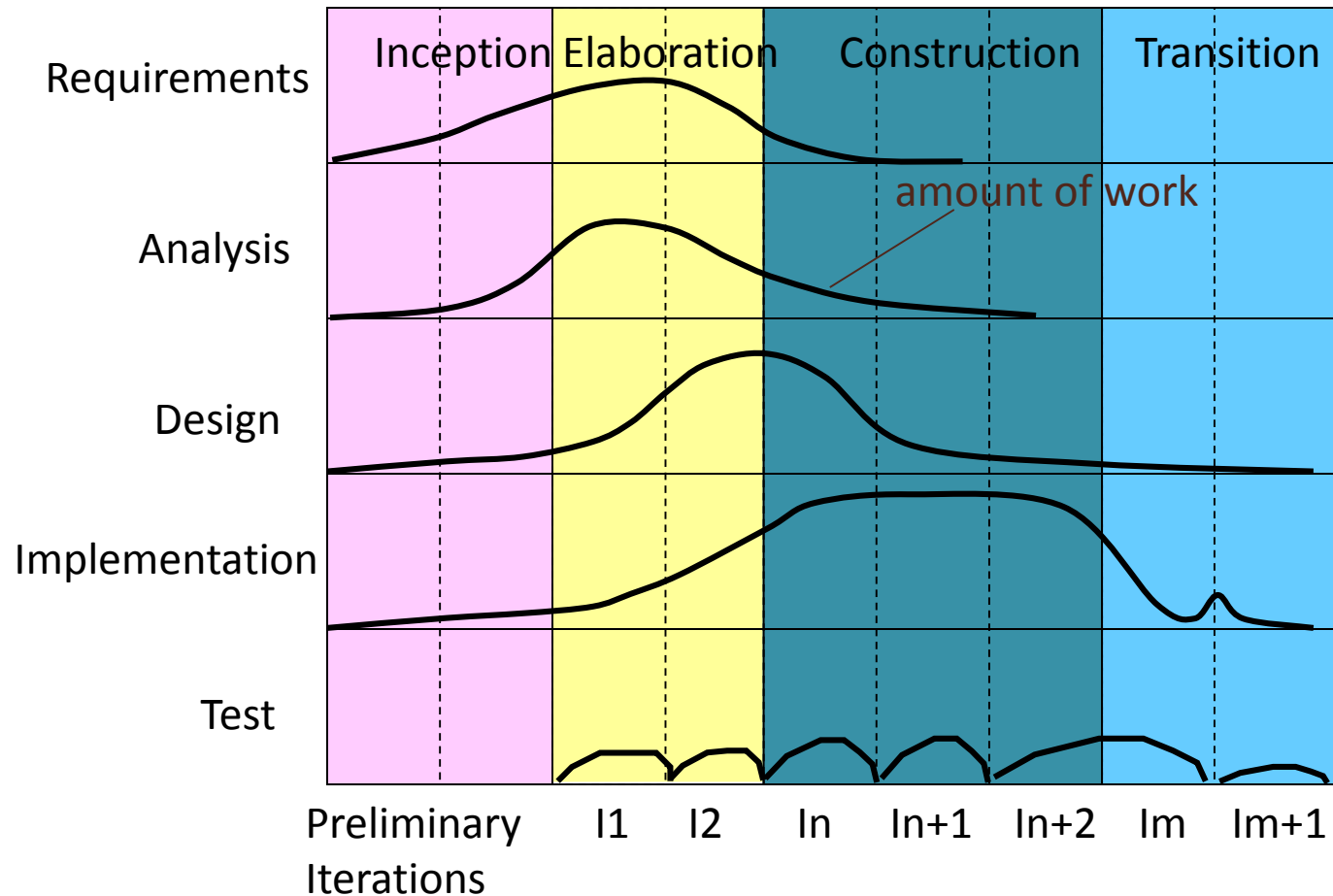
Struktura UP



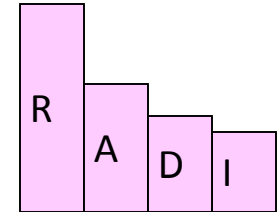
- Přesný počet iterací je daný velikostí projektu

Fáze a pracovní postupy

- Pro každou fázi sledujeme:
 - Jaké prac.post. jsou klíčové
 - cíl fáze
 - milník na konci fáze

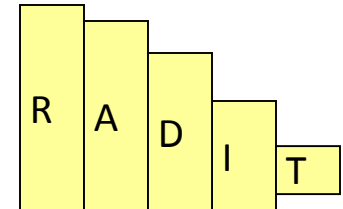


Zahájení



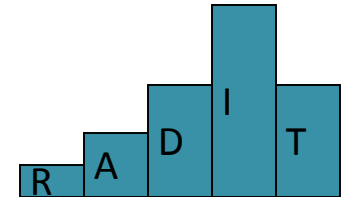
Focus	Requirements – určení klíčových požadavků	Inception	Elaboration	Construction	Transition
	Analysis – posouzení proveditelnosti				
	Design – navrhnout prototyp				
	Implementation – implementovat prototyp				
	Test – nic				
Goals	Posouzení proveditelnosti projektu –vytvoření prototypu Nadnesení obchodního případu – ukázat, že projekt bude mít přínos Zachycení kritických požadavků, které umožní určit rozsah projektu Označení kritických rizik				

Rozpracování



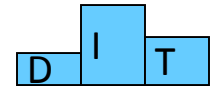
Focus	Requirements – upřesnění rozsahu systému	Inception	Elaboration	Construction	Transition
	Analysis – stanovení toho, co budeme tvořit				
	Design – návrh stabilní architektury				
	Implementation – tvorba stabilního architekt. základu				
	Test – testování stabilního architekt. základu				
Goals	Tvorba spustitelného architektonického základu Zpřesnění odhadu rizik Zachycení až 80% Use Case Vytvoření detailního plánu fáze Konstrukce Formulace přesné nabídky, která zahrnuje veškeré prostředky, čas, vybavení, personál, náklady.				

Konstrukce



Focus	Requirements – odhalit všechny přehlédnuté požadavky	Inception	Elaboration	Construction	Transition
	Analysis – dokončit analytický model				
	Design – dokončit návrhový model				
	Implementation – zajistit počáteční funkční variantu				
	Test – otestovat počáteční funkční variantu				
Goals	Dokončit všechny Use Cases (od analýzy až po testování) Dokončit anlýzu, návrh i testování				

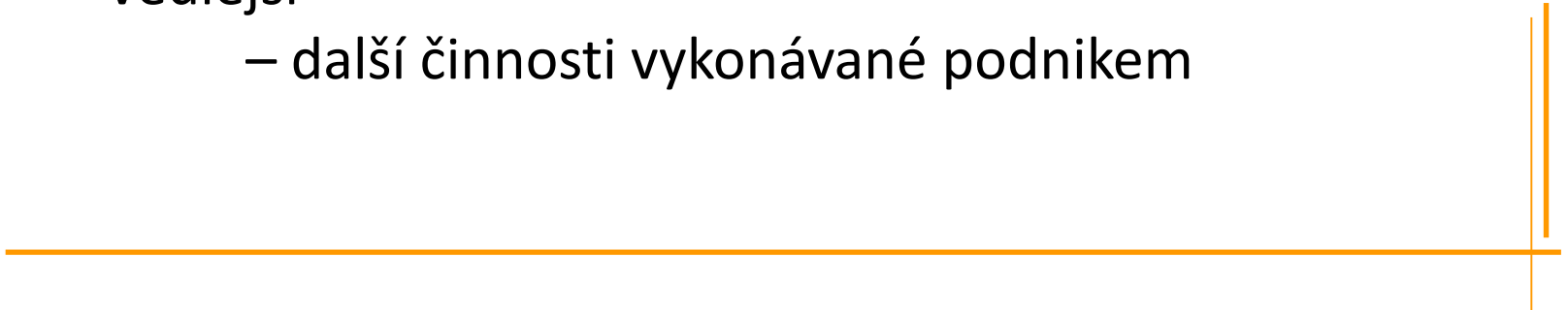
Zavedení



Focus		Inception	Elaboration	Construction	Transition
	Requirements – nic				
	Analysis – nic				
	Design – drobné úpravy pokud vznikly problémy při beta testování				
	Implementation – příprava instalačních skriptů. Oprava chyb odhalených při beta test.				
	Test – beta testování u zákazníka				
Goals	Oprava chyb Příprava prostředí u klienta na instalaci SW Úprava SW, pokud vyvstane potřeba Tvorba příruček a dalších dokumentů Poskytnout konzultantskou podporu zákazníkovi				

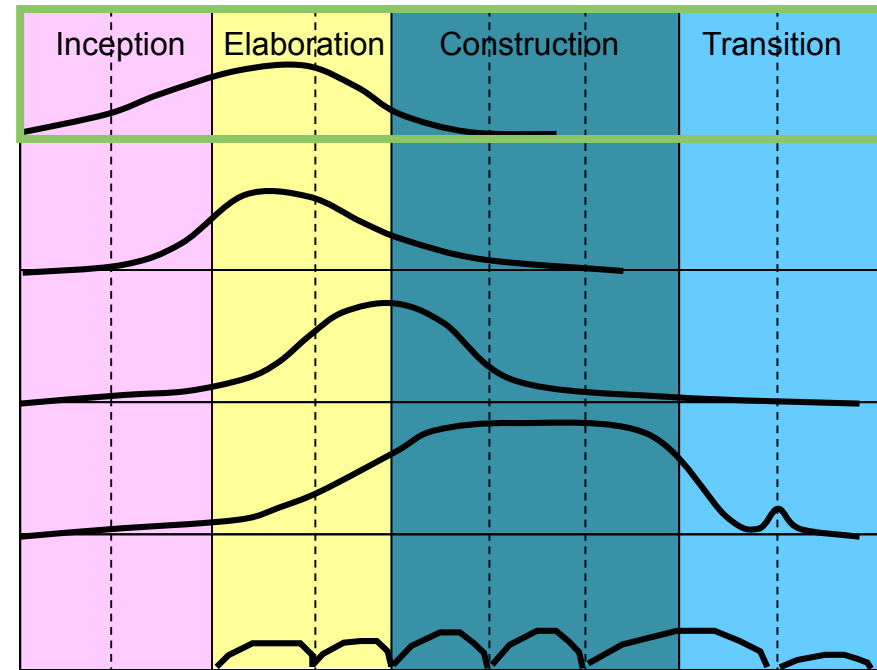
Co je to proces?

- Soubor **činností vytvářejících na základ nějakých vstup určitý výstup**
- Primární :
 - výstup má význam pro zákazníka – splňuje jeho požadavky (uspokojuje potřeby)
- Podpůrné
 - pro podporu klíčových procesů
 - jsou nezbytnou součástí klíčových procesů
- Vedlejší
 - další činnosti vykonávané podnikem



Sběr požadavků

- Vytvořit specifikaci toho, co by měl systém dělat
- Sběr požadavků se dělá na základě interview se zákazníkem



Artefakty - Specifikace požadavků

- Vision Document
- Use Case Model
- Supplementary Specification (Katalog požadavků)
 - Functional Requirements- Features
 - NonFunctional Requirements
 - UserInterface
 - Additional Constraints
- Domain Rules
- Glossary



Vision document – úvodní studie

- Slouží jako úvod do problému
 - Problém, který vyvíjený systém má řešit
 - Současný stav (systém)
 - Představa o budoucím stavu
 - Přínos systému (zadavateli, např. Zefektivnění práce, úspora peněz,...)
 - Měřítko úspěchu (kvantifikovatelná)
 - Základní funkční požadavky
 - Popis budoucího uživatele + jak bude systém používat
 - Základní nefunkční požadavky
 - Kvalitativní omezení kladená na systém
 - Popis domény a business procesů AS-IS jako příloha

- Domain (Business) Rules
 - Business Rules, Fyzikální zákony, právní záležitosti,... Všechna pravidla, která mají být zakomponována do systému
- Glossary
 - Vysvětlení pojmů dané domény- slouží jako zdroj pro hledání analytických tříd



Úvodní studie

- Co by měla obsahovat:
 - Stručný popis o jaký produkt se jedná (definice hranic systému a poskytovaných služeb) – deklarace záměru
 - Odborný článek
 - Odhad nákladů a výnosů
 - Předpokládané hardwarové / softwarové parametry pro běh systému
 - Případný návrh různých variant řešení
 - Přesvědčení investora, že projekt se vyplatí a má smysl ho realizovat

!!!! TOTO BUDE PRVNÍ CO BUDE POTŘEBA VYTVOŘIT !!!!!



Co nás čeká příště...

- Nástroje pro komunikaci SW týmů a nástroje pro vývoj
- Formální a neformální požadavky
- Use case
- Atd.



Q&A

Děkuji za pozornost



Zpracování analytické specifikace

Ing. Jan Kelnar

Analýza

Měla by odpovědět na otázku CO?

- Musí definovat konceptuální model řešeného problému
 - datový model – entity, vztahy, omezení
 - funkční model – služby pro záznam, využití dat
 - dynamický model – možné stavy dat a jejich změny
- Musí stanovit za jakých podmínek je analytická dokumentace akceptovatelná


Formální a neformální specifikace

1. Neformální specifikace / formální specifikace
 - odborný článek / jednoznačná sémantika
2. Funkční / Nefunkční požadavky
3. Seznam událostí a reakcí systému
4. Požadované výstupy
5. Procesní model
6. Datový model
7. Prezentační model

Požadavky obecně

- Je třeba myslet na všechny typy požadavků a je potřeba ptát se všech relevantních skupin zainteresovaných osob (stakeholder)

Minimálně následující:

- Požadavky na vlastní funkce
 - Požadavky na rozhraní (uživatelské, softwarové, HW, komunikační...)
 - Nefunkční požadavky (výkon, bezpečnost, spolehlivost, dostupnost, škálovatelnost)
 - Další požadavky (legislativní, vícejazyčnost ...)
- 

Cíle

- Vymezit hranice systému
- Umožnit přesnější odhad pracnosti
- Vyjasnit si zadání se zákazníkem
- Zachytit omezení, která jsou na systém kladena

Způsob zachycení:

- Strukturovaný text
 - Grafické zobrazení
 - UML use case
 - Využití vlastních stereotypů
-

Funkční požadavky

- Specifikují co by systém měl dělat (umět) vs. nefunkční požadavky – jaký by systém měl být
- „systém má zobrazit x záznamů z DB“ vs. „záznamy musejí být každý den aktualizované“
- Fční požadavky mohou být výpočty, technické detaily, manipulace a zpracování dat a jiné procesy, které musí systém vykonávat
- „systém musí umět <<funkční požadavek>>“ vs „systém by měl splňovat <<nefunkční požadavek>>“
- Charakterizuje určitý výsledek systému

Funkční požadavky

- Funkční požadavky (Features) –
 - Co má systém dělat (jen základní funkcionalita – blíže popisují Use Cases
 - Vztah UC a f.p. Je obecně $M \times N$

Psaní požadavků

`<id> <system> bude <function>`

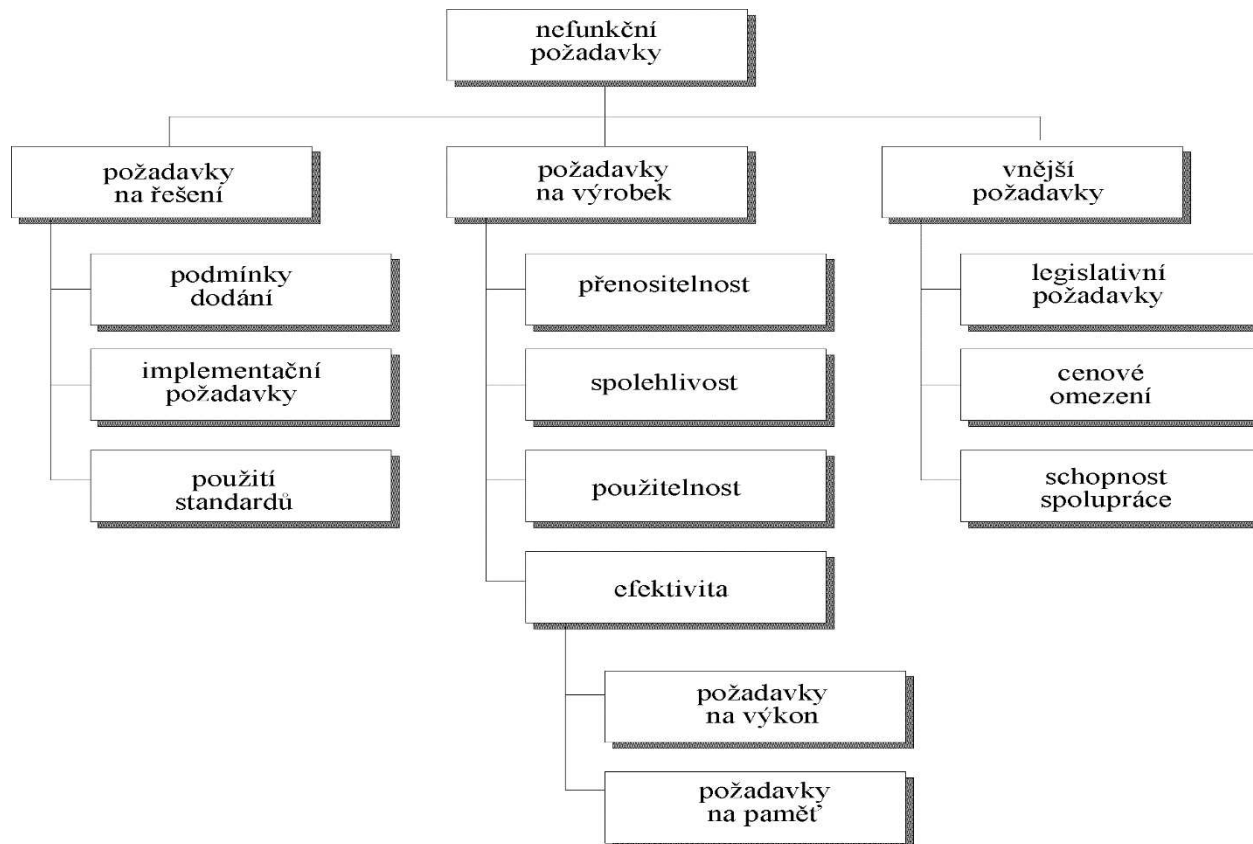
unique identifier name of system keyword function to be performed

e.g. „RQ32 Bankomat bude validovat PIN.“

- Neexistuje standard na psaní požadavků
-

Nefunkční požadavky

- Kritéria podle kterých se dá hodnotit funkce systému, spíš než jeho specifické funkce – „kvalita systému“



Kategorizace FURPS+

(F)URPS+

Functionality – zda SW splňuje požadavky na daný business

Usability – snadnost používání aplikace (např. dodržet součas. způsob)

Reliability – měřitelné požadavky na spolehlivost (mean time failure)

Performance - měřitelné pož. na výkon(response time, throughput)

Supportability - přizpůsobitelnost, udržitelnost, přenositelnost (SW, HW)

Implementation – nástroje, prog. Jazyk, HW,

Interface – výměnné formáty, rozhraní pro komunikaci s dalšími syst.

Operation – pož. na administraci a správu systému

Packaging – způsob dodání systému

Legal – licence, certifikáty

Pozor na ...

- **Nejednoznačnost požadavků**
 - Systém bude výkonný
 - Systém bude spolehlivý
 - Systém bude přívětivý
- **Stanovit priority požadavků**
- **Kolize požadavků**

Modelování business procesů

1. Globální model procesů

- Struktura systému – vazby, komunikace procesů

2. Model postupu (průběhu) procesu

- Logika postupu jednoho procesu

3. Základní popisná tabulka procesu

BPMN (2.0)

Flow Objects

Events



Activities



Gateways



Connectors

Sequence Flow



Message Flow



Association



Artifacts

Data Object



Name
[State]

Text Annotation



Add Text Here

Group



Swimlanes

Pool

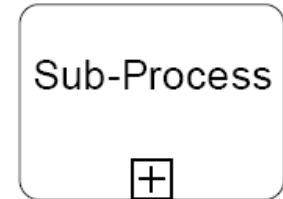
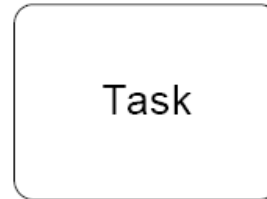


Lanes (within a Pool)



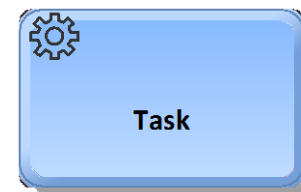
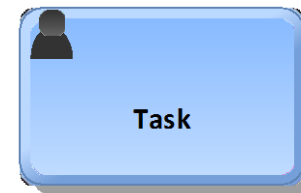
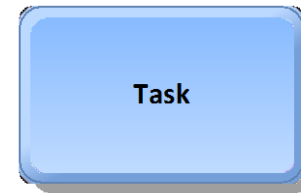
Task

- Práce prováděná uvnitř procesu
 - Task
 - atomická
 - Sub-process
 - dále dělitelná



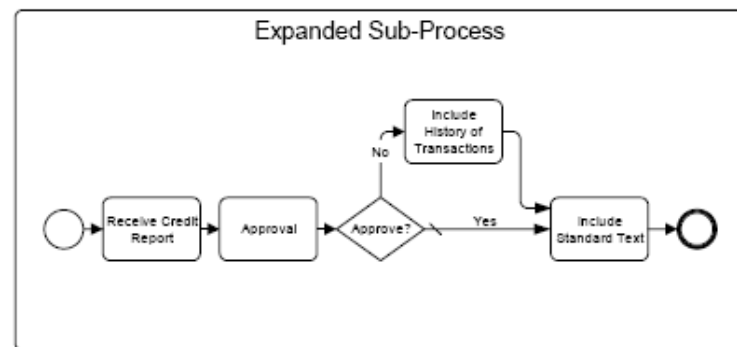
Task - Activity

- Atomická-dále nedělitelná aktivita
- Možno blíže charakterizovat pomocí dodané ikonky (vykonávaná uživatelem, strojově vykonávaná)



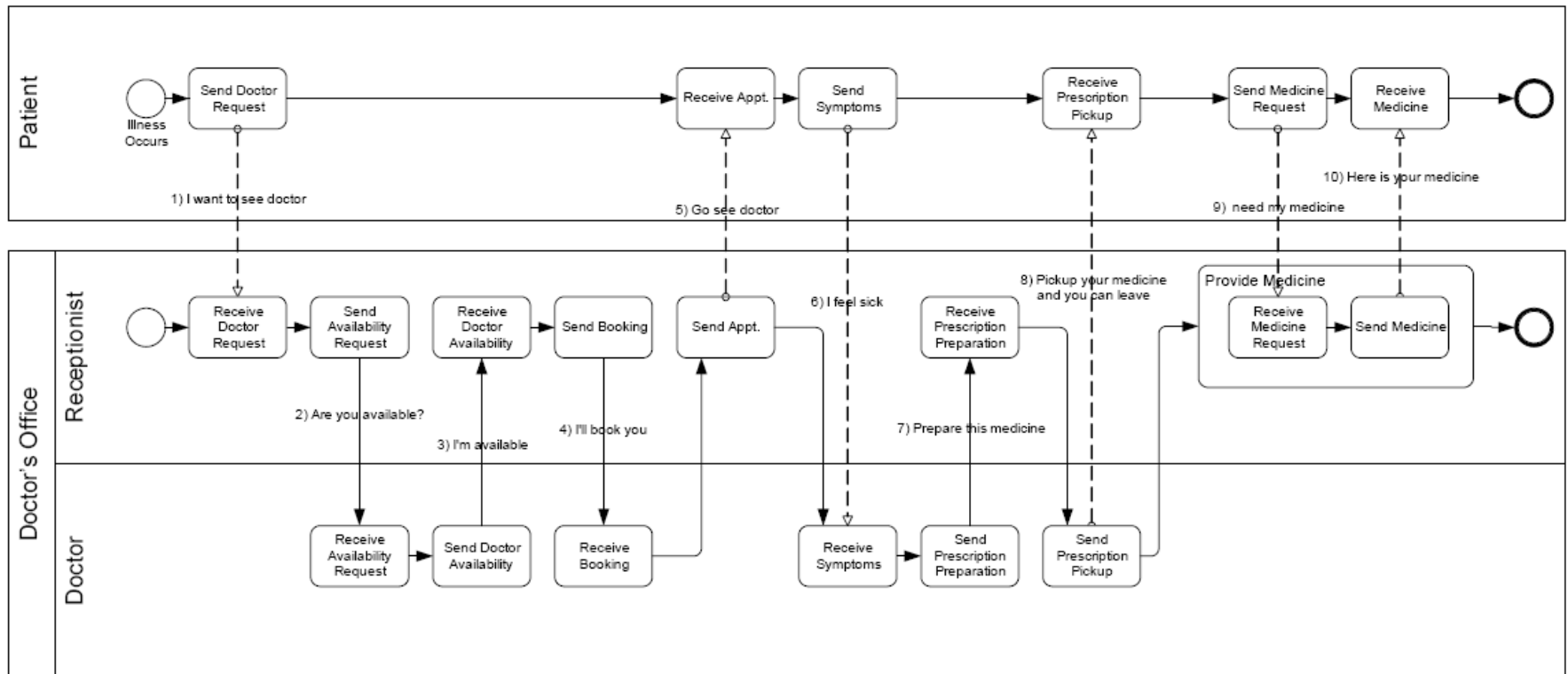
Sub process

- Umožňuje vytvářet hierarchie procesů-návrh od shora dolů



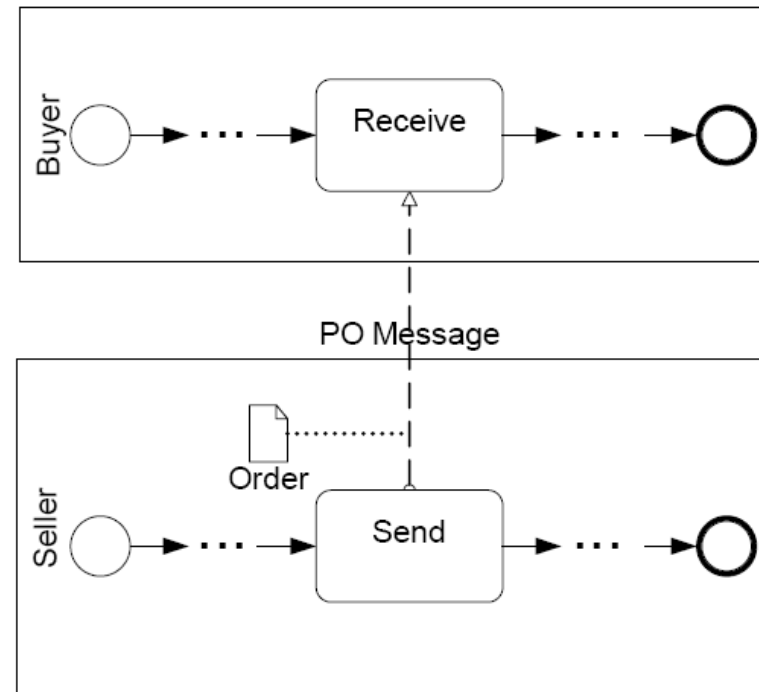
Swim lanes

- Pool- reprezentuje participanta procesním diagramu
- Lane – dělení v rámci 1 poolu



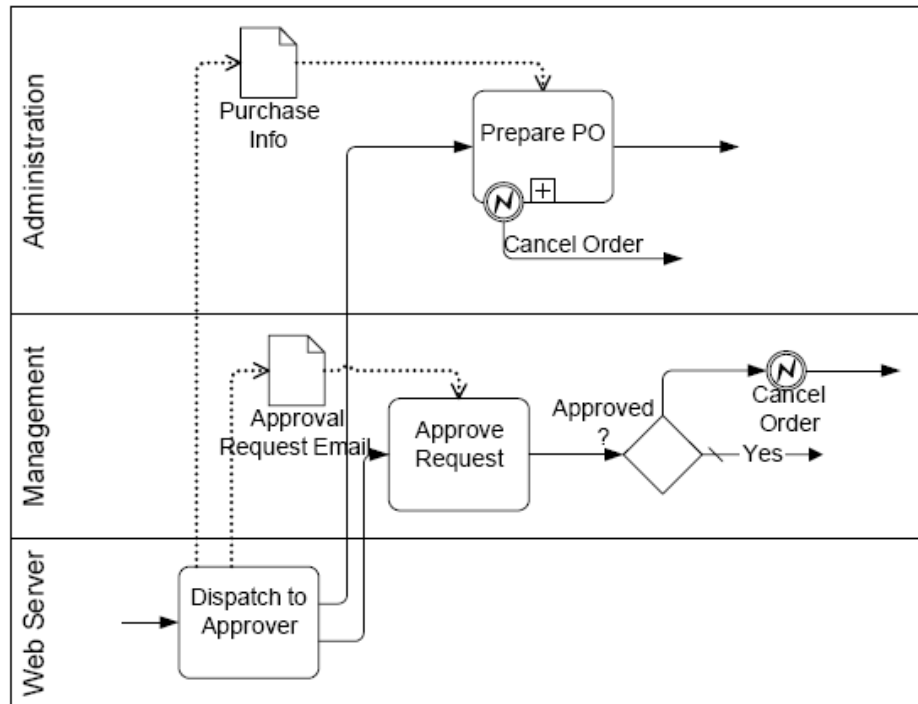
Pool

- Reprezentuje participanta v interaktivním B2B BPD
- Participant = business role (osoba, organizace)
- Pool může být prázdný tzv. black box(externí)
- Interakce mezi pooly pomocí message flow
- Sequence flow nesmí překročit hranice poolu



Lane

- Rozdělení poolu na oddíly
- Obvykle reprezentují role uvnitř organizace (oddělení, pracovní role)
- Sequence flow může křížit jednotlivé Lanes



Konektory

- SF ukazuje posloupnost aktivit
- MF ukazuje tok zpráv mezi 2 procesy
- A slouží k připojení artefaktů k aktivitám nebo tokům

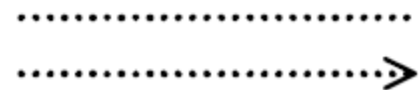
Sequence
Flow



Message Flow

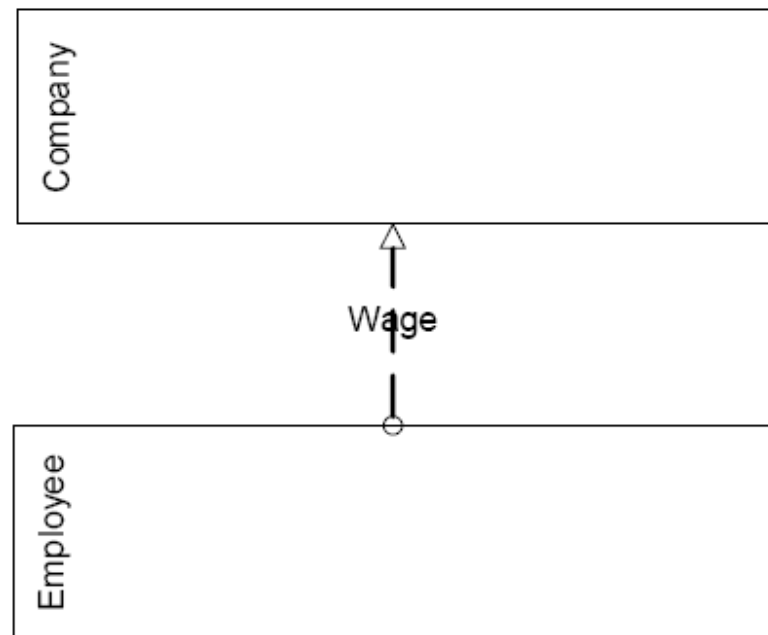


Association



Message flow

- Komunikace mezi 2 processy (pooly) nebo objekty z 2 různých procesů (poolů)
- Nikdy, komunikace v rámci poolu



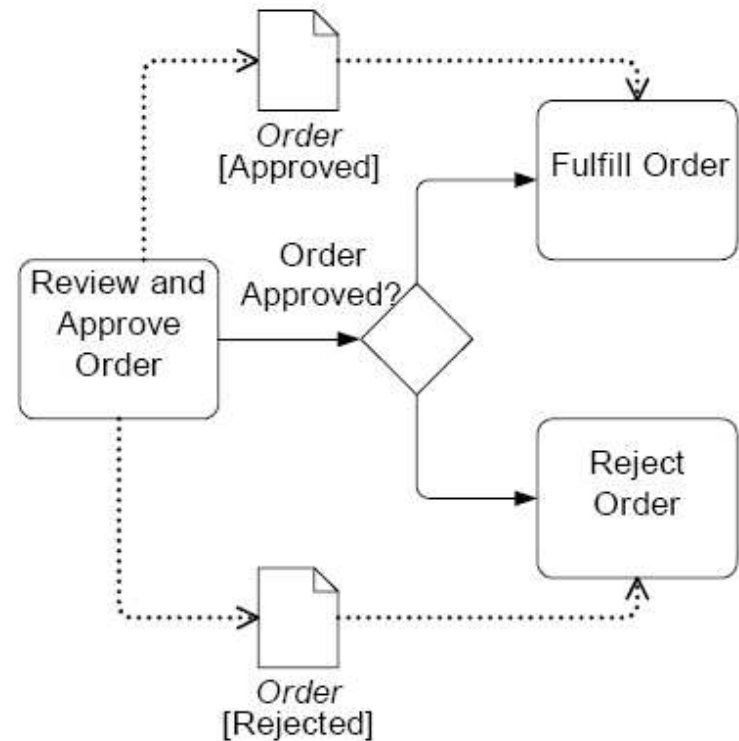
Sequence flow

- Ukazuje následnost jednotlivých aktivit
- Zdrojem nebo cílem:
 - Gate
 - Activity
 - Event
- Nesmí křížit hranici SubProcessu ani Poolu



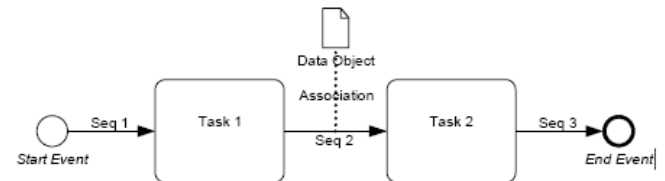
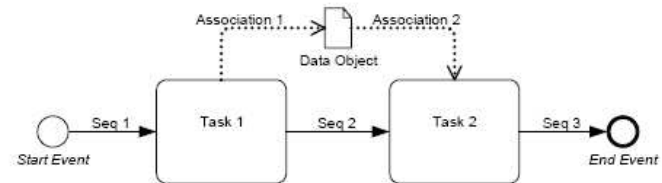
Objekty

- Reprezentují data a informace uvnitř procesu
- Slouží jako vstupy a výstupy aktivit
- Mohou mít přiřazen stav (Zobrazen v hranatých závorkách)



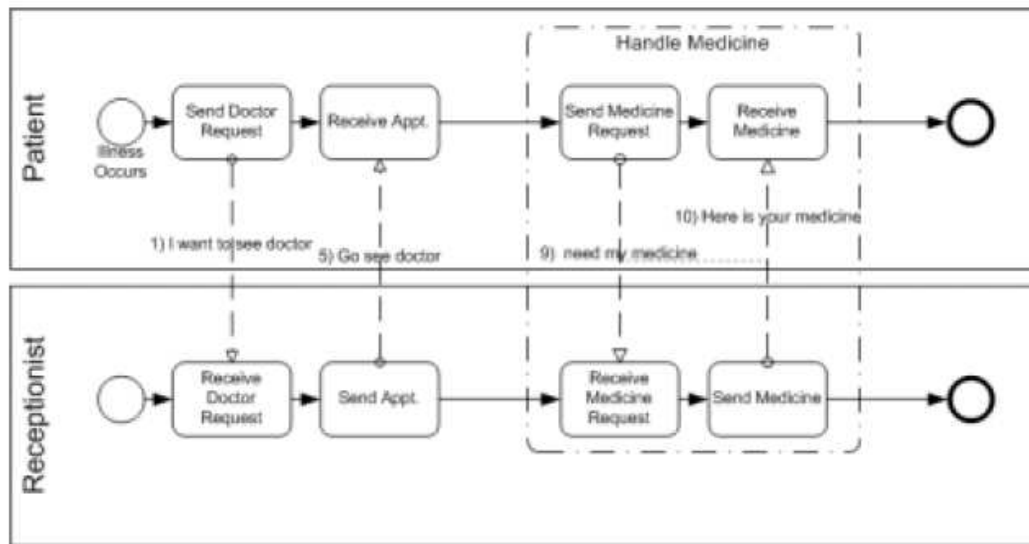
Data flow

- Znázornění toku objektů mezi aktivitami
- 2 možnosti
 - Orientované asociace, data flow je odělen od sequence flow
 - Neorientovaná asociace, Data flow je spojen se sequence flow



Skupina

- Umožňují zvýraznit určitou vybranou část procesu
- Nemá další dopad např. Na vykonávání procesu



Události

- „Něco“, co se stane v průběhu procesu
- Mohou tok
 - Nastartovat
 - Přerušit
 - Ukončit
- Obvykle mají spouštěč (trigger) nebo efekt (result)

Start



Intermediate



End




Brány

- „Brány umožňují větvení a slučování toků nebo procesů v závislosti na uvedených podmínkách. Znázorňují se pomocí kosočtverce
 - **Exkluzivní** - vytváří několik cest toku procesu, ale podmínkou je, že tok procesu proběhne pouze jednou z možných cest.
 - **Inkluzivní** - tok procesu může projít přes bránu více než jednou cestou
 - **Komplexní** - dochází k dělení cest v několika branách
 - **Paralelní** - tok procesu proudí více cestami najednou
-

Komunikační infrastruktura SW týmu

- Repository

Sdílený prostor SW týmu:

- Uživatelské účty všech členů týmu
 - Uživatelské účty pro zákazníka
 - Zdrojový kód
 - Sledování požadavků (issue tracking)
 - Sledování chyb (bug tracking)
 - Související dokumenty
- 

Komunikační infrastruktura SW týmu

Název	Vlastnosti	Testy	Upravitelné workflow	Vlastní pole	SLA	Plugin API	Více projektů	Fulltext
Assembla	Emailová notifikace, integrace se SVN, grafy, reporty	X	X	X	-	X	X	-
BugTracker.NET	Notifikace, jednoduchá app, customizovatelná	-	X	X	-	-	X	X
Bugzilla	Reporty grafy	X	X	X	-	X	X	X
codeBeamer	Správa dokumentů, Scrum, waterfall, wiki, dashboard	X	X	X	X	X	X	X
Google Code Hosting	Emailové notifikace, SVN, wiki	-	X	X	-	-	X	X
Team Foundation Server	Definice workflow, dokumentace procesů atd.	X	X	X	-	X	X	X
Redmine	Wiki, forum, novinku, e-mailová integrace, exporty, gantovy diagramy	X	X	X	-	X	X	X

Veřejné repozitáře

- <http://sourceforge.net> (CVS, subversion)
- <http://tigris.org> (CVS, subversion)
- <http://code.google.com/hosting> (subversion)
- <http://www.assembla.com> (subversion)
- <http://www.codeplex.com> (subversion, TFS)
- <http://repo.or.cz> (Git)

Správa zdroj.kódu s centrální repository

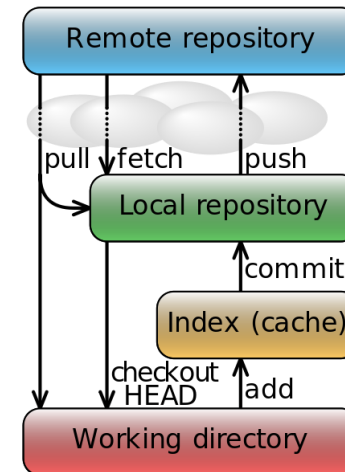
- CVS (neatomické commity, neřeší přesuny a přejmenování)
 - řeší konflikty metodou "kdo dřív přijde ten dřív mele" (když se soubor změní, musí stáhnout aktuální změny a problém vyřešit)
- Subversion
 - Podporuje větvení (zdrojové fční kódy v hlavní větvy (trunk), nové funkce v pracovních větvích (branch), které se potom sloučí s trunk)
- Rational ClearCase - placená licence
- Microsoft TFS - zdarma pouze pro Open source projekty (codeplex.com) nebo pro max 5 uživatelů v TFS Express edici, robustní

Distribuovaná správa zdroj.kódu

- Git
 - Silná podpora nelineárního vývoje (branch, merge)
 - Podpora např. i SVN
- Mercurial - více GUI, jednodušší na učení
- Arch
- Darcs

1 programátor = 1 repository

- Operace pull
- Operace push



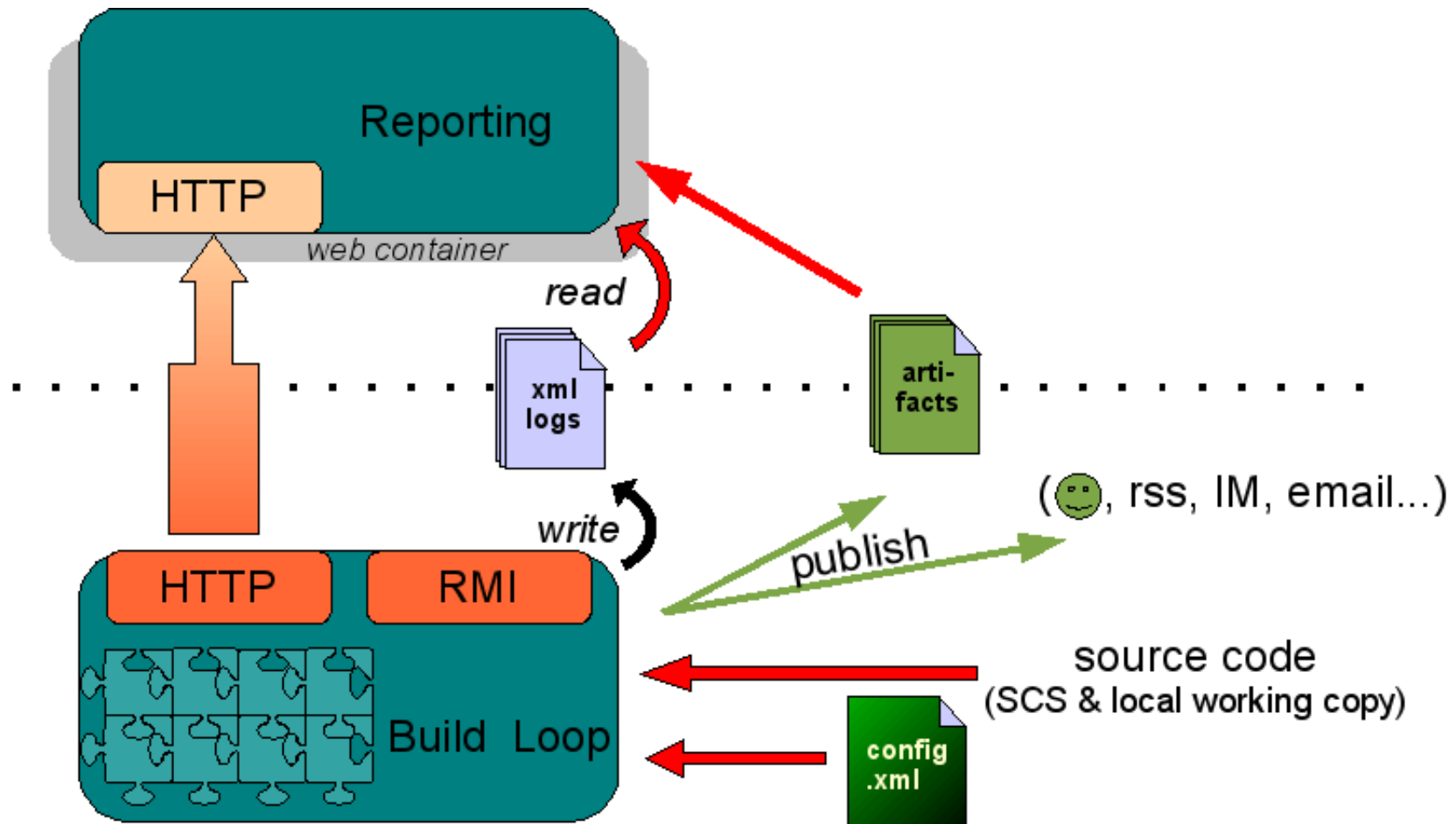
Automatický build / průběžná integrace

- Nástroj, který stáhne poslední verzi z repozitáře, zkompiluje ji a připraví instalační balíček
- CruiseControl (.NET), BuildBot, Hudson, TFS

Výhody:

- Rychlé nalezení chyb ve zdrojovém kódu
 - Automatická kontrola kódu
 - Přehled všech členů týmu o stavu buildu
 - Přehledné verzování jednotlivých verzí
 - Rychlý přístup k poslední verzi aplikace
 - Ušetření času při kompilaci a vydání nové verze
 - Ušetření testovacích kapacit při využití automatického testování
-

Automatický build / průběžná integrace



Code swarm

- Vizualizace práce na projektu

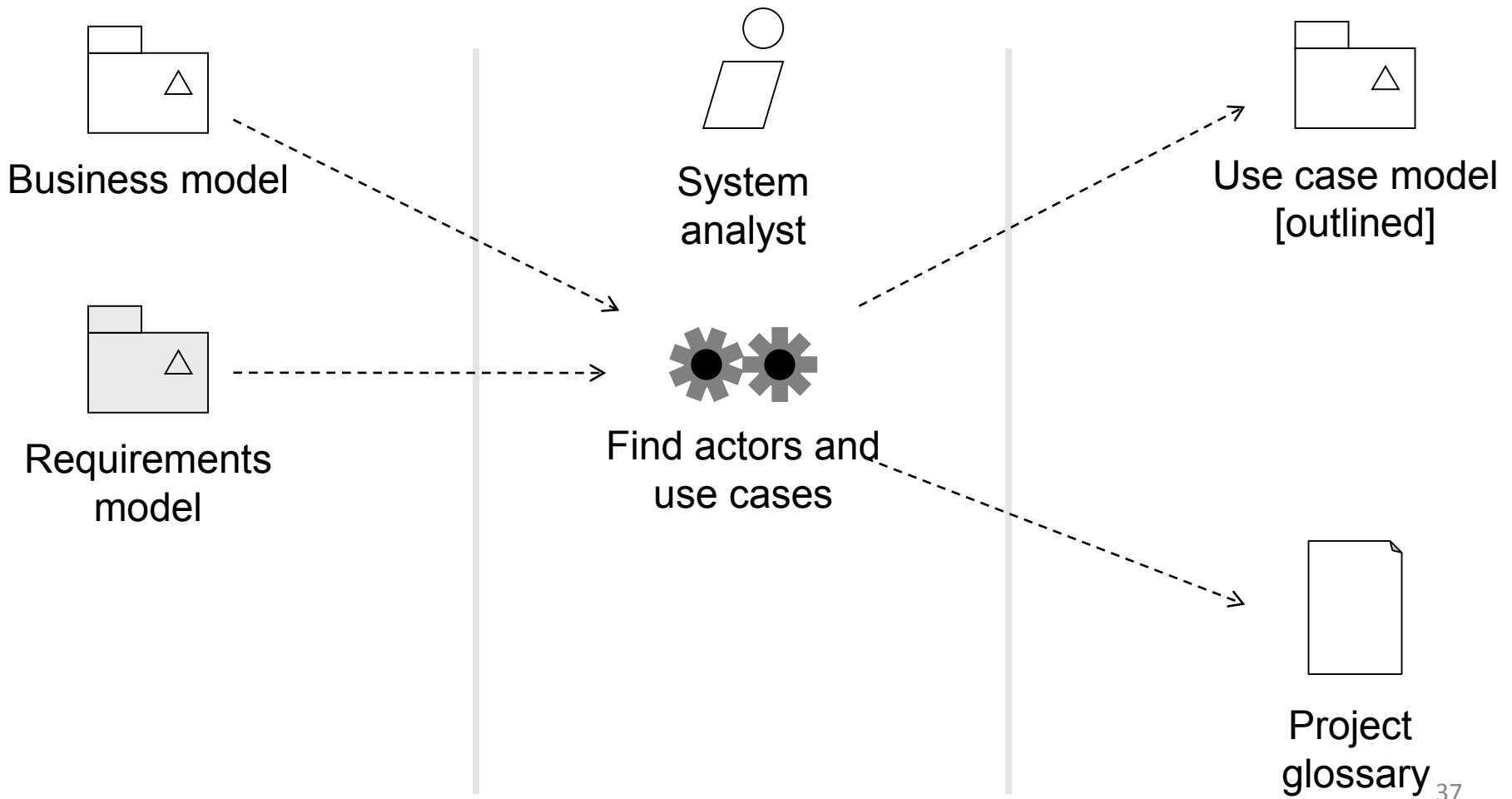


Modelování Use Case

Use case modelling

- Způsob zachycení funkčních požadavků
- Vyjadřují, kdo bude jakým způsobem používat systém
- Cíle modelování:
 - Najít hranice systému (co je součástí, co je vně systému)
 - Najít aktéry
 - Vytvořit vlastní use case
 - Určení jaké Use Case
 - Psaní scénářů

Nalezení aktérů a use case



Mapování požadavků na UC

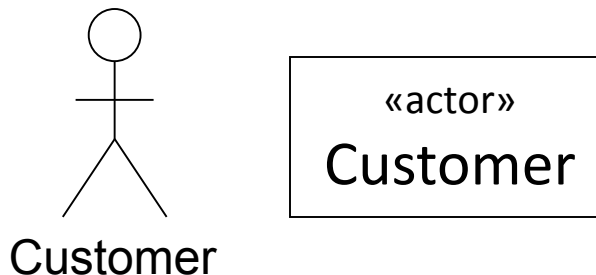
- Vztah UC a Features je obecně MxN

	Use cases			
	U1	U2	U3	U4
Requirements	R1			
	R2			
	R3			
	R4			
	R5			

Requirements
Traceability
Matrix

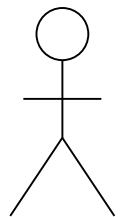
Co je aktér?

- Aktér je cokoliv, co komunikuje přímo se systémem
 - Člověk, jiný systém, čas
 - Určením aktérů de facto určujeme hranice systému (co je součástí a co je aktér)
- Aktér stojí vně systém
- Aktér je uživatelská role, jedna fyzická osoba může mít více rolí
- Odlišení ne/živých



Identifikace aktérů

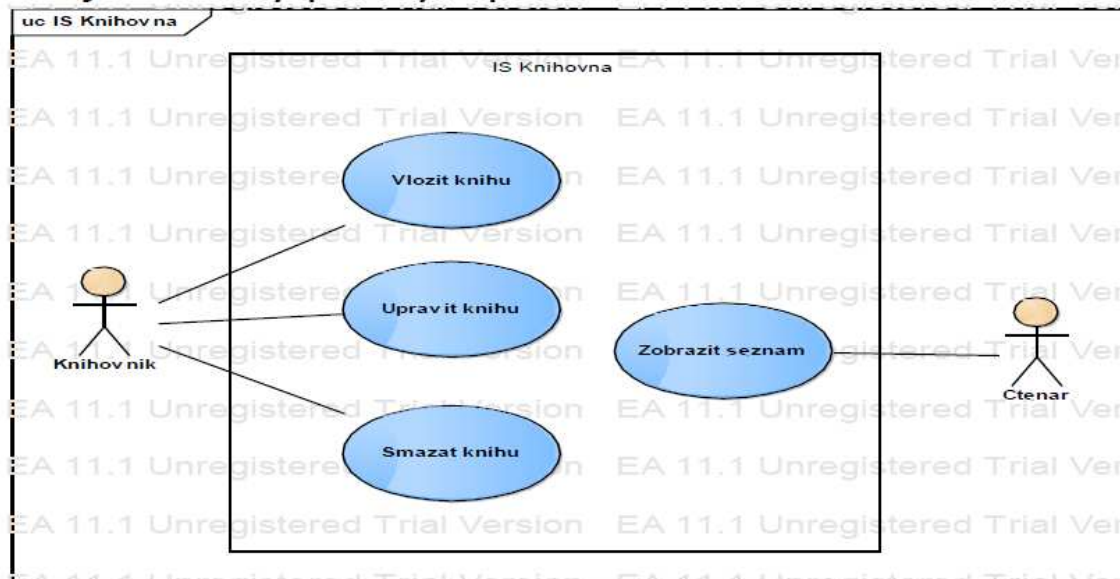
- Ptáme se:
 - Kdo používá systém
 - Kdo instaluje systém?
 - Kdo vypíná a zapíná systém?
 - Kdo udržuje?
 - Jaké další systémy používá řešený systém?
 - Kdo získává a poskytuje informace systému?
 - Děje se něco v systému pravidelně resp. V určitém čase?
- Pozn. Ne všechny aktéry odvodíme s business procesů



Time

Co jsou use cases?

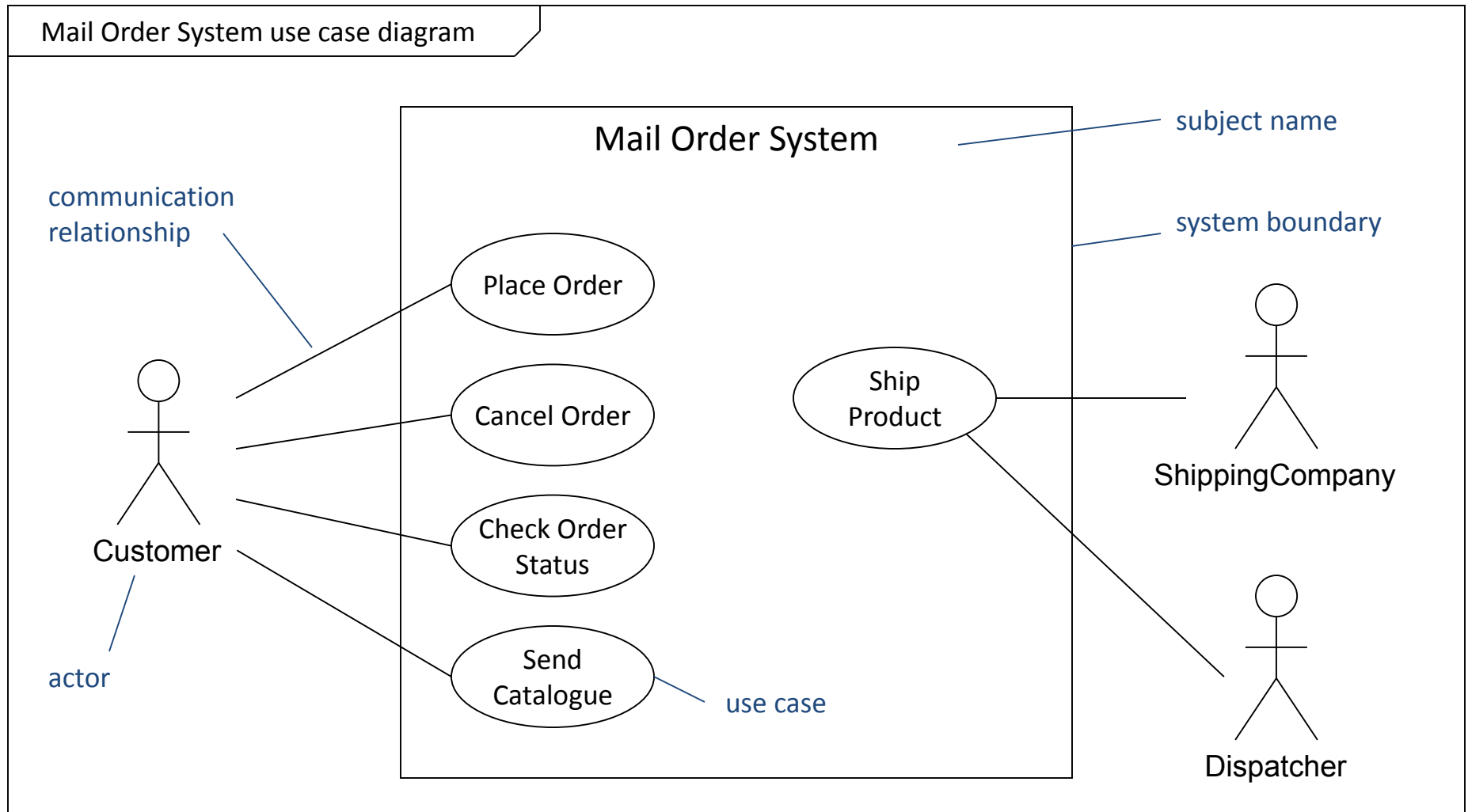
- Use Case je něco, co aktér chce, aby systém dělal
- Use Case je vždy spouštěn aktérem
 - Primární aktéři spouštějí use case
 - Sekundární aktéři mohou interakovat s UC
- Use Case jsou vždy psány z pohledu uživatele



Odvození Use Cases z business Analýzy

- Na základě funkčních požadavků víme, které části business procesů bude systém podporovat
- Rozpracujeme BP na jednotlivé Use Case
- Část BP podporovaná systémem, je předobraz pro budoucí scénář
- Pozor, ne všechny UC lze odvodit z BP (administrace, konfigurace...)

Use case diagram



Postup tvorby use case

1. Nalezení hlavních use case
2. Napsání prvotních scénářů (přehledově)
3. „Refactoring“ use case- odvození pomocných use case
4. Přepřarování a zjemnění scénářů
 1. Nejprve píšeme hlavní toky
 2. Pak doplňujeme alternativní (viz dále)

Specifikace Use Case

use case name

use case identifier

brief description

the actors involved in the
use case

the system state before the
use case can begin

the actual steps of the use
case

the system state when the
use case has finished

alternative flows

	Use case: PaySalesTax
	ID: 1
	Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter.
	Primary actors: Time
	Secondary actors: TaxAuthority
	Preconditions: 1. It is the end of the business quarter.
	Main flow: 1. The use case starts when it is the end of the business quarter. 2. The system determines the amount of Sales Tax owed to the Tax Authority. 3. The system sends an electronic payment to the Tax Authority.
	Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax.
	Alternative flows: None.

Pre a postconditions

- Preconditions obsahují stav systému před spuštěním UC (co musí platit, aby šel UC spustit)
- Postconditions obsahují stav systému po spuštění UC (efekt UC)

Place Order
Preconditions: 1. A valid user has logged on to the system

Postconditions: 1. The order has been marked confirmed and is saved by the system

Hlavní scénář (Happy scenario)

<číslo> <aktér/systém> <akce>

- Posloupnost akcí
- Vždy začíná akcí aktéra
 - Např:
 - 1) Use Case začíná když <actor> <function>
- Odpovídá situaci, kdy vše jde bez problémů a aktér dosáhne svého cíle
- Alternativy je možné vyjádřit větvením nebo alternativními scénáři



Větvení pomocí If

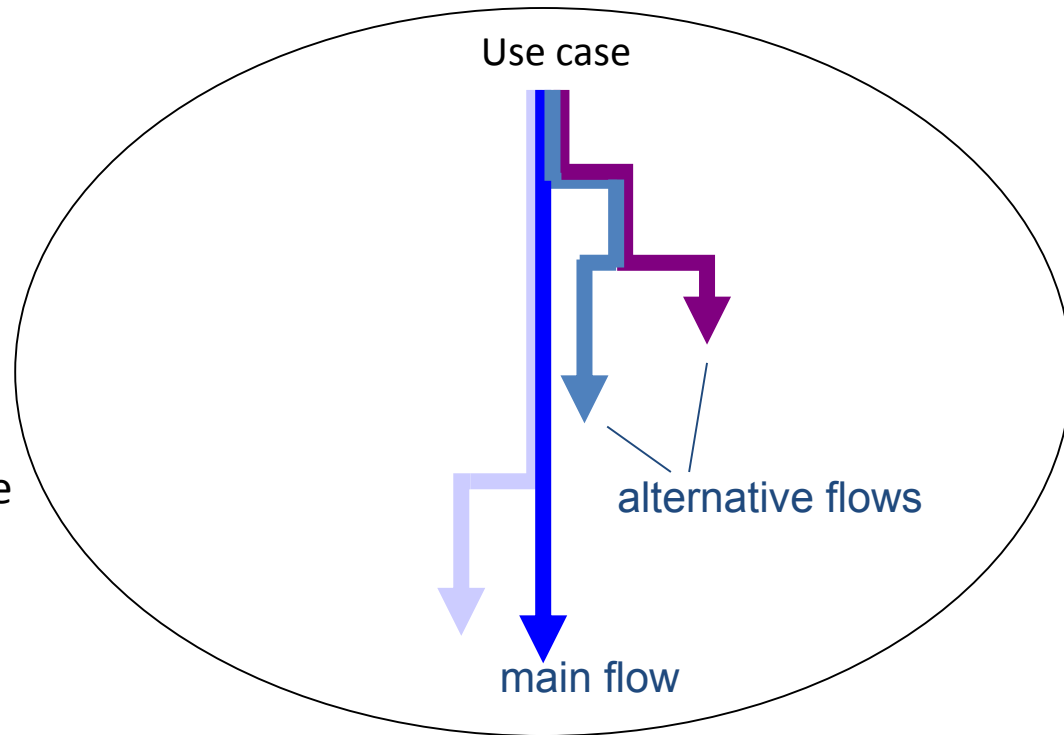
Use case: ManageBasket
ID: 2
Brief description: The Customer changes the quantity of an item in the basket.
Primary actors: Customer
Secondary actors: None.
Preconditions: 1. The shopping basket contents are visible.
Main flow: 1. The use case starts when the Customer selects an item in the basket. 2. If the Customer selects "delete item" 2.1 If the Customer types in a new quantity 3. The system removes the item from the basket. 3.1 The system updates the quantity of the item in the basket.
Postconditions: None.
Alternative flows: None.

Opakování pomocí For

Use case: FindProduct
ID: 3
Brief description: The system finds some products based on Customer search criteria and displays them to the Customer.
Actors: Customer
Preconditions: None.
Main flow: 1. The use case starts when the Customer selects "find product". 2. The system asks the Customer for search criteria. 3. The Customer enters the requested criteria. 4. The system searches for products that match the Customer's criteria. 5. If the system finds some matching products then 5.1 For each product found 5.1.1. The system displays a thumbnail sketch of the product. 5.1.2. The system displays a summary of the product details. 5.1.3. The system displays the product price. 6. Else 6.1. The system tells the Customer that no matching products could be found.
Postconditions: None.
Alternative flows: None.

Alternativní scénáře

- Alternativní scénáře odpovídají chybám, výmkám v hlavním scénáři
- Teoreticky může existovat veliká řada a.s.
 - Vyber jen ty nejdůležitější
 - Pro skupinu analogických vyber je jednu alternativu



Odkazovat na alter. scénáře

Use case: CreateNewCustomerAccount	
ID: 5	
Brief description: The system creates a new account for the Customer.	
Primary actors: Customer	
Secondary actors: None.	
Preconditions: None.	
Main flow: <ol style="list-style-type: none">1. The use case begins when the Customer selects "create new customer account".2. While the Customer details are invalid<ol style="list-style-type: none">2.1. The system asks the Customer to enter his or her details comprising email address, password and password again for confirmation.2.2 The system validates the Customer details.3. The system creates a new account for the Customer.	
Postconditions: <ol style="list-style-type: none">1. A new account has been created for the Customer.	
alternati ve flows {	Alternative flows: InvalidEmailAddress InvalidPassword Cancel

Alternativní scénář



Alternative flow: CreateNewCustomerAccount:InvalidEmailAddress

ID: 5.1

Brief description:

The system informs the Customer that they have entered an invalid email address.

Primary actors:

Customer

Secondary actors:

None.

Preconditions:

1. The Customer has entered an invalid email address

Alternative flow:

1. The alternative flow begins after step 2.2. of the main flow.
2. The system informs the Customer that he or she entered an invalid email address.

Postconditions:

None.

Alternativní tok začíná
v nějakém kroku
hlavního scénáře



- Alternativní tok může nastat místo hlavního toku
- Alternativní tok může nastat za jistým krokem hlavního scénáře
- Alternativní tok může nastat kdykoliv během hlavního toku

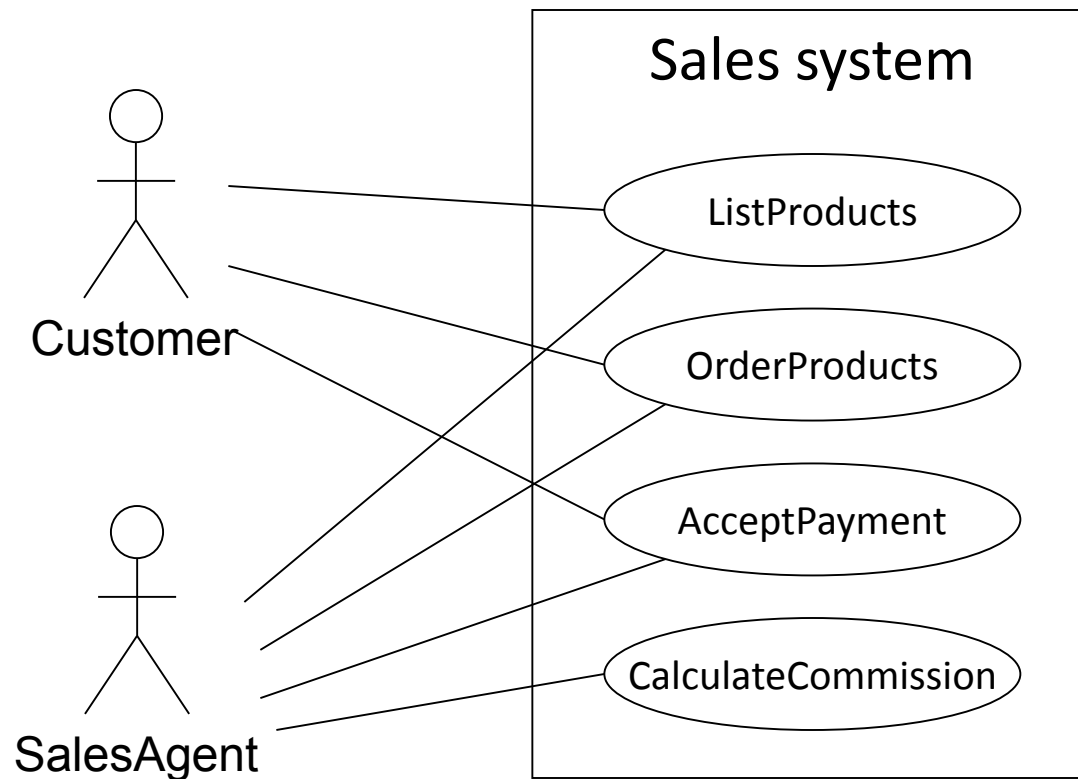
Kdy používat UC

- Uce cases popisují systém z hlediska aktérů a jsou dobrou volbou když:
 - Systém je definován primárně funkčními požadavky
 - Systém má velkou řadu uživatelů vyžadujících rozličné funkce
 - Systém má hodně rozhraní
- Use Cases není vhodné použít když:
 - Systém je definován nefunkčními požadavky
 - Systém má málo požadavků
 - Systém má málo nebo žádné rozhraní

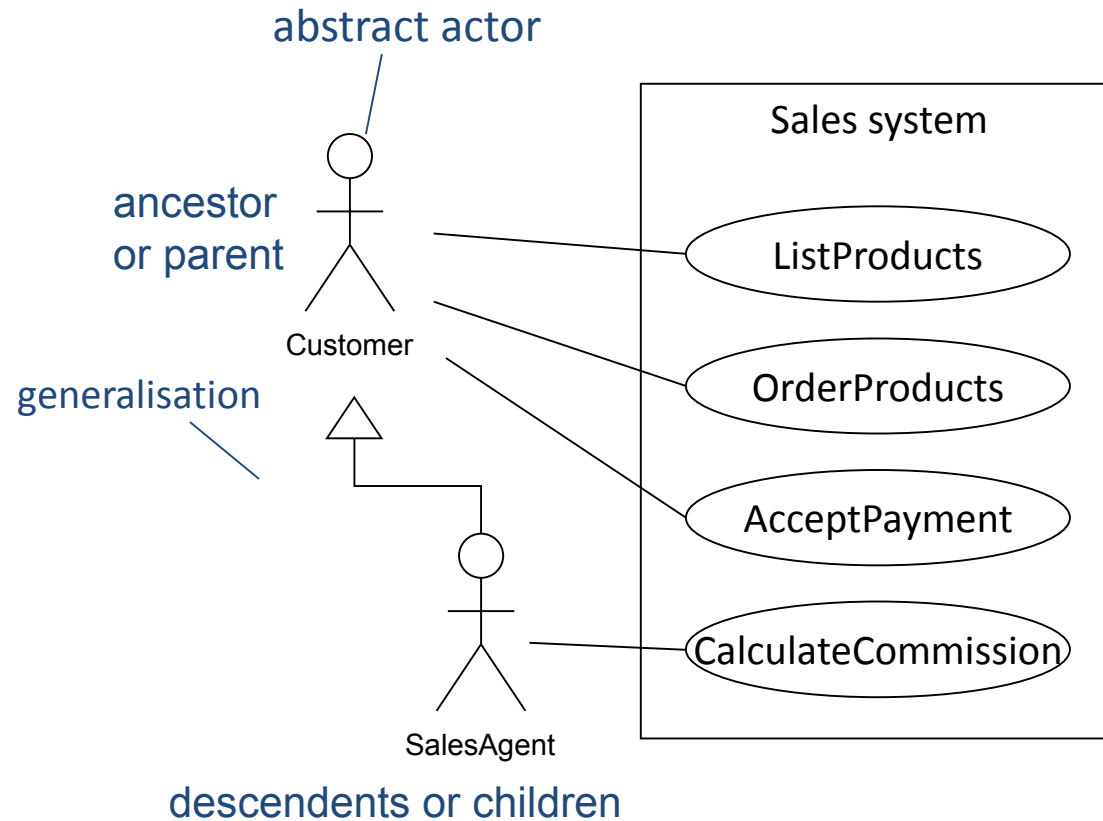
Zjemnění Use Cases

- Použijeme další relace pro zpřehlednění, zjednodušení UC
 - Generalizace aktérů
 - Generalizace UC
 - «include»
 - «extend»

Actor generalization - example

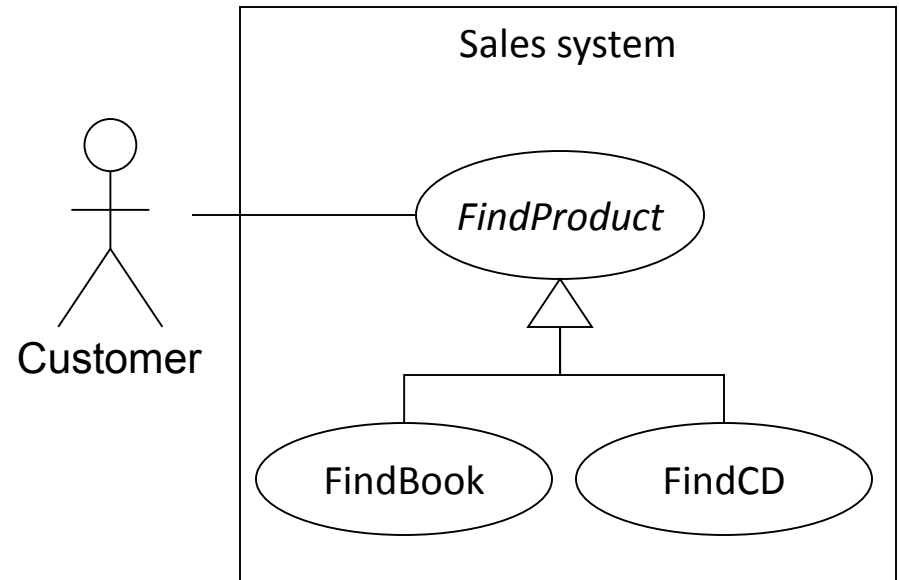


Actor generalisation



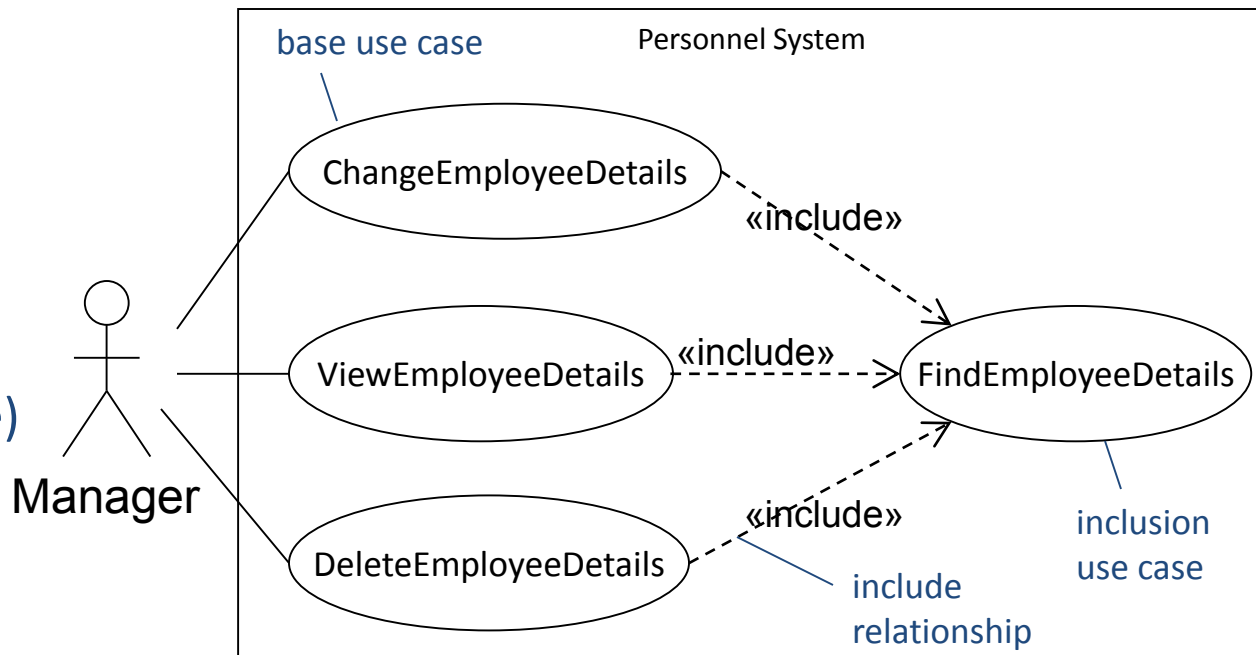
Use case generalisation

Use case generalization semantics			
Use case element	Inherit	Add	Override
Relationship	Yes	Yes	No
Extension point	Yes	Yes	No
Precondition	Yes	Yes	Yes
Postcondition	Yes	Yes	Yes
Step in main flow	Yes	Yes	Yes
Alternative flow	Yes	Yes	Yes

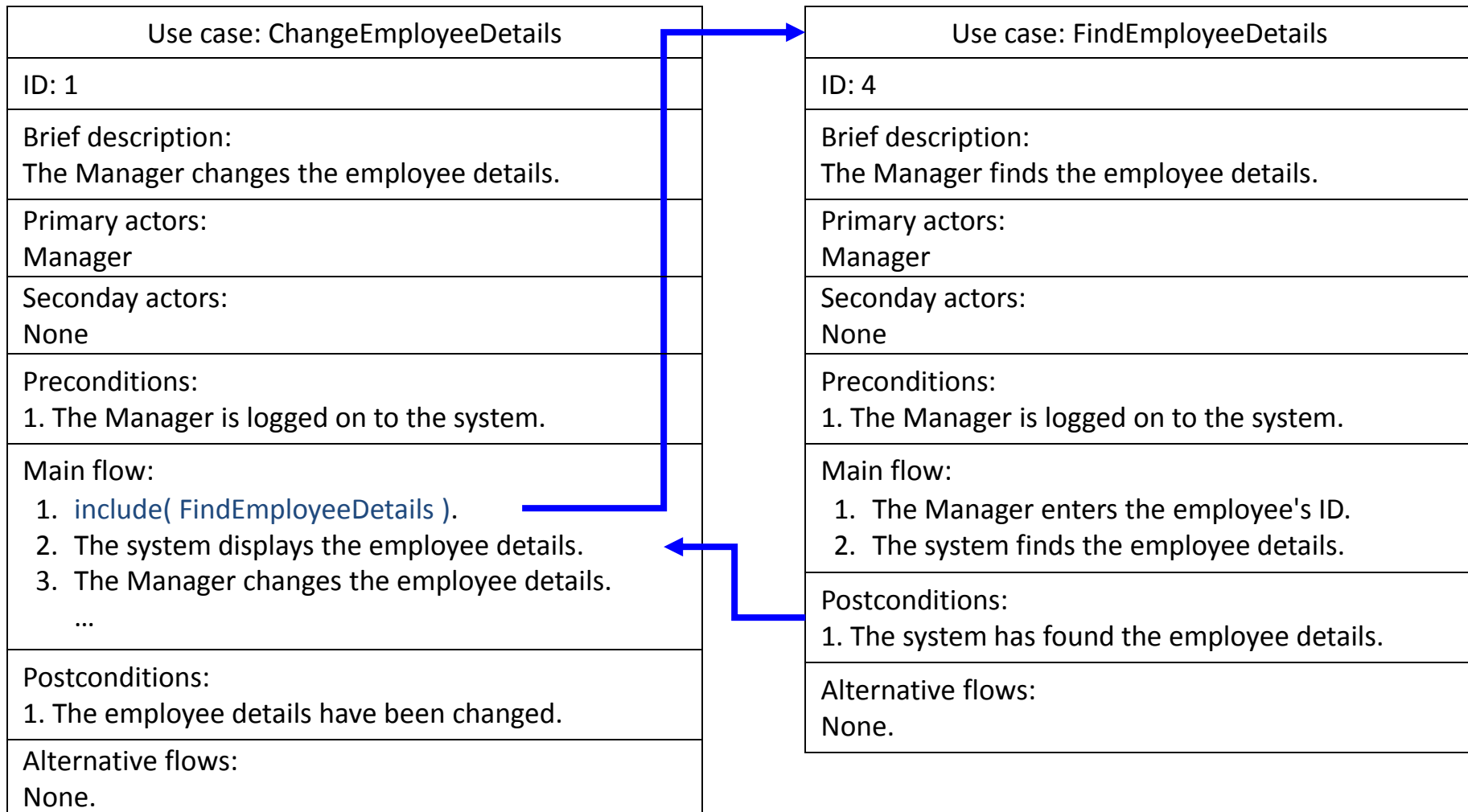


«include»

- Důvody použití:
 - Opakovaně používaný scénář
 - Samostatně používaný scénář
- Základní UC volá vnořený pomocí:
include(InclusionUseCase)
 - Když v kládná skončí
vracíme se na místo
volání
 - Základní UC je
nekompletní

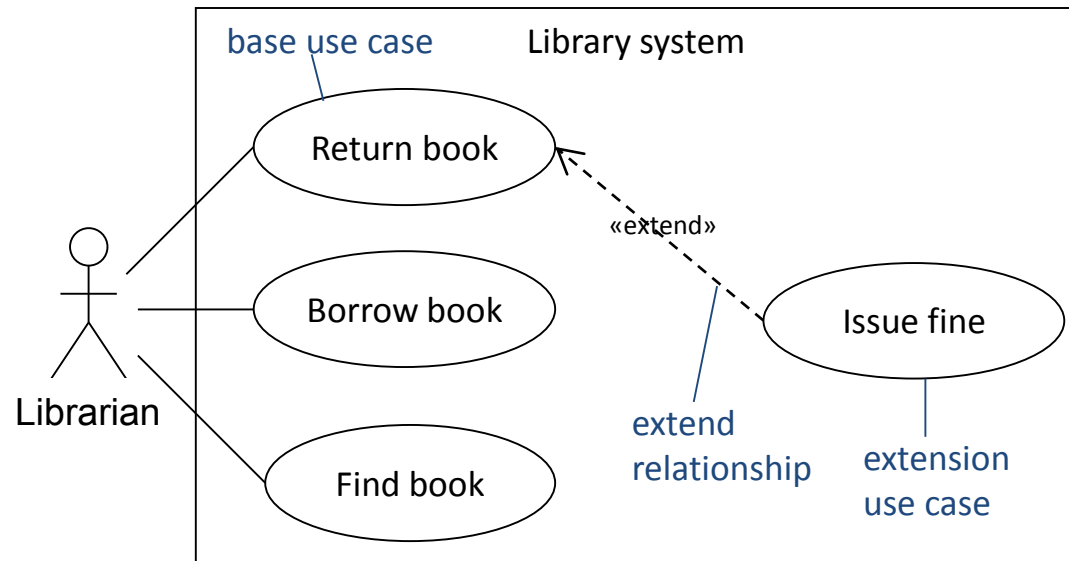


«include» příklad

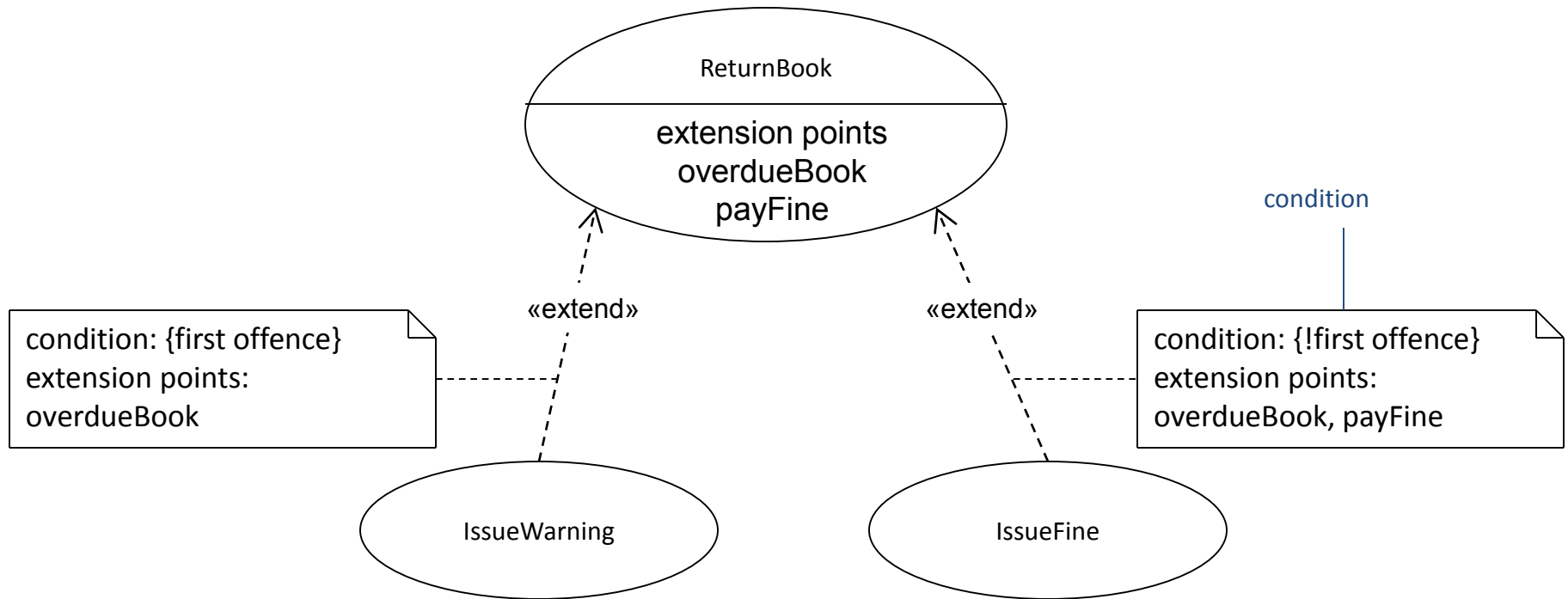


«extend»

- «extend» je způsob, jak přidat funkcionalitu k základnímu UC
- Základní UC o rozšiřovaném neví
- Vkládaný může vkládat více segmentů
- Základní specifikuje místa rozšíření: Extension points



Podmíněné rozšíření



- Specifikujeme podmínku rozšíření, rozšíření se provede jen pokud je splněna

Sekvenční diagram

- Je to druh diagramu interakcí, který klade důraz na časovou souslednost zpráv
- Na horizontální ose jsou všechny objekty v systému, vertikální reprezentuje čas

Q&A

Děkuji za pozornost



Návrh statického modelu programového systému

Jan Kelnar

„Systém pracuje s daty. Jaká je jejich reprezentace?“

Něco veselého 😊

Během semestru odpadnou následující cvičení

- **13.11.2015**
- **18.12.2015**

I.prezentace

Prezentace analytického řešení –½ zápočtu

- **11.12.2015 v pátek –!!!povinné pro celý tým!!!**
- **cca 30 -45 minut** –logická prezentace studie s UML diagramy
- Studie by měla následně navazovat na implementaci, u implementace se bude také hodnotit rozsah implementované funkčnosti a zda si odpovídá se studií

Musí obsahovat:

- Požadavky + Vision dokument
- Business Proces Model / Business Object Model (BPMN 2.0), 5 netriviálních min.
- Use case + use case realization pomocí vhodných diagramů / popisů
 - Use case realizace nemusí být na úrovni tříd SW, ale s určitou obecností
 - Alespoň 5 sekvenčních / aktivity diagramů pro netriviální use case
- Traceability–mapování požadavků na use case atd.
- Class diagram entit z BOM a příslušné DDL
- Nejlépe také kontextový diagram, diagram architektury, komponent apod.

Co to je statický model PS?

- Popis dat.struktur pomocí entit, atributů, vazeb a integritních omezení
- ER-model nebo OO-diagramy (class diagram)
 - ER model zdůrazňuje vztahy a identity (DB návrh)
 - Class diagram zdůrazňuje hierarchie tříd a operace (OO návrh)
- Další integritní omezení, která nejsou zachycena v diagramech
- Datový slovník

Datový model (konceptuální)

Komponenty:


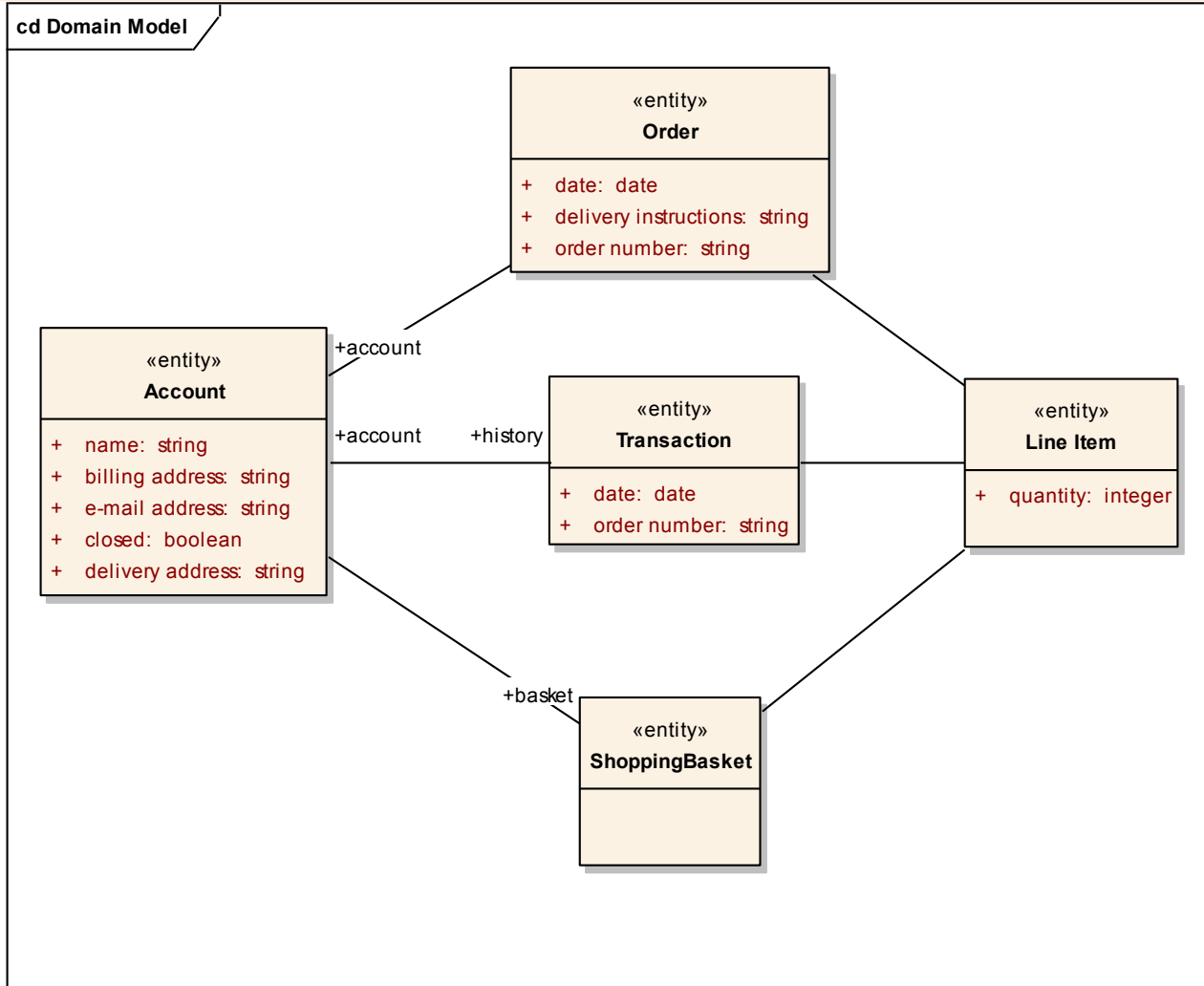
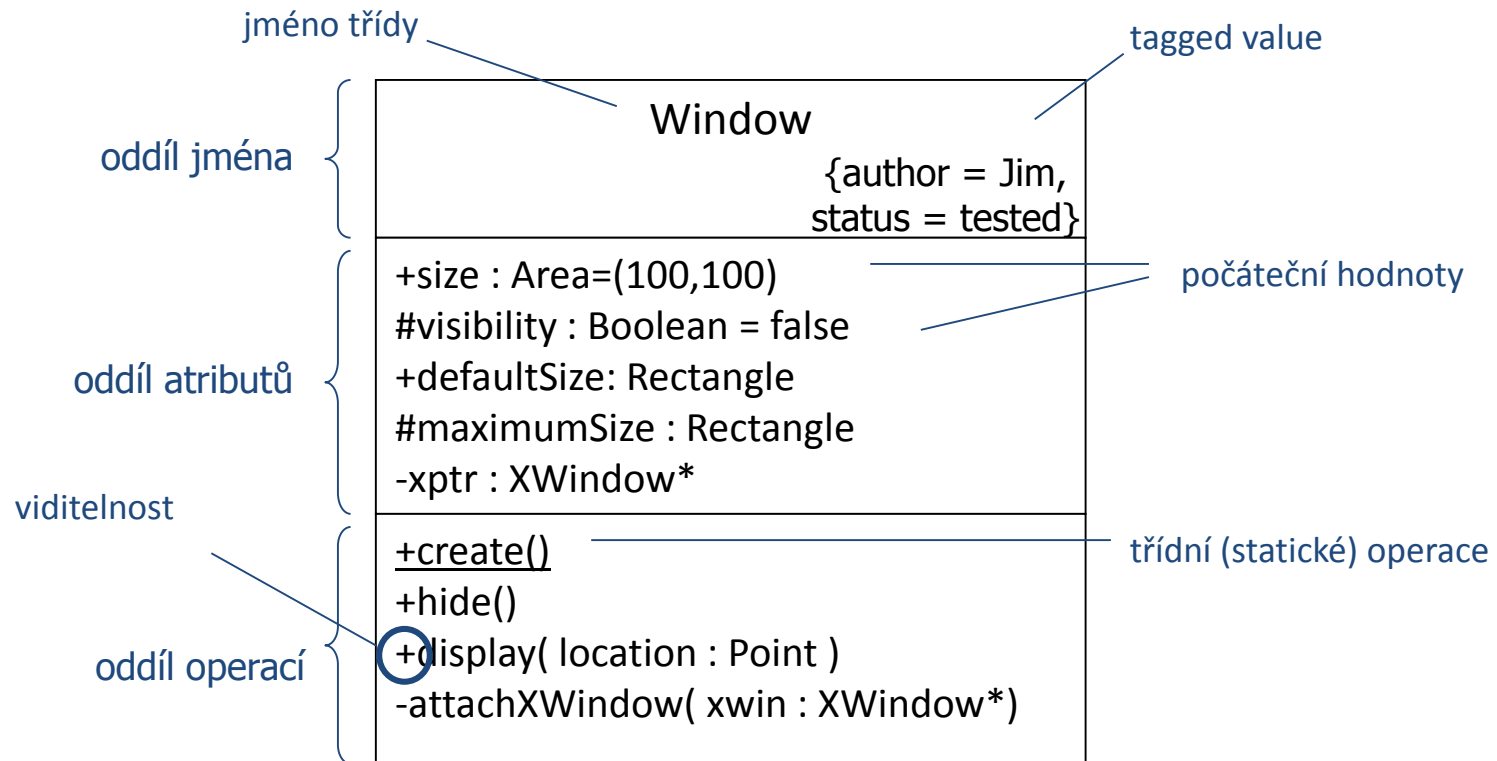
- Typy objektů: entity = rozlišitelný objekt
 - Vztahy: relationships = množiny instancí reprezentujících vztahy mezi objekty
 - Indikace přidružených objektů – pro vztahy o nichž si potřebujeme něco pamatovat
 - Indikace vztahů nadtyp-podtyp (gen-spec), celek-část (whole-part), vyjádření vztahu společný-speciální (dědičnost)
- 

Diagram tříd



UML notace tříd



- Třídy pojmenované UpperCamelCase
- Pojmenovávejte jména podst. jmény
- Nepoužívejte zkratky

Atributy

+size : Area=(100,100)
- address : String [3]

visibility name : type multiplicity = initialValue

- Vše kromě jména je volitelné
 - initialValue je hodnota, kterou atribut dostane při instanciaci
 - Atributy pojmenované lowerCamelCase
 - pojmenovávejte atributy podstatnými jmény
 - nepoužívejte zkratky
 - Atribut může mít
 - stereotyp jako prefix
 - seznam tagged values jako postfix
-

Viditelnost

Symbol	Jméno	Kdo má přístup
+	public	Každý
-	private	Pouze operace uvnitř třídy
#	protected	Pouze operace uvnitř třídy nebo potomci třídy
~	package	Každý ze stejného balíčku či podbalíčku uvnitř třídy

Násobnost (Multiplicita)

- Umožňuje modelovat kolekce věcí

PersonDetails
-name : String [2..*] -address : String [3] -emailAddress : String [0..1]

name se skládá z či více Stringů

address skládá právě ze 3 Stringů

emailAddress je 1 String nebo není nastavena(null)

multiplicita

Operace

+display(location : Point)

visibility name(direction parameterName: parameterType = default, ...) : returnType

seznam parametrů

může být více
tupů
oddělených
čárkou - r1,
r2,... rn

- Operace pojmenovány lowerCamelCase
 - Nepoužívat zkratky a speciální symboly
 - Operace pojmenovány slovesem či slov. frází
- Operace může mít více návratových typů
 - může vracet více objektů
- Operace může mít:
 - stereotyp jako prefix
 - seznam tagged values jako postfix

Druhy parametrů

druh parametru	význam
in	vstup do operace operace nemění jeho hodnotu
out	je do něj uložen výstup operace
inout	slouží jako vstup operace ho může změnit
return	návratová hodnota operace

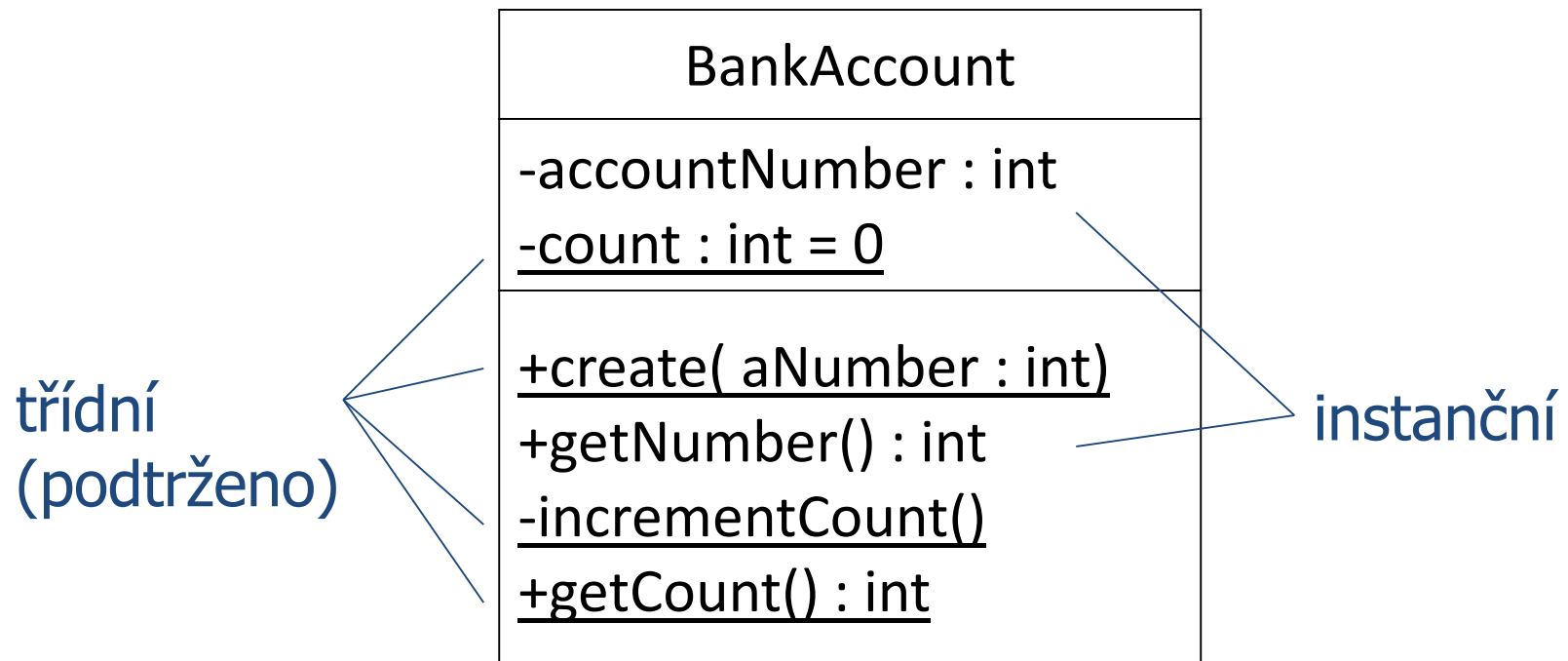
`maxMin(in a: int, in b:int, return maxValue:int , return minValue:int)`

`maxMin(in a: int, in b:int):int,int`

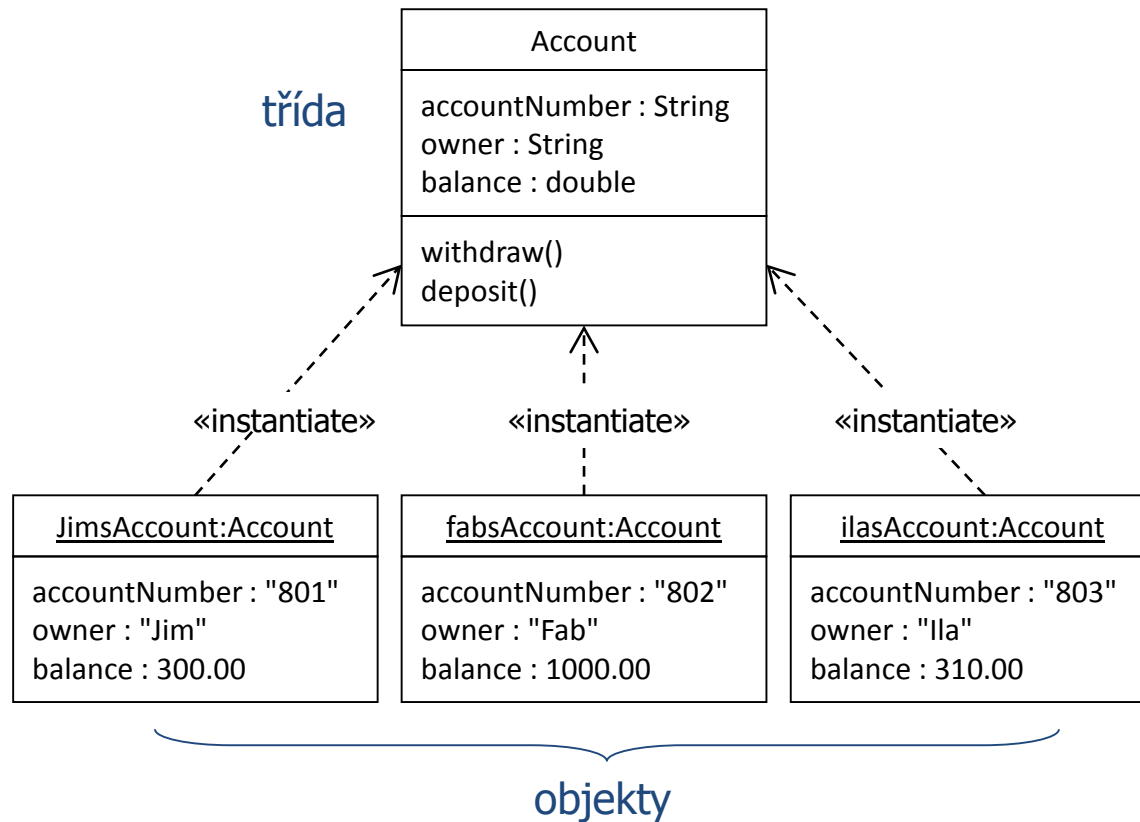
`max, min = maxMin(5, 10)`

Rozsah platnosti

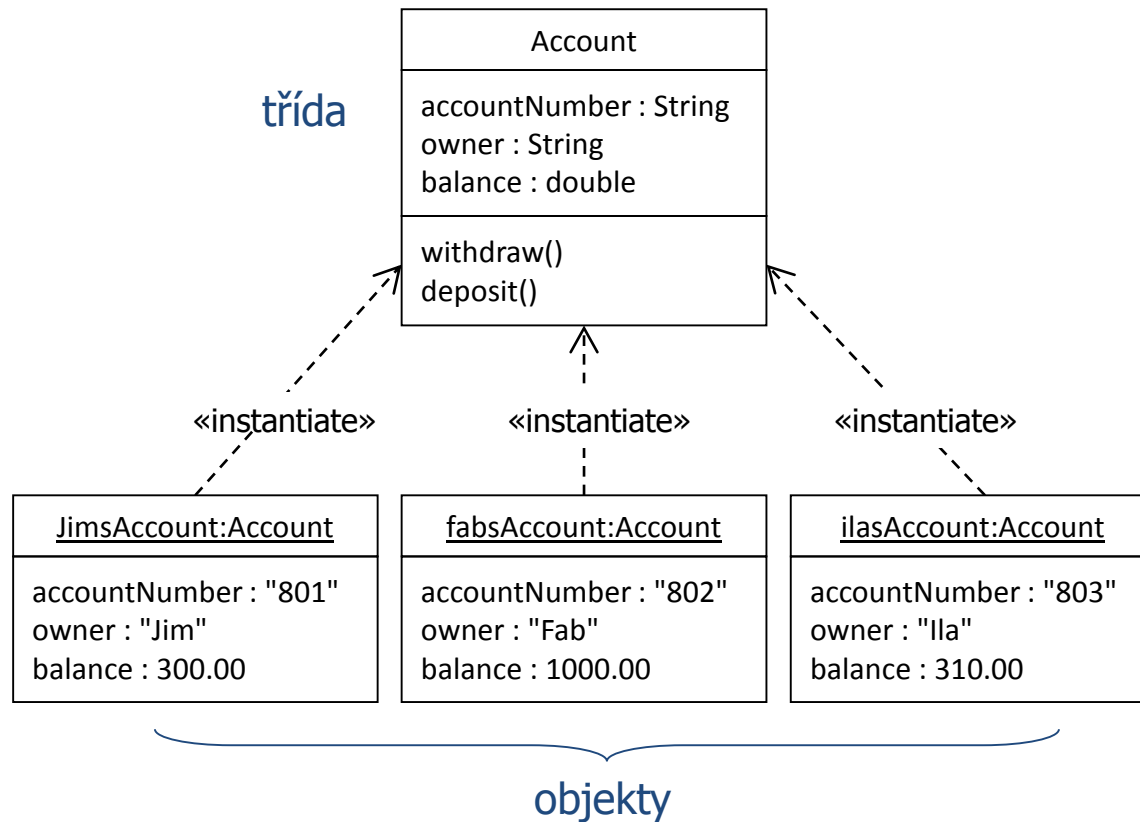
- Dva druhy rozsahu pro atributy a operace:



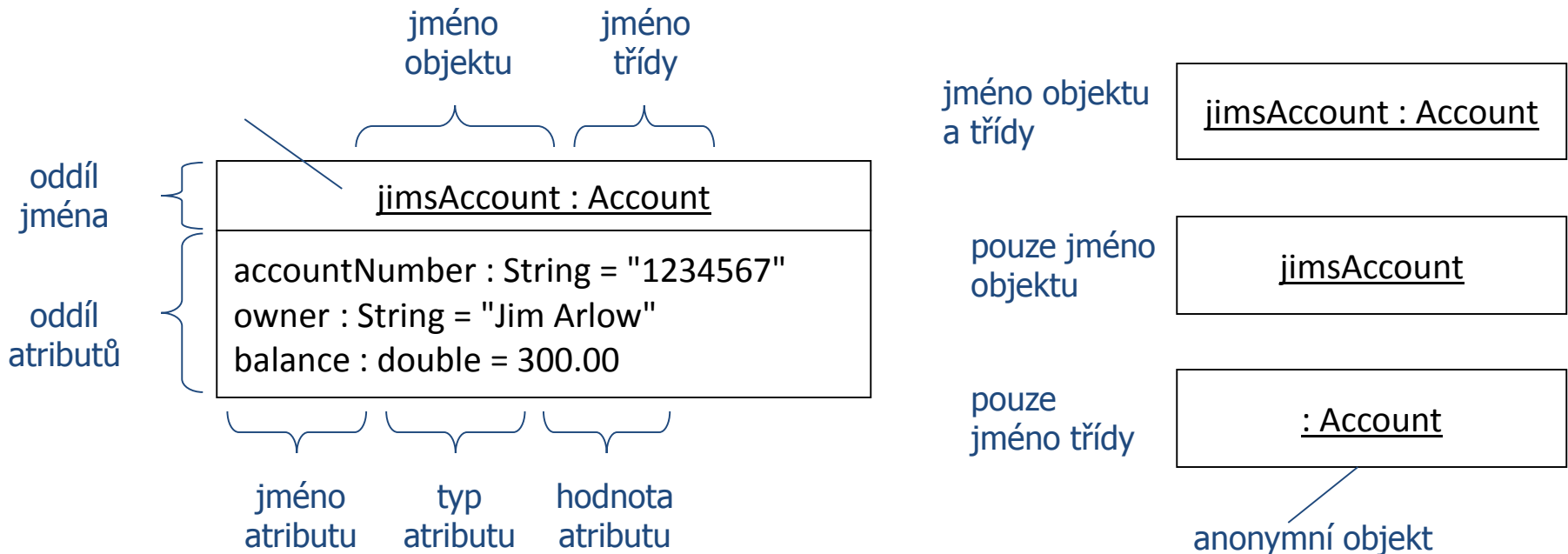
Třídy a objekty



Třídy a objekty



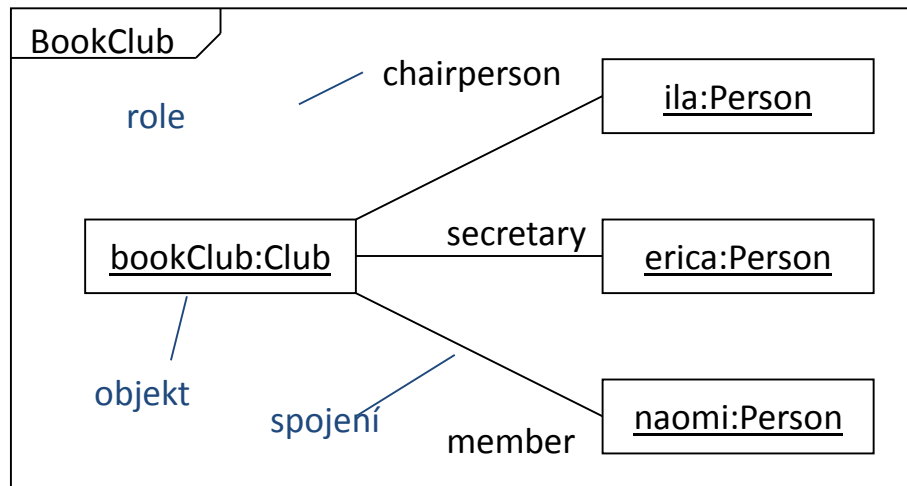
Syntaxe objektu



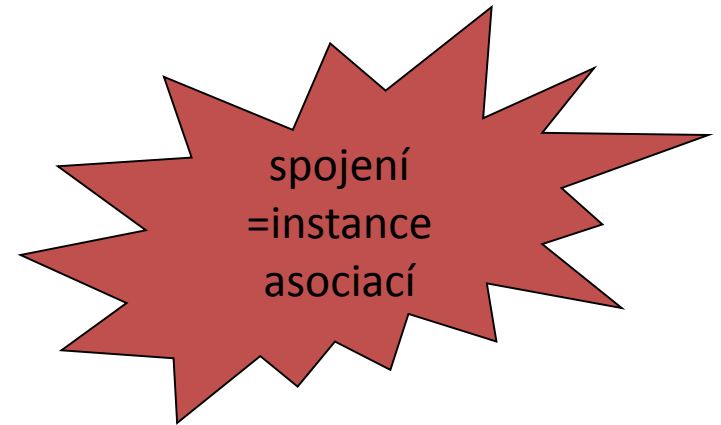
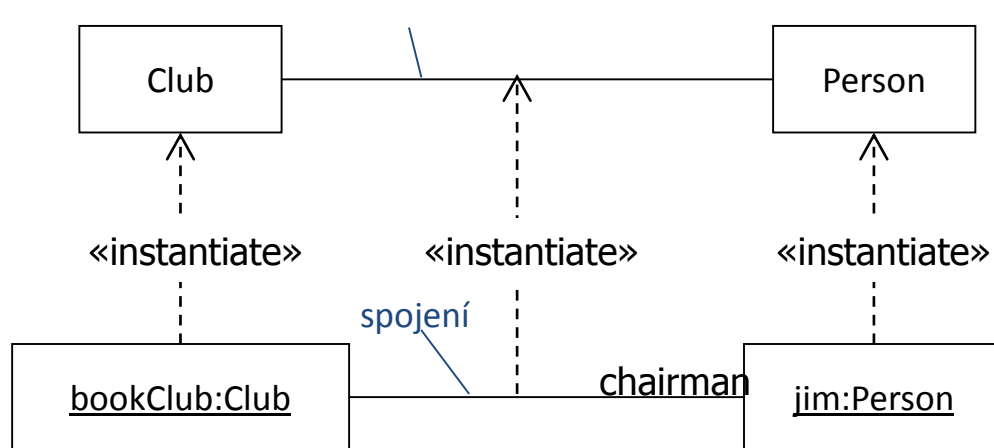
- Všechny objekty dané třídy mají stejnou množinu operací – nezobrazují se u objektů ale u tříd).
- Typy atributů pro přehlednost obvykle vynechány
- Konvence pojmenování:
 - objekty a atributy lowerCamelCase
 - jméno třídy UpperCamelCase

Co je to vztah?

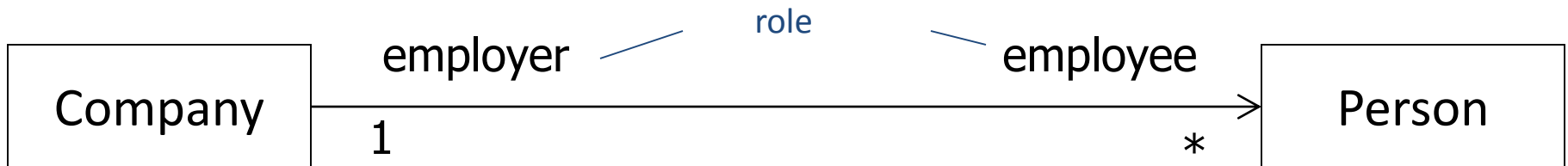
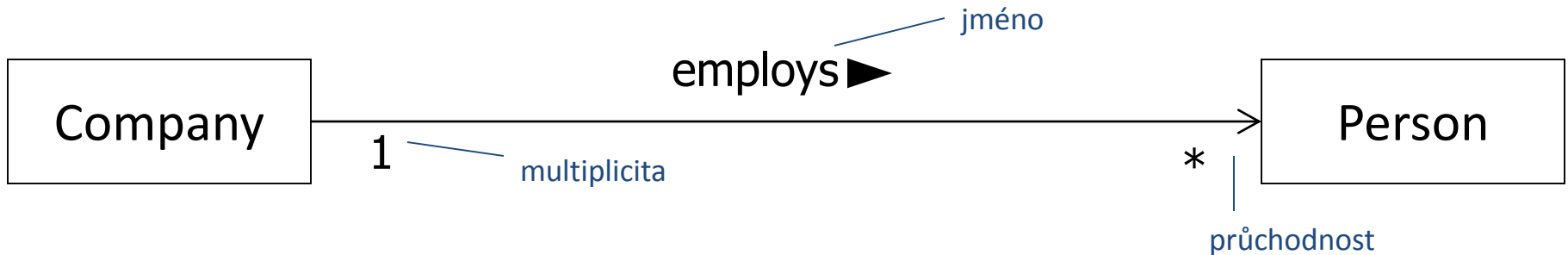
- Mezi instancemi: Spojení
 - Pomocí spojení objekty komunikují
 - Zprávy posílány pomocí spojení
 - Zprávy vyvolávají operace
 - OO prog. jazyky implementují spojení jako reference či pointery



Co je to asociace?

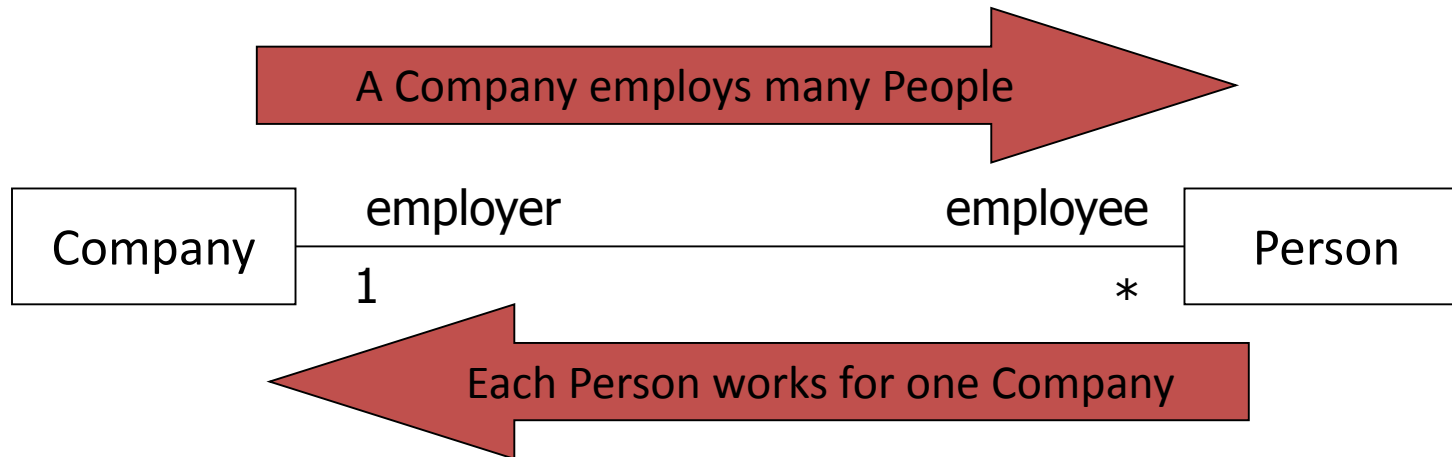


Syntaxe asociace



- pojmenování asociace:
 - jméno
 - role
- asociaci čteme: “Company employs many Person(s)”

Multiplicita



- specifikuje počet objektů participujících na vztahu
- není povinná

multiplicity syntax: minimum..maximum	
0..1	nula nebo 1
1	právě 1
0..*	nula či více
*	nula či více
1..*	1 či více
1..6	1 až 6

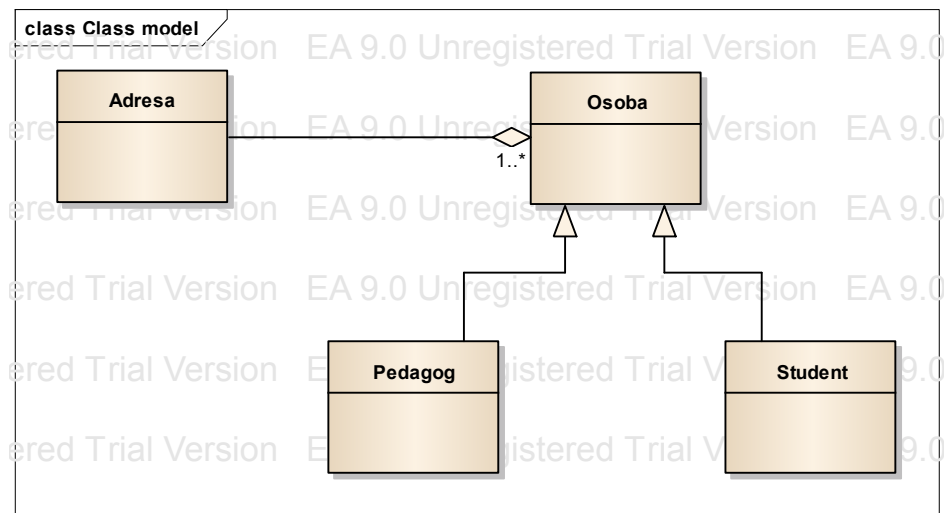
Typy vztahů mezi třídami (a objekty)

- Agregace (obsažení)
- Asociace
- Užití (volání metody, zaslání zprávy)
- Dědičnost
 - Generalizace
 - Specializace

Objektově založené X třídově založené vztahy (kromě dědičnosti jsou to objektově založené vztahy)

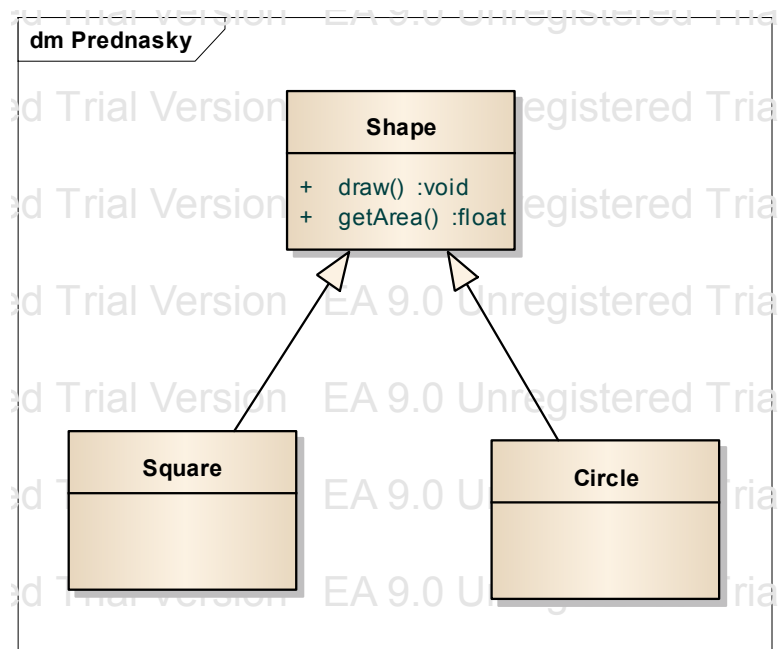
Agregace a generalizace

- **agregace** (*aggregation*) – druh vztahu, kdy jedna třída je součástí jiné třídy - vztah typu celek/část
- **Kompozice** (*composition*) – silnější druh agregace – u kompozice je část přímo závislá na svém celku, zaniká se smazáním celku a nemůže být součástí více než jednoho celku
- **generalizace** (*generalization*) – druh vztahu, kdy jedna třída je zobecněním vlastností jiné třídy (jiných tříd) - vztah typu nadtyp/podtyp, generalizace/specializace - **Dědění**



Dědění / dědičnost

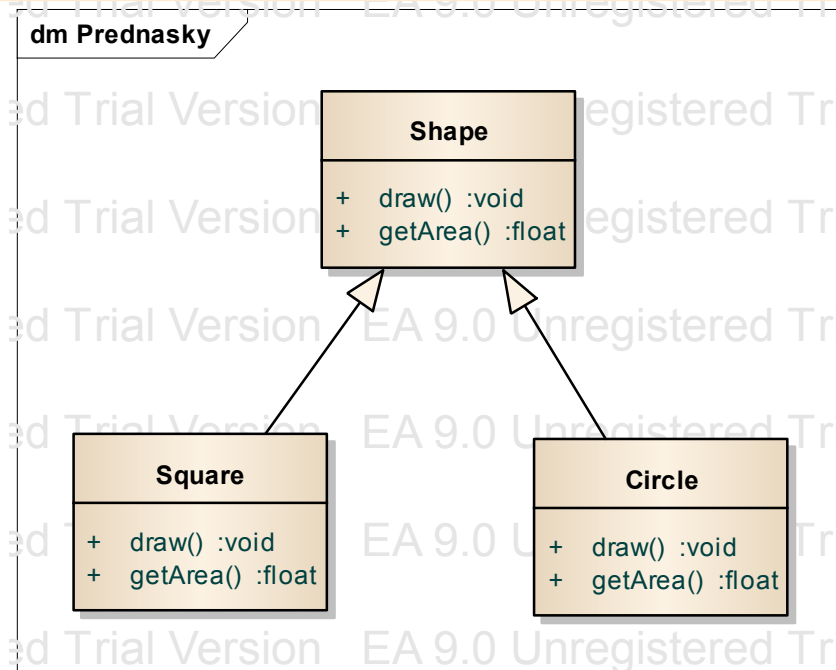
- Podtřída dědí všechny vlastnosti nadtřidy:
 - atributy
 - operace
 - vztah
 - stereotypy, tagy, omezení
- Podtřída přidává nové prvky
- Podtřída může změnit implementaci operací



Co je špatně?

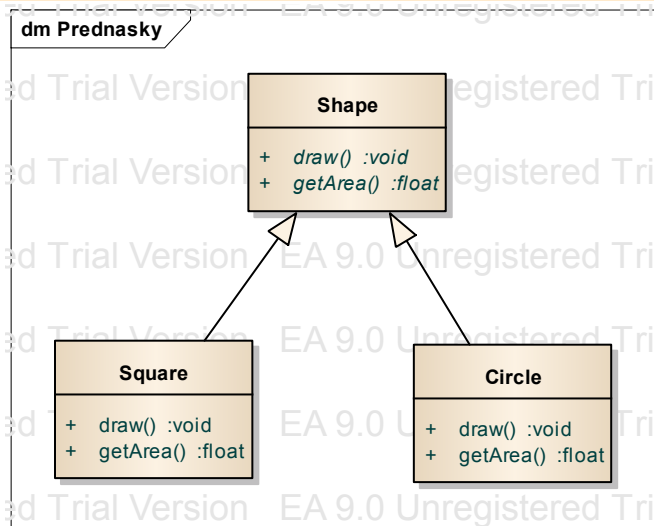


Překrývání



- Podtřída často potřebuje změnit chování definované v předkovi
- Při přetížení signatura v potomkovi musí být identická s předkovou
 - jména parametrů nehrají roli

Abstraktní operace



- Abstraktní operace – signatura bez implementace
- Třída abstraktní pokud má nějaké abstraktní operace
- Od abstraktní třídy není možné vytvořit instanci
- Potomek abstraktní třídy také abstraktní pokud neimplementuje abstraktní operace

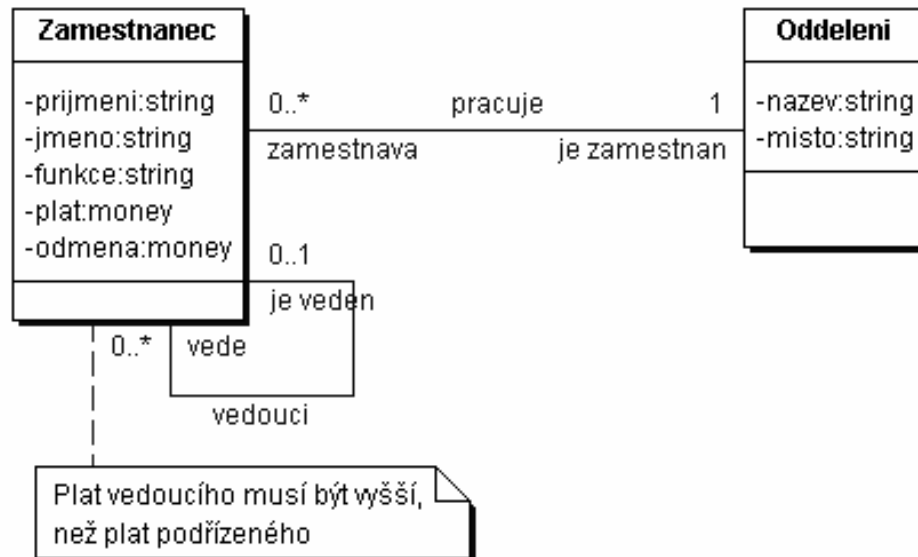
Interface

- Speciální druh (abstraktní) třídy
 - Specifikuje pojmenovanou množinu vlastností
 - Účel – oddělení specifikace funkčnosti od implementace
 - Rozhraní definuje kontrakt, který musí implementující klasifikátory realizovat
-

Konceptuální datový model

- Typy dat (které entity/třídy, atributy...)
- Vztahy mezi nimi
- Další logická omezení (integrity constraints)

Model tříd popisuje i operace, ale ty se často k modelu připojí až v návrhu



Entitní integrita – primary key

Primární klíč je základním prostředkem adresace n-tic relace a platí zde tato pravidla:

- U žádného atributu primárního klíče **nesmí chybět hodnota** (doména)
- Každá n-tice relace musí být **v každém okamžiku identifikovatelná** hodnotou primárního klíče

[illegible]

Referenční integrita – foreign key

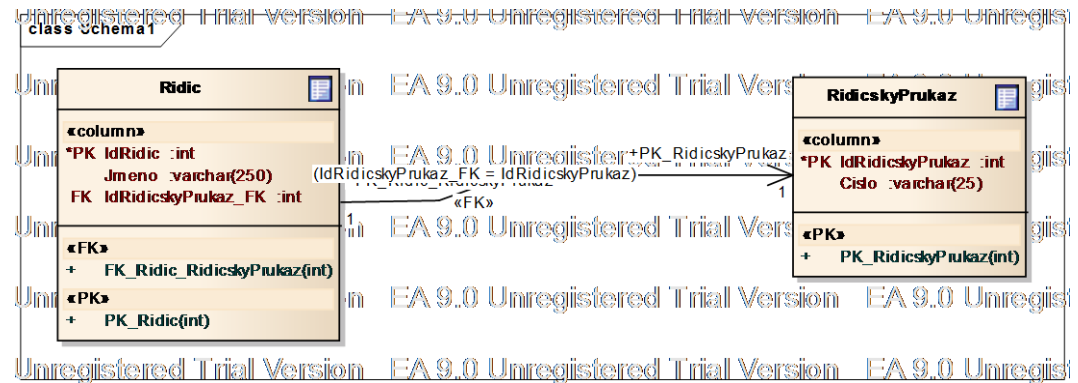
Cizí klíč (foreign key) je atribut, který splňuje tyto nezávislé vlastnosti:

- Každá hodnota je buď plně zadána nebo plně nezadána
- Existuje jiná relace, s takovým primárním klíčem, že každá hodnota cizího klíče = hodnotě primárního klíče

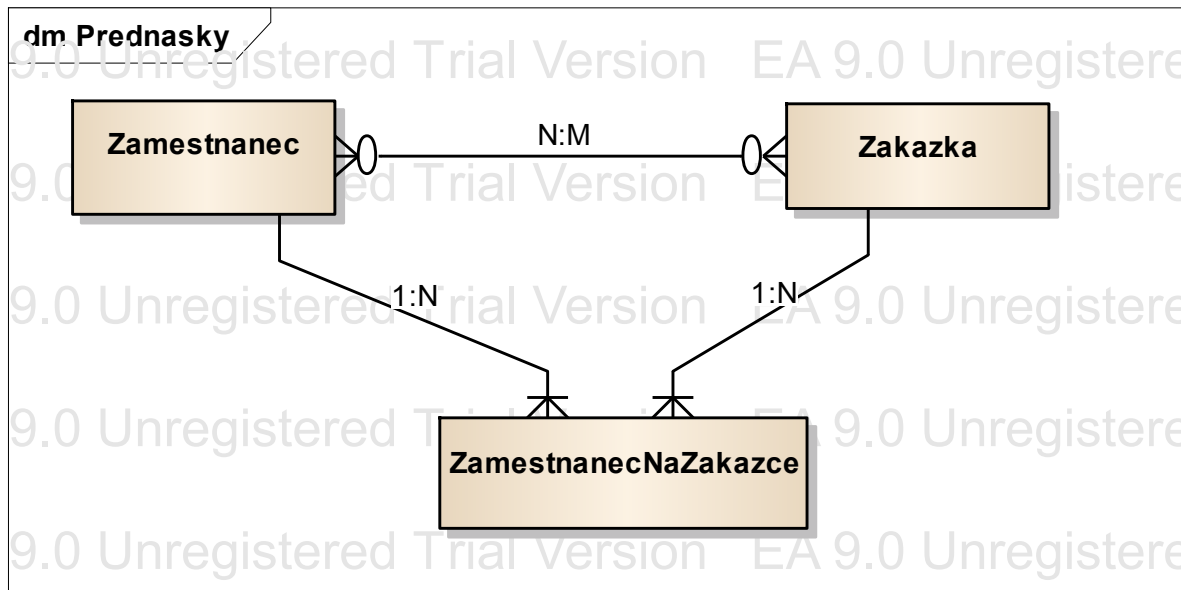
[illegible]

Integritní omezení pro vztahy

- 1:1 (jedna entita?)
- 1:N
- N:M



Dekompozice M:N



Návrh reprezentace dat

Návrh reprezentace dat pomocí relačního databázového systému

Vstup: konceptuální datový model (diagram tříd + popis integritních omezení)

Výstup: logický relační datový model

Výstupem je obecné SQL, při skutečné implementaci návrhu musí být ještě výstup přizpůsoben konkrétnímu stroji

Návrh reprezentace dat

- Pro každou jednoduchou entitu (typ) navrhne tabulku, někdy se používá množné číslo
 - Návrh jmen sloupců pro reprezentaci atributů a odpovídajících domén.
 - Doplníme informace o volitelnosti formátu sloupců.
 - Z nejčastěji používané unikátní identifikace vytvoříme primární klíč, nebo zavedeme nový identifikační sloupec (ID).
 - Pro N-konce vztahů přidáme k tabulce jednoznačné identifikace z tabulky na 1-konci (volitelné vztahy indikují nepovinnost. Současně přidáme odpovídající cizí klíče.
 - Pro každý vztah typu nadtyp/podtyp navrhne reprezentaci (společná tabulka s rozlišovací položkou, samostatné tabulky).
-

Návrh reprezentace dat

- Pro každý vztah typu celek/část navrhne reprezentaci (společná tabulka s rozlišovací položkou, samostatné tabulky).
- Navrhne indexy pro často využívané unikátní kombinace, které nejsou realizovány jako primární klíče.
- Pro generované primární klíče přidáme definice sekvencí pro jejich generování (může být implementačně závislé).
- Navrhne řešení integritních omezení (použijeme deklarativní relační integritní omezení, nebo navrhne „triggery“).


Další postup?

Další postup

- Z datového modelu se snažíme odvodit funkce : Vytvoříme matici CRUD (Create, Read, Update, Delete) a zkoumáme, zda pro každý typ dat existuje odpovídající funkce
- Z datového modelu se snažíme odvodit dynamiku: Pro každý typ dat zkoumáme, zda objekty nevykazují změny stavu

Kontrola datového modelu

Je datový model úplný?

- existuje entita pro každý typ objektu?
 - nejsou zde nadbytečné entity (entity tvořené pouze identifikací, entity s jedinou instancí, apod.)?
 - jsou zde zaneseny všechny vztahy (včetně generalizací a agregací)?
 - nejsou zde odvoditelné vztahy?
 - je model v normální formě?
 - jsou zanesena všechna integritní omezení?
- 

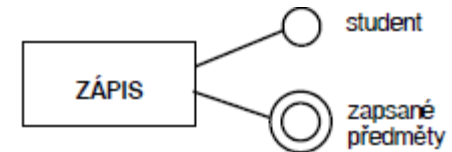
Nadbytečné entity

- entity tvořené pouze identifikací
- entity s jedinou instancí
- entity s vazbou typu 1:1

Dobrou technikou je představit si příklady entit a objektů

Normalizace dat. modelu

- Předpoklad: všechny entity jsou jednoznačně identifikovatelné označenou kombinací atributů a/nebo vztahů
- 1.normální forma: entity neobsahují násobné atributy ani komponované (skupinový atribut) atributy
 - násobný (vícehodnotový atribut): přidání instancí
 - Komponovaný atribut: rozdělení na víc atributů např. příjmení se skládá ze zkratky, názvu
- 2.normální forma (navíc): neklíčové atributy závisí pouze na celém klíči
 - Tzn. entity obsahují pouze takové atributy, které jsou funkčně (významově) závislé pouze na celém klíči
 - Např. evidence předmětů zapsaných studenty (název ani zkratka nejsou funkčně závislé na ID studenta)



Normalizace dat. modelu

- 3.normální forma (navíc): neklíčové atributy nejsou závislé na neklíčových položkách (atributech)

Další vylepšení

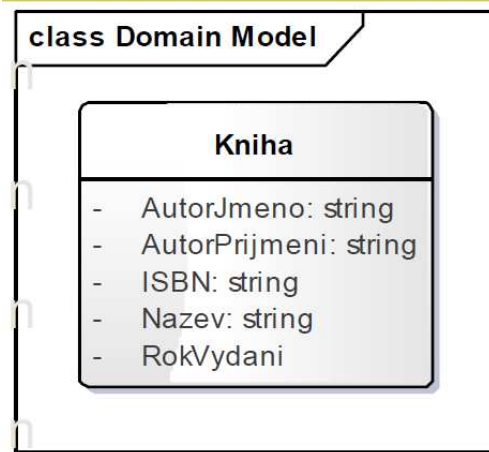
- Nelze doplnit generalizace?
- Nelze doplnit agregace?
- Nelze model vylepšit?

Návrh datového modelu – další

Uložené procedury

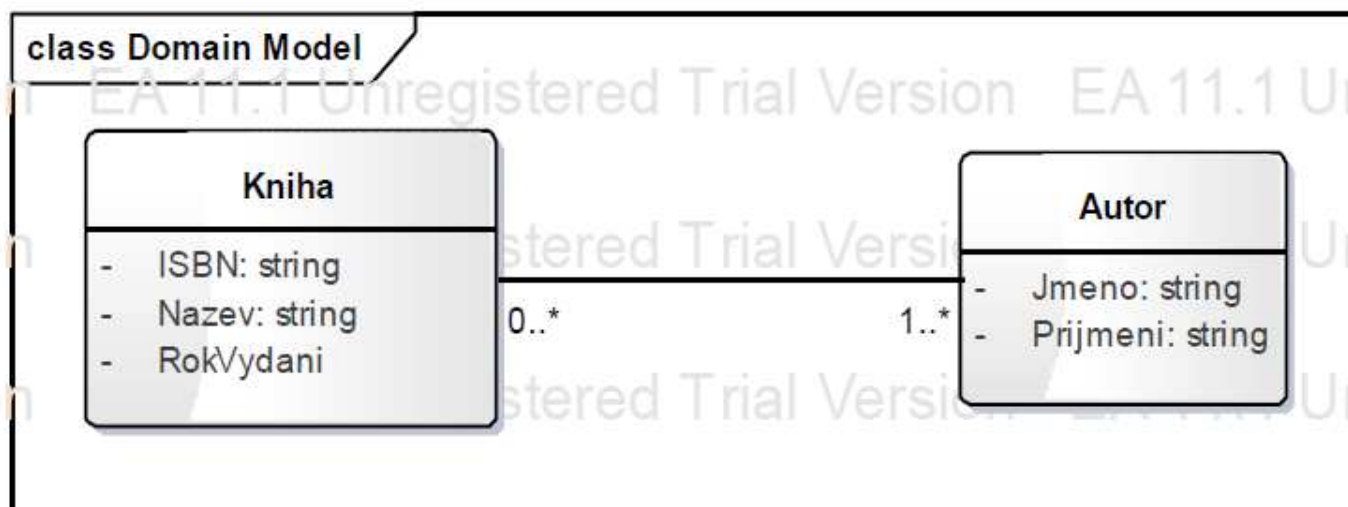
- Vylepšení SQL přidávající možnost používat parametry, deklarovat proměnné, větvit běh příkazů
- „podprogram – objekt databáze“ běžící na SQL serveru
- Speciálním druhem je **trigger**
- Klady:
 - Výkon
 - Nepřerušitelnost – vykonaná jako celek
 - Zabezpečení DB
 - Aplikační logika na DB serveru????

Tvorba doménového modelu - příklad



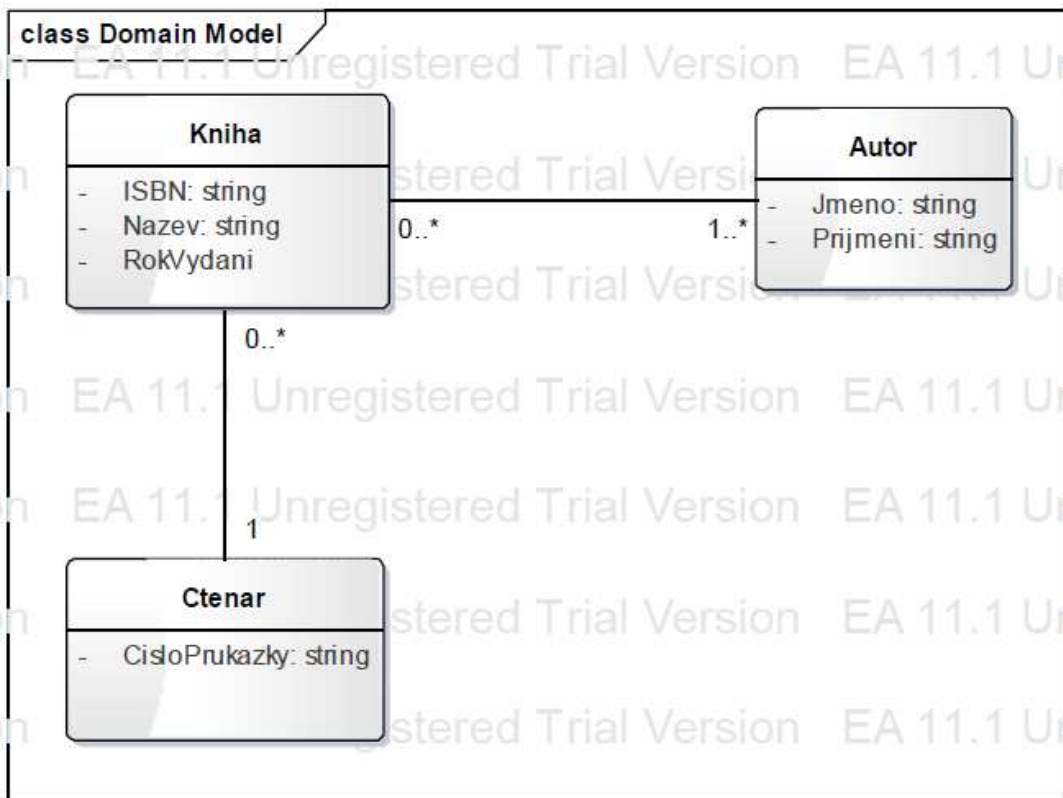
- Duplikace informace o autorovi
- Nesouvisející informace o autorovi a knize v jedné struktuře

Tvorba doménového modelu - příklad



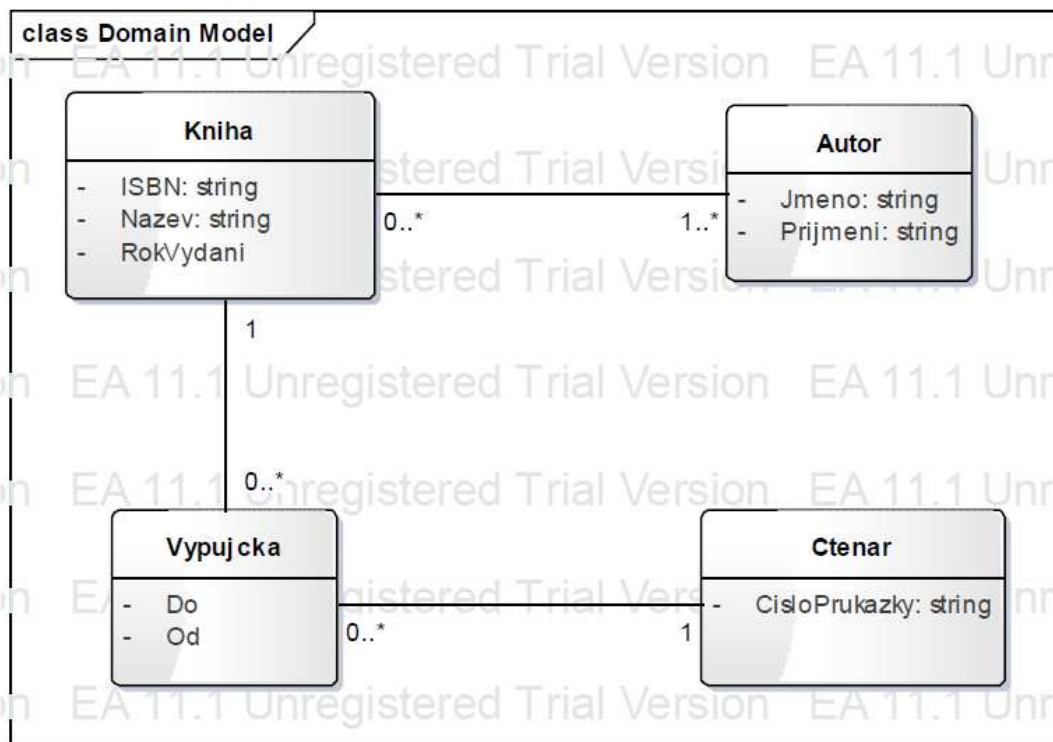
Tvorba doménového modelu - příklad

- Sledování výpůjček
- Nevýhody: nelze sledovat historii



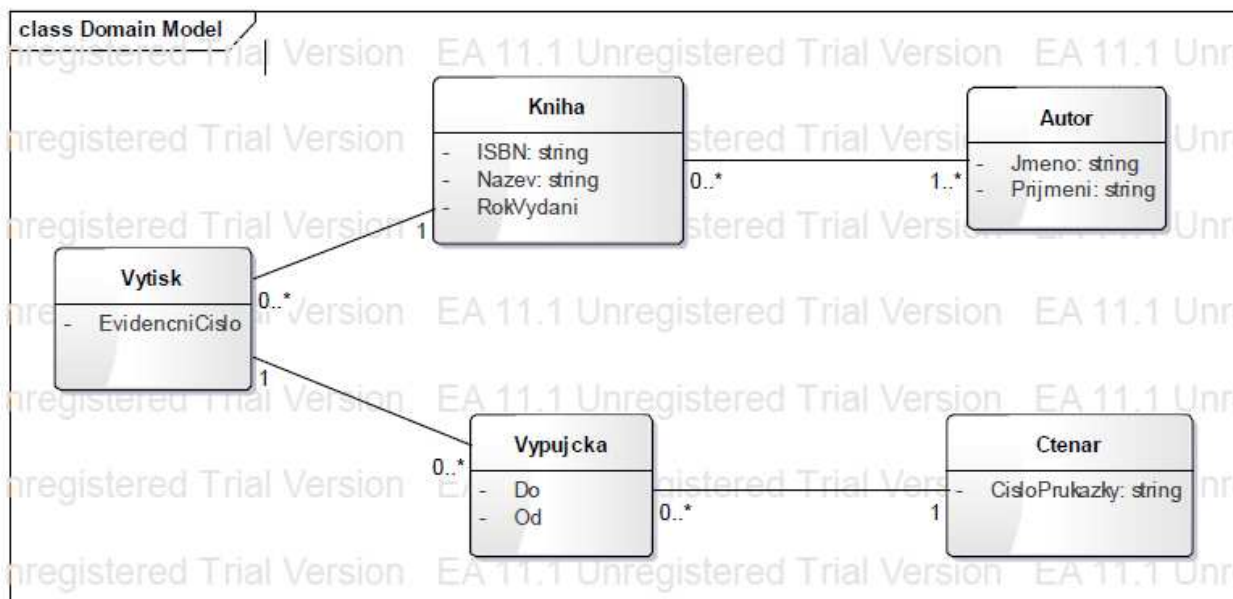
Tvorba doménového modelu - příklad

- Nevýhody: Pouze jeden exemplář knihy



Tvorba doménového modelu - příklad

- Řešení pro více výtisků:
 - Atribut počet kusů
 - Nelze rozlišit jednotlivé výtisky a zachytit vypůjčení konkrétního výtisku
 - Atribut evidenční číslo
 - Duplikování informací o knize u jednotlivých výtisků
 - Nová třída pro výtisk



Q&A

Děkuji za pozornost



Návrh funkčního modelu

Jan Kelnar

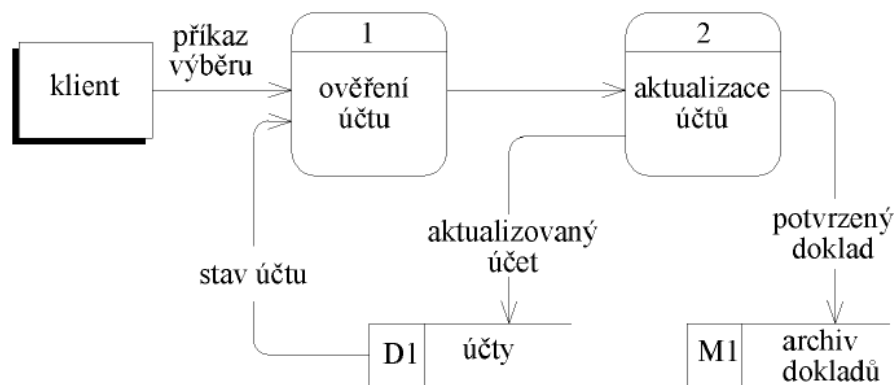
„Systém má poskytovat nějaké funkce, ale jaké to budou?“

Funkční model

- Model jednání – use case (seznam událostí, příp. scénáře případů užití – diagram sekvencí)
 - pokud scénář obsahuje složitější aktivity, pak dekompozice těchto aktivit na popisy jednodušší – mohou to být podrobnější scénáře, diagramy aktivit, hierarchická sada diagramů datových toků (DFD – kontextový diagram + diagramy úrovně 0,1,... + popis)
- minispecifikace elementárních operací
- datový slovník

Funkční modely systémů

- Procesní model (process model)
- Diagram toku práce (workflow diagram)
- Funkční model (sequence model)
- Bublinový diagram (bubble chart)
- Diagram datových toků (DFD)
 - Síťová reprezentace systému pomocí komponent a jejich rozhraní



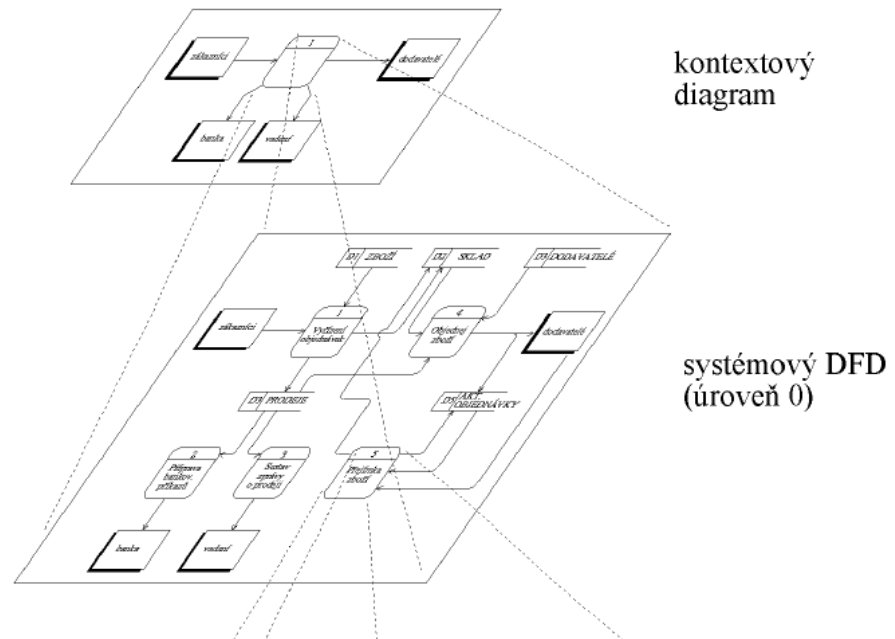
DFD - Definice dle De Marca

- Diagram datových toků je síťovou reprezentací systému
- Systém může být automatizovaný, manuální nebo smíšený
- DFD zobrazuje systém pomocí jeho komponent a určuje rozhraní mezi komponentami
- DFD je grafická reprezentace toku dat mezi procesy (zachycení vazeb funkcí a toků dat)
- Jinými slovy: a) co do systému vstupuje b) jak se to změní c) co z něho vystupuje

DFD

Víceúrovňové DFD

1. Kontextový diagram
2. DFD systému
3. DFD pro každý systémový proces
4. ...



Úrovně DFD

Úroveň 0

- Rozložení systému na několik hlavních subsystémů
- Doporučuje se dodržovat pravidlo 7 tj. jeden diagram zobrazuje maximálně 7 subsystémů

Úroveň 1

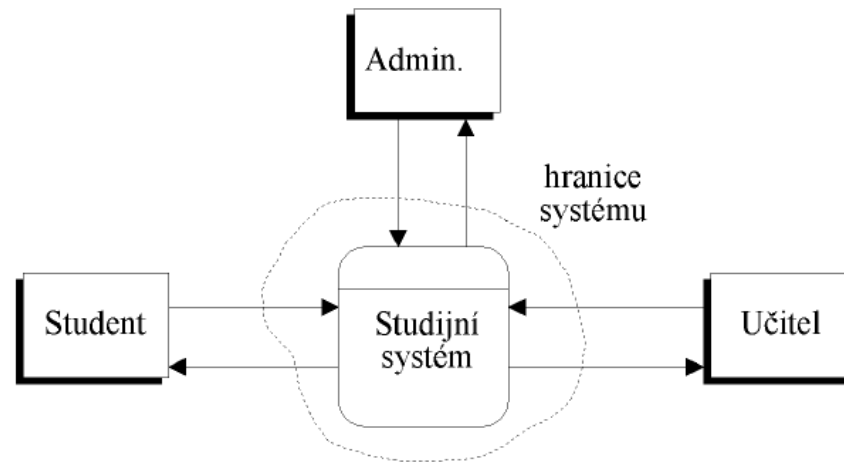
- Podrobnější popis každého subsystému z předchozí úrovně

Úroveň 2

- Jednotlivé funkce z úrovně 1 (při dodržení pravidla 7 by jich nemělo být víc, než 49)
- Rozklad procesů na nižší úrovně pokračuje do té doby dokud není možné je popsat pomocí takzvaných minispecifikací

Kontextový diagram

- Systém jako jediný proces ve spojitosti s „okolním světem“ – model vnějšího chování systému
- Kdo se systémem komunikuje
- Hranice mezi systémem a okolním světem
- Obsahuje pouze aktéry, systém jakožto celek jako jediný proces a datové toky mezi aktéry a systémem



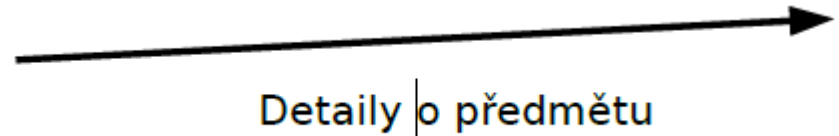
Části DFD

Proces

- transformuje vstupní data na výstupní data
- Název by měl vystihovat podstatu funkce
- Vstupní i výstupní data mají odlišný názor i když jde o data s podobnou strukturou

Datový tok

- tok dat mezi procesy
- u obecného modelu může znamenat např i pohyb hmoty
- Popis v souladu s přenášenými informacemi



Aktér

- Zdroje nebo příjemce dat
- Mohou to být osoby / objekty / jiné systémy mimo sledovanou oblast

Části DFD

Datová paměť

- dočasné uložení dat
- Paměť realizována na PC, ale může to být i kartotéka, objednané zboží na skladě
- Důvod existence: důsledek implementace, požadavek uživatele (data předávaná mezi procesy které pracují v různých čas.obdobích)
- Pravidlo: do datové paměti alespoň jeden vstup a jeden výstup datového toku

Předměty

Pravidla pro DFD

- Všechny procesy musejí mít alespoň jeden datový tok dovnitř a alespoň jeden ven (jeden výstupní tok – perpetuum mobile, jeden vstupní – černá díra)
- Všechny procesy by nějakým způsobem měli modifikovat data, která z nich budou vystupovat
- Datová paměť může být propojená jedině prostřednictvím nějakého procesu
- Datový tok znázorňuje jen přenášená data ne volání funkcí

Pravidla pro DFD

- Datový tok s týmž jménem může být v DFD pouze tehdy pokud znamená tentýž datový tok s týmž obsahem
 - jestliže tok z paměti přenáší celý výskyt dat, nemusí se pojmenovávat
 - přenáší-li se jeden nebo více celých záznamů, pojmenovává se stejně jako datová paměť
 - přenáší-li se pouze část záznamů, pojmenuje se jednoznačně jinak.

Minispecifikace

- Popisují logiku každé z funkcí na poslední úrovni DFD
- Každému elementárnímu procesu z poslední úrovně odpovídá jedna minispecifikace
- Popisuje postup jak jsou vstupní data transformovaná na výstupní

Minispecifikace

Např.

1. Zákazník prohlíží katalog a vybere si zboží k nákupu
2. Zákazník zvolí nákup
3. Zákazník vyplní dodací informace (adresa, expresní nebo standardní dodávka)
4. Systém zobrazí plnou cenu včetně ceny dodání
5. Zákazník vyplní platební informace (číslo kreditní karty)
6. Systém autorizuje platbu
7. Systém potvrdí prodej
8. Systém zašle potvrzovací e-mail zákazníkovi

Alternativy:

- 3a. Uživatel je pravidelným zákazníkem
- 3a1. Systém zobrazí naposled zapamatované dodací a platební informace

Minispecifikace

A1:

IF odběratel je registrován,

THEN,

IF objednávka je správně vyplněna AND všechny druhy výrobků v objednávce existují,

THEN přidej údaje do tabulky objednávka od odběratele.

OTHERWISE, informuj odběratele o chybě v objednávce.

OTHERWISE, informuj odběratele o jeho neexistenci v databázi odběratelů.

A2:

IF všechny výrobky v objednávce jsou rezervovány,

THEN pošli objednávku k dalšímu zpracování oddělení prodeje.

OTHERWISE,

FOR EVERY nezarezervovaný výrobek v objednávce DO:

Zkus najít volný výrobek a rezervuj ho.

IF výrobek není na skladě,

THEN informuj správce.

Vztah DFD a ER

- Výsledek funkční analýzy může ovlivnit již hotovou datovou analýzu

Po naddefinování minispecifikací je nutné:

- Existují všechny atributy?
- Existují tabulky pro uložení mezivýsledků?

Diagram sekvencí – scénář událostí

- Zachycení časového sledu událostí
- Vertikální osa je čas, horizontální jsou objekty a zprávy mezi nimi

Diagram sekvencí – scénář událostí

- Zachycení časového sledu událostí
- Vertikální osa je čas, horizontální jsou objekty a zprávy mezi nimi

Lifelines

- Reprezentuje individuálního participanta

Messages

- Synchronní vs. Asynchronní
- Volání nebo singál
- Kompletní, ztracené nebo nalezené

Diagram sekvencí – scénář událostí

Výtah

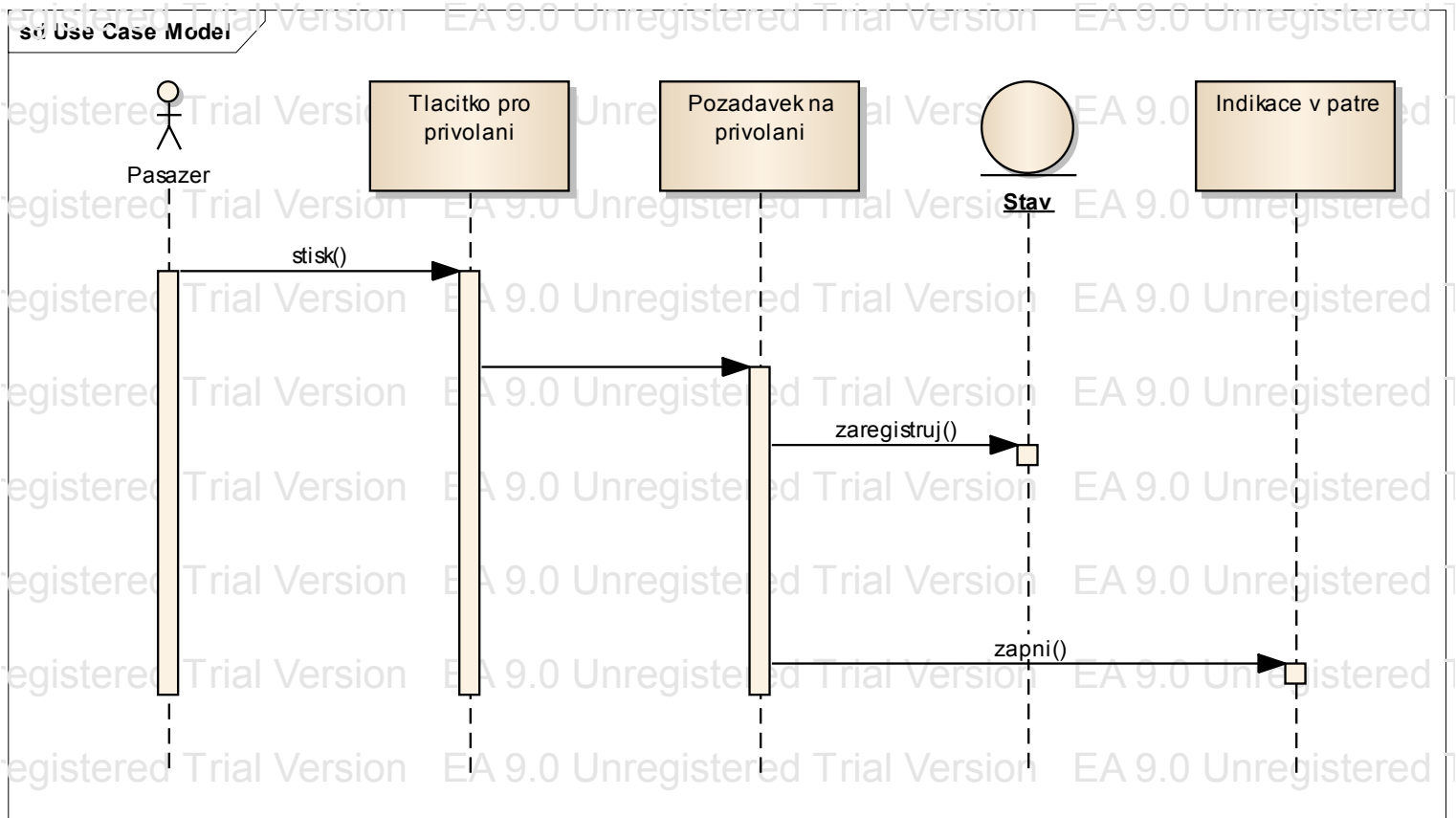


Diagram sekvencí – scénář událostí

Knihovna Objednávky

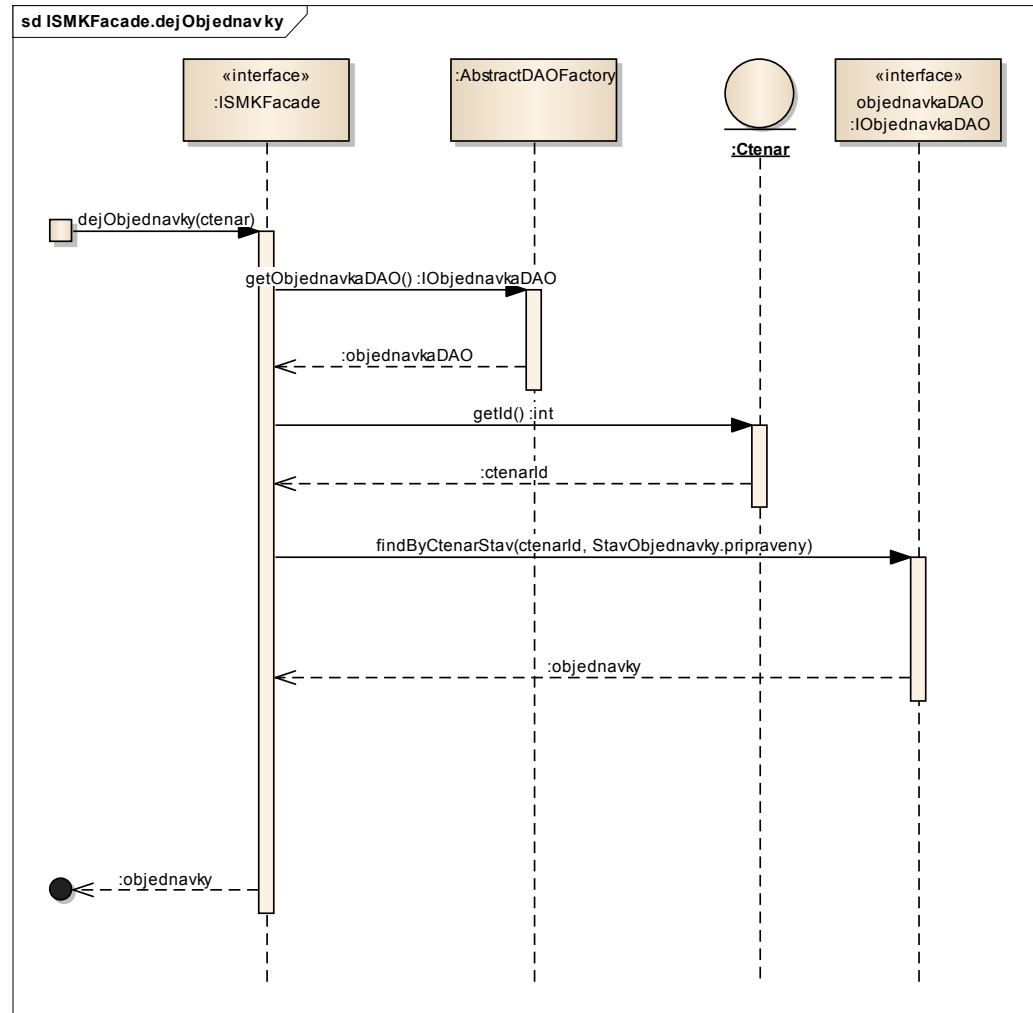


Diagram sekvencí

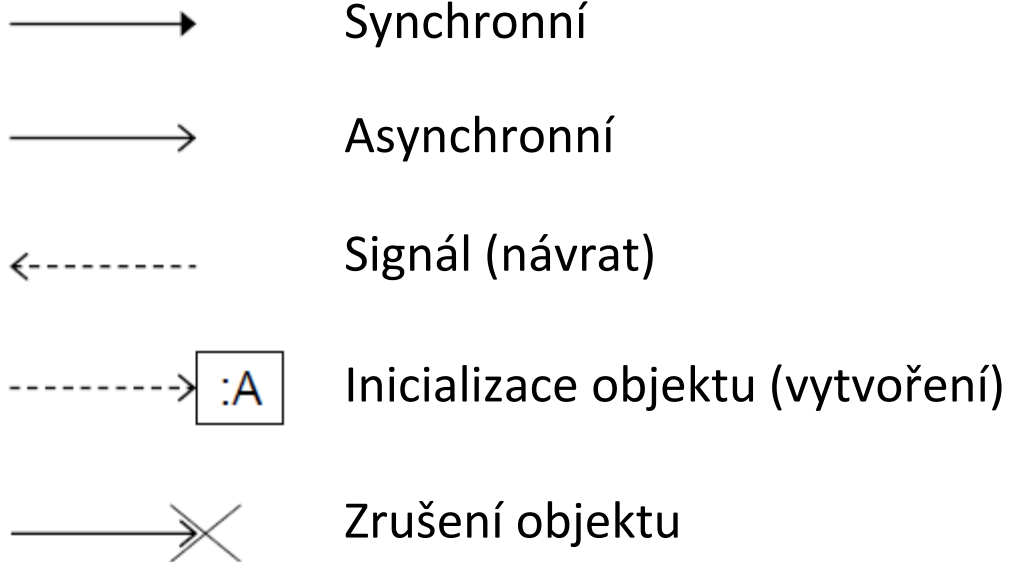


Diagram sekvencí – scénář událostí

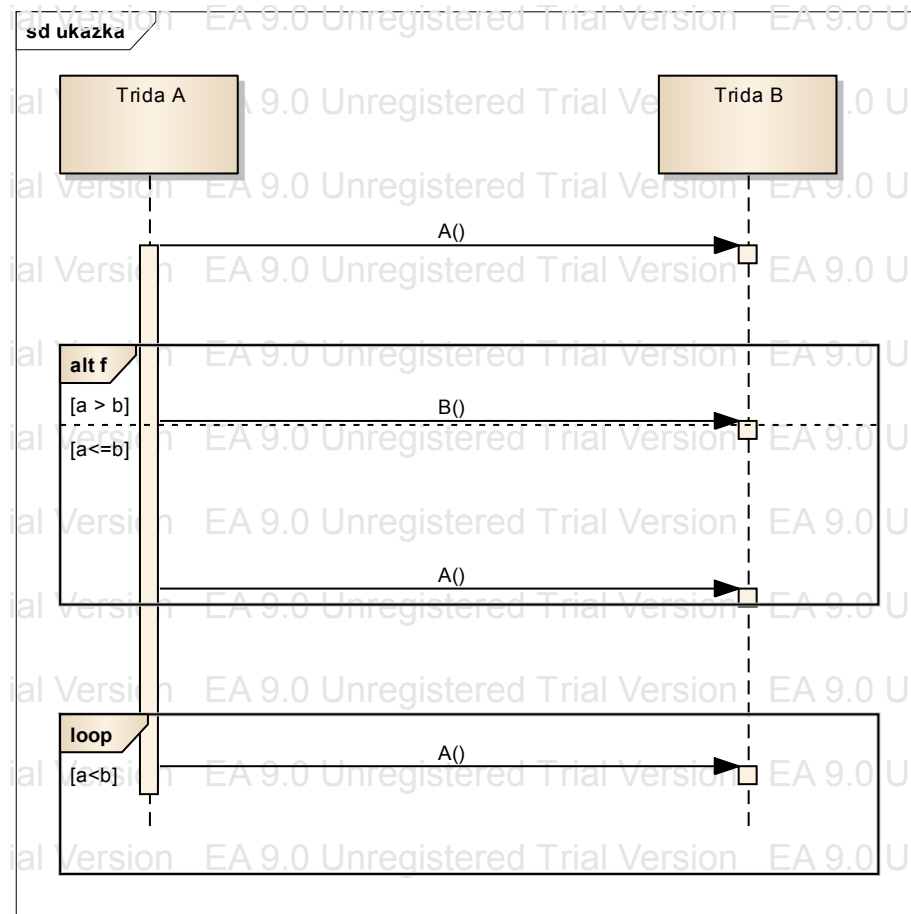


Diagram komunikace

- Stejný účel jako diagram sekvencí
- Zdůrazňují strukturní hledisko
 - Jakým způsobem jsou objekty při spolupráci provázané
 - Čas zachycen pomocí číslování zpráv

Diagram komunikace

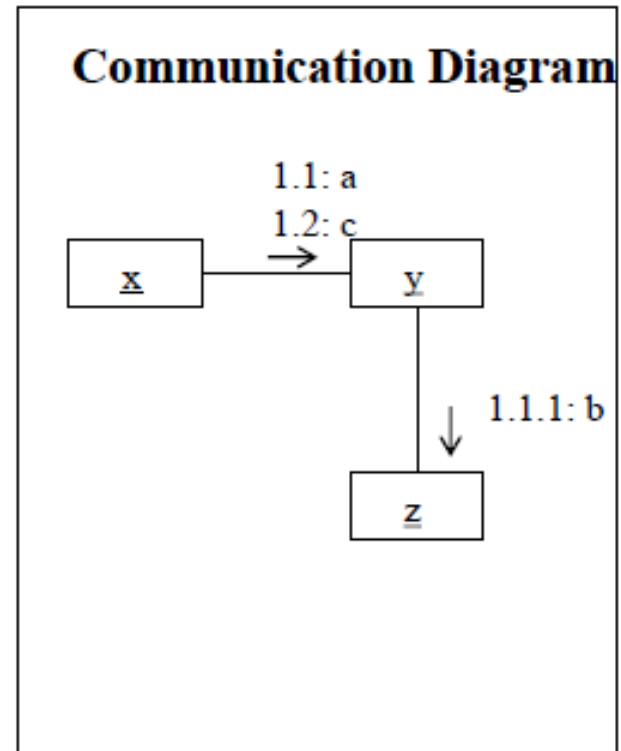
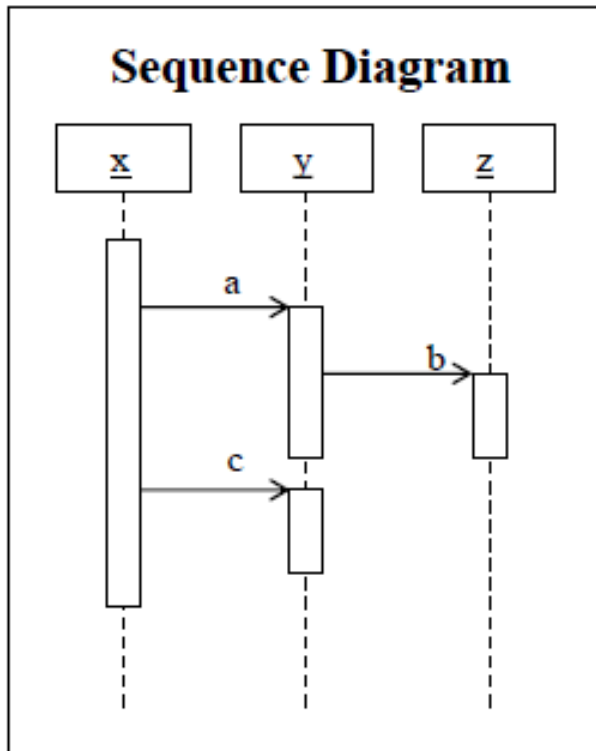


Diagram komunikace - syntaxe

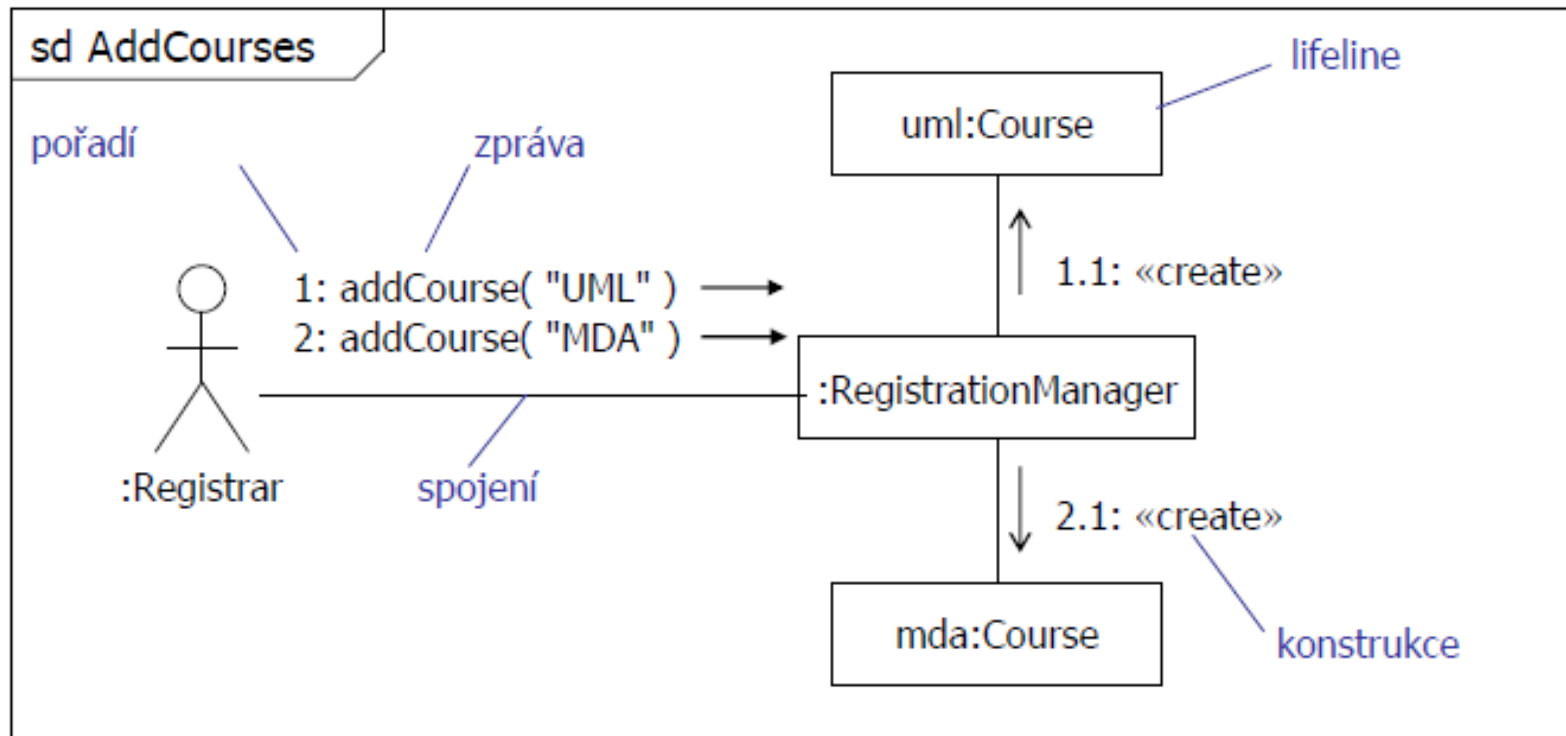


Diagram komunikace - iterace

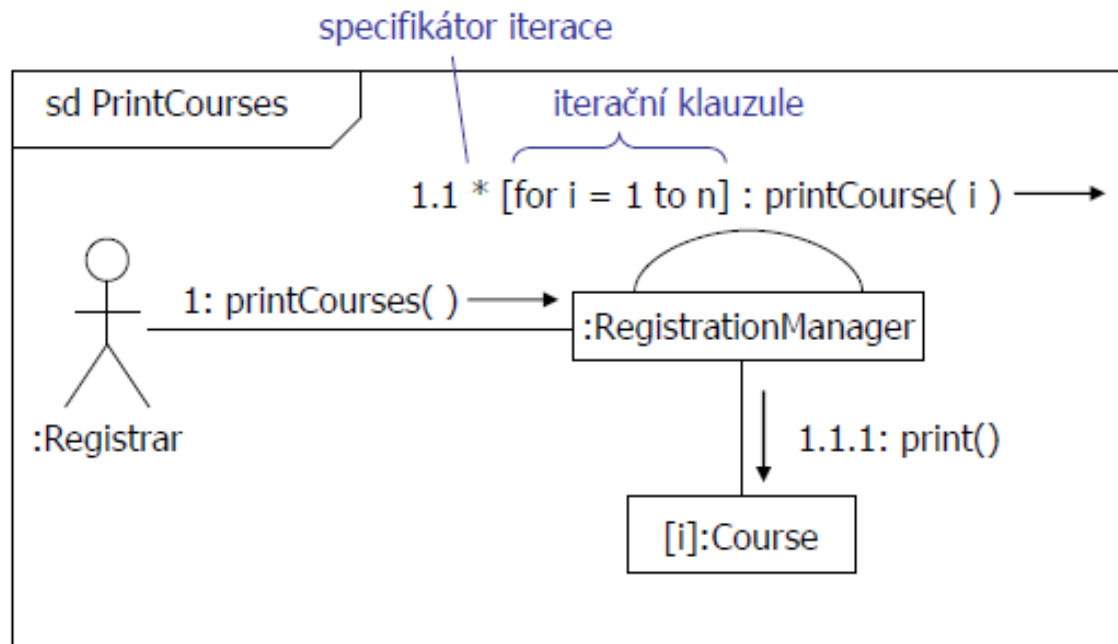
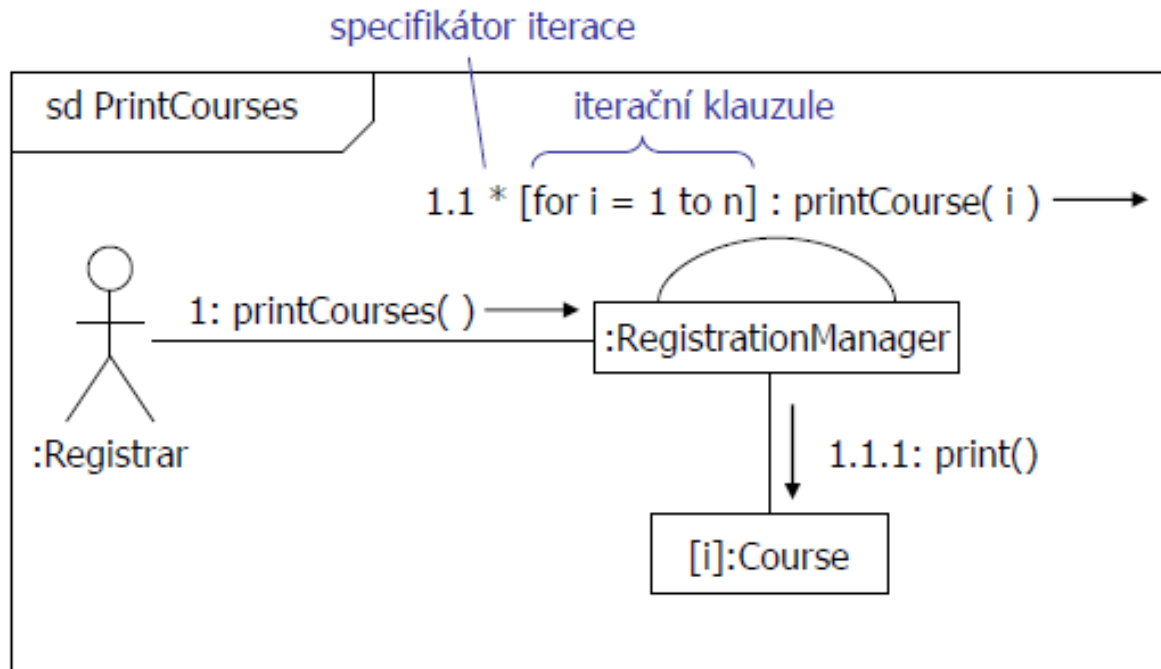


Diagram komunikace - větvení



Kontrola funkčního modelu

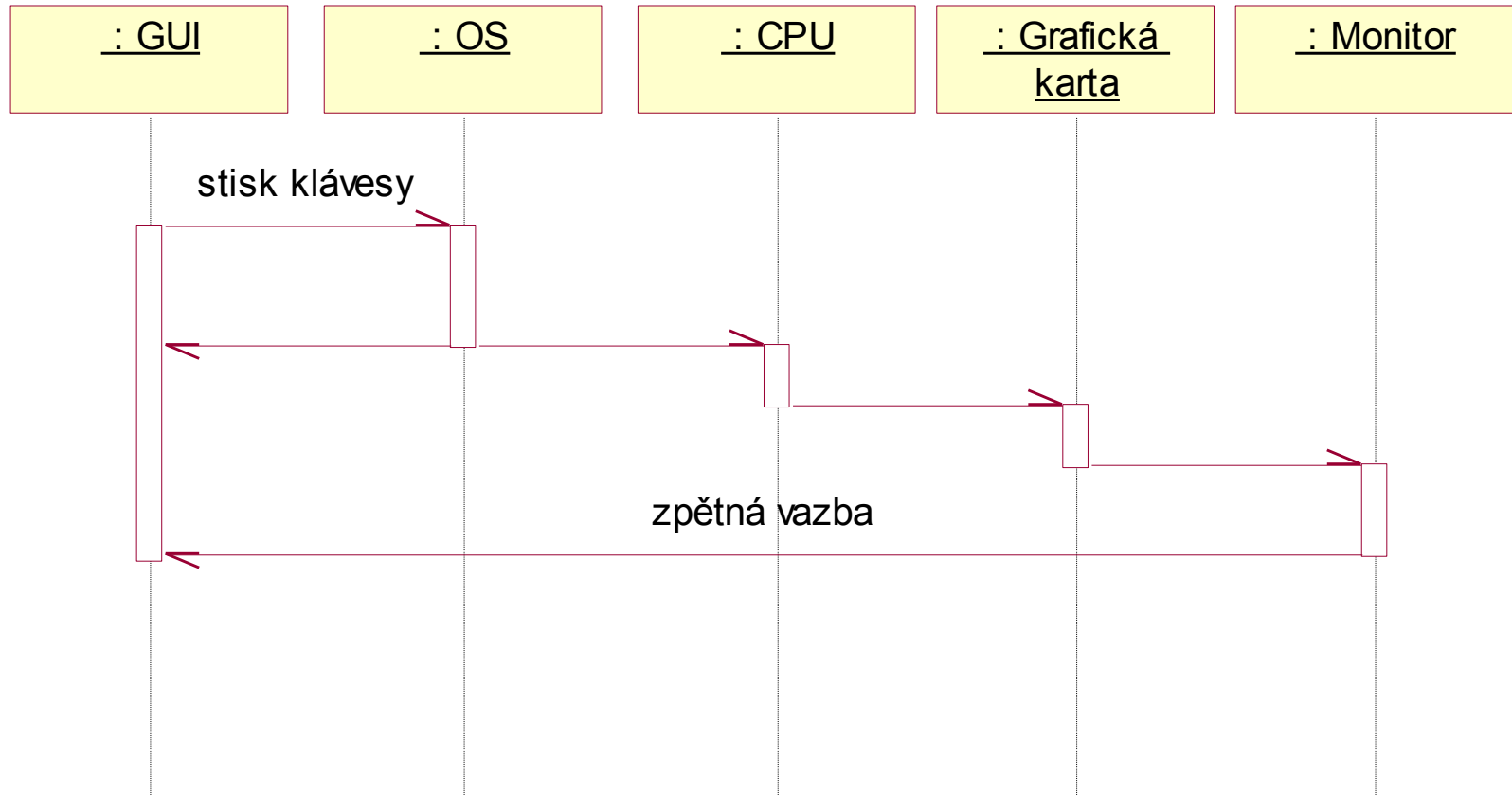
- existuje funkce/metoda pro každou událost?
- každá funkce/metoda musí být popsána dekompozicí, nebo mít minispecifikaci (vstupy a výstupy musí odpovídat)
- nejsou zde nadbytečné funkce/metody?

Příklady - opakování

Pro příklad operačního systému počítače vytvořte sekvenční diagram představující stisk klávesy v textovém editoru, který způsobí zobrazení tohoto znaku na obrazovce. Předpokládejte interakce mezi těmito objekty:

- GUI (grafické uživatelské rozhraní)
- OS (operační systém)
- CPU (procesor)
- Grafická karta
- Monitor

Příklady - opakování

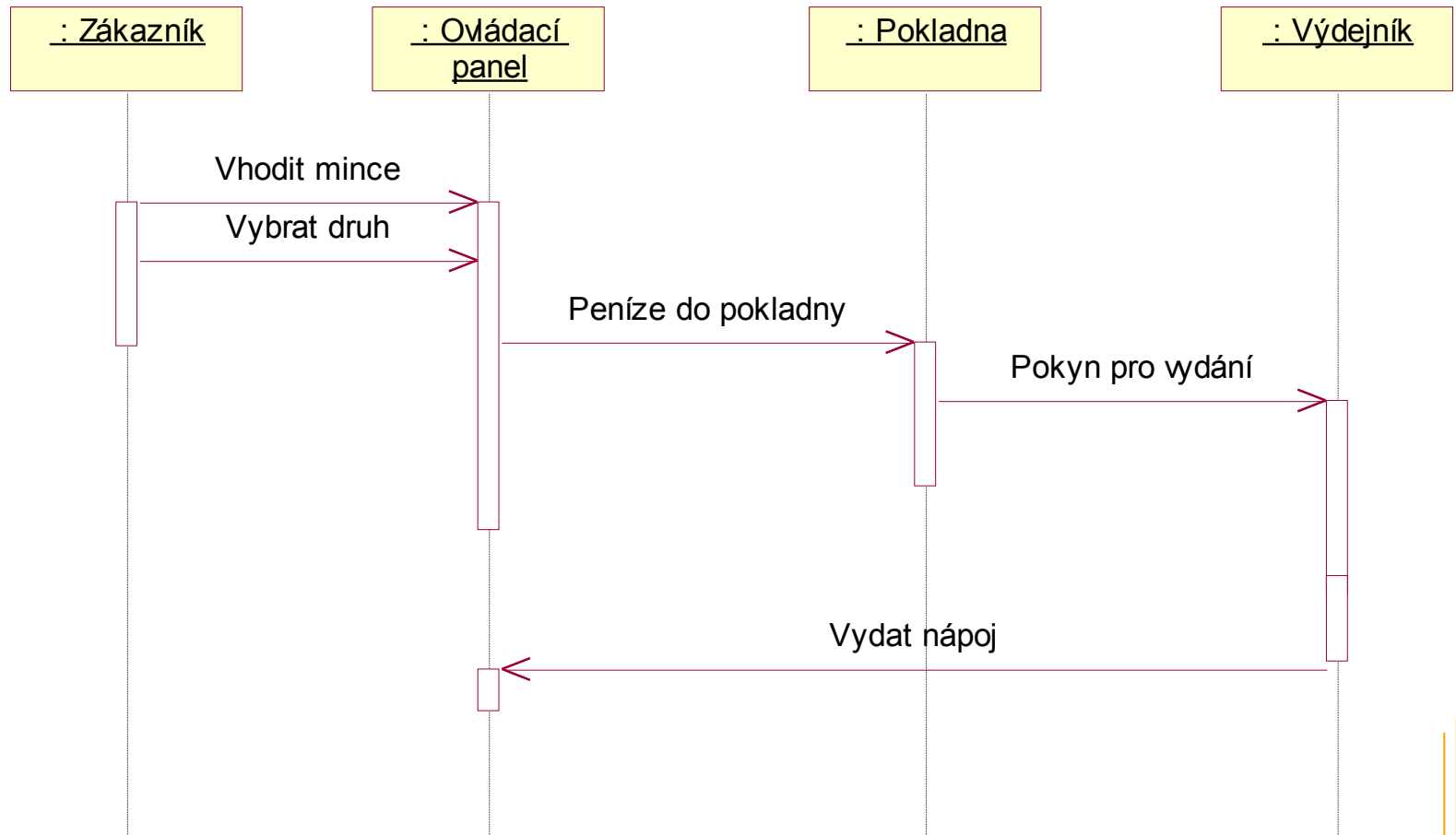


Příklady - opakování

Vytvořte sekvenční diagram pro nákup nápoje v automatu. Předpokládejte, že se automat skládá z těchto čtyř objektů:

- Zákazník
 - Ovládací panel
 - Pokladna
 - Výdejník
-
1. Zákazník vhodí mince do otvoru v ovládacím panelu
 2. Zákazník si vybere druh nápoje
 3. Peníze propadnou do pokladny
 4. Pokladna předá na výdejník pokyn pro vydání nápoje
 5. Výdejník vydá nápoj
-

Příklady - opakování

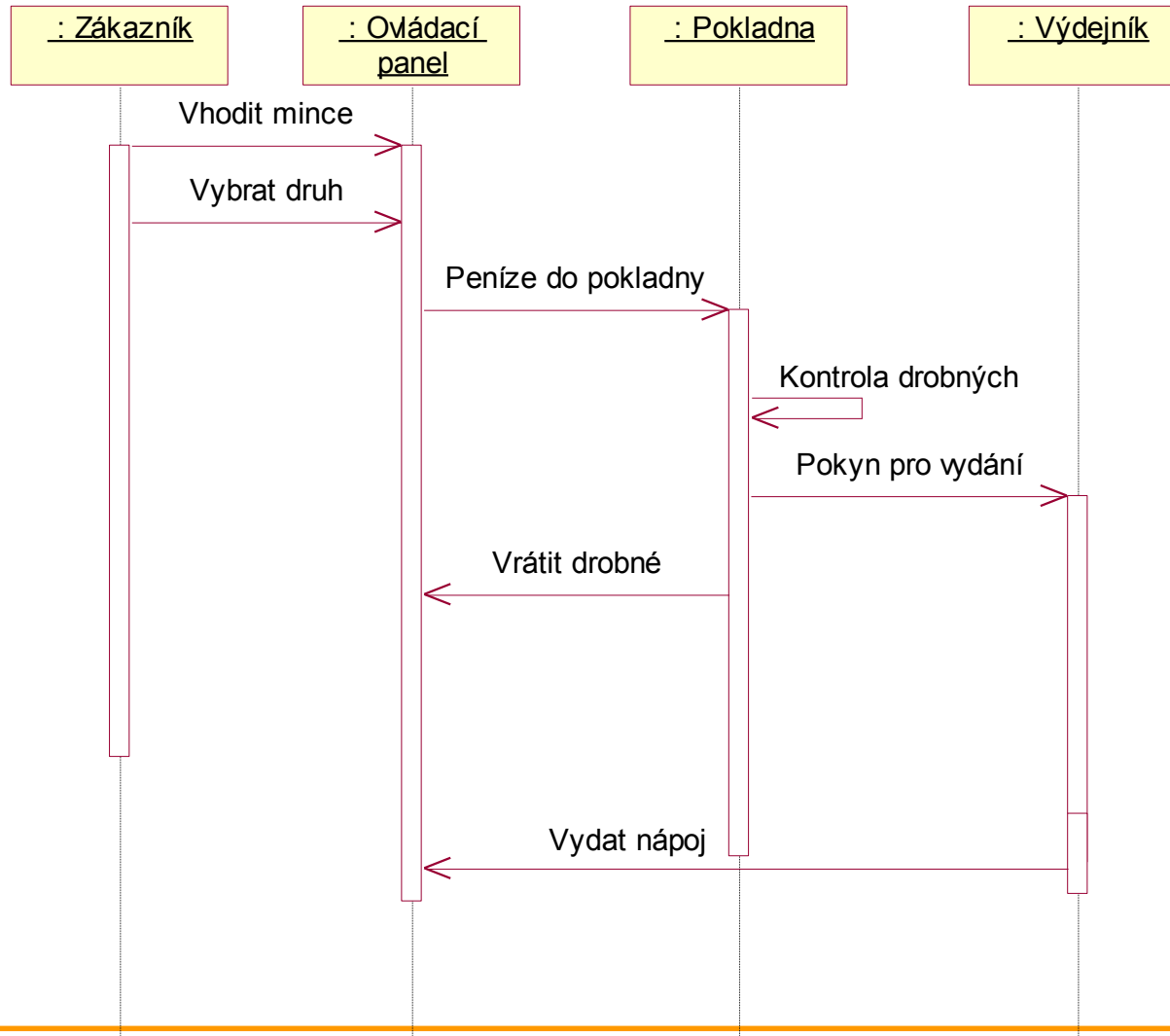


Příklady - opakování

Pro případ nápojového automatu uvažujte scénář, v němž zákazník nemá správný obnos.

1. Pokladna zkontroluje, zda množství vhozených mincí odpovídá ceně nápoje
 2. Pokud je vhozená částka vyšší, vyplatí pokladna rozdíl
-

Příklady - opakování

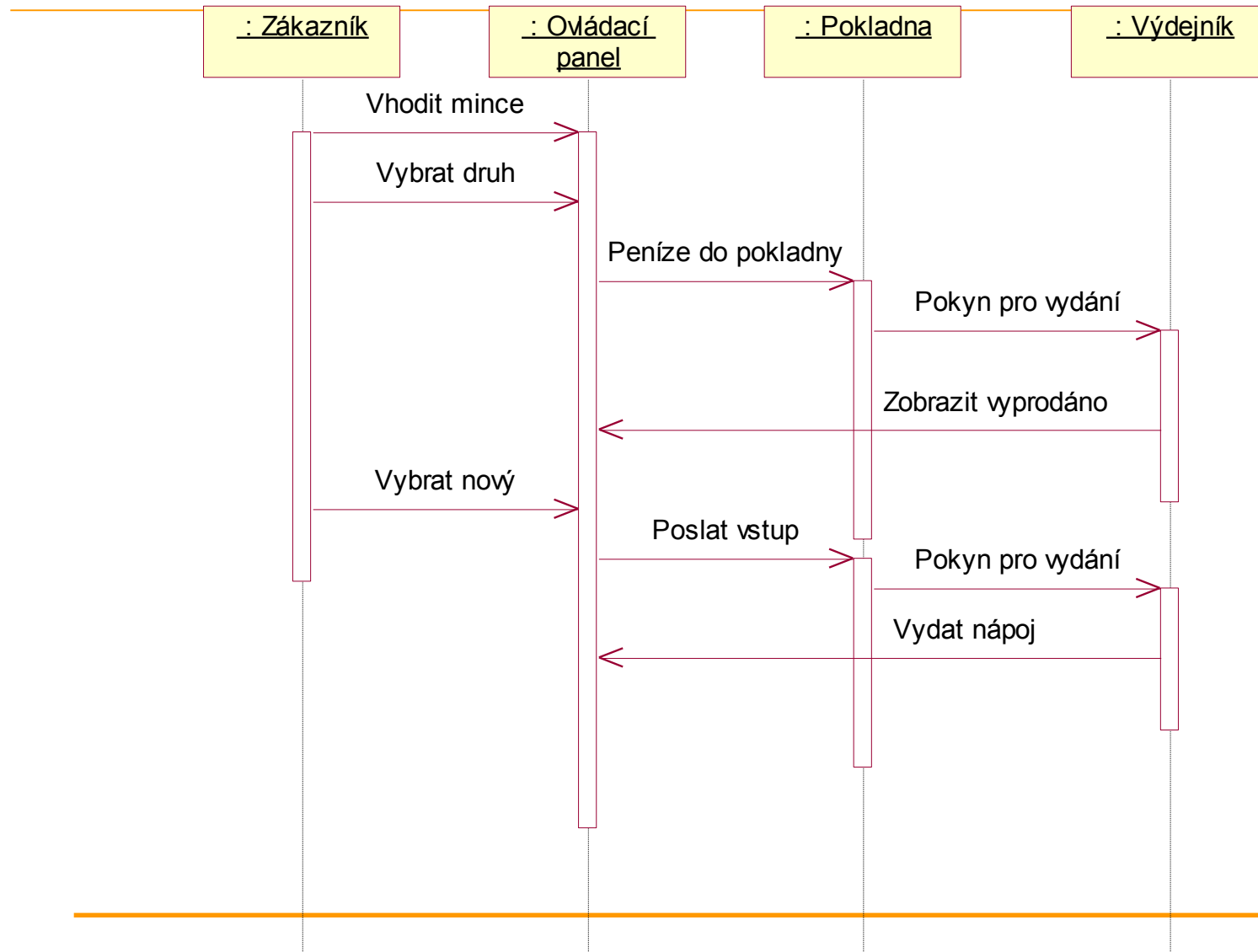


Příklady - opakování

Pro případ nápojového automatu uvažujte scénář, kdy je požadovaný druh nápoje vyprodán.

1. Poté, co zákazník zvolí vyprodaný druh, automat vypíše zprávu *Vyprodáno*
 2. Automat čeká na volbu jiného druhu nápoje
-

Příklady - opakování



Příklady - opakování

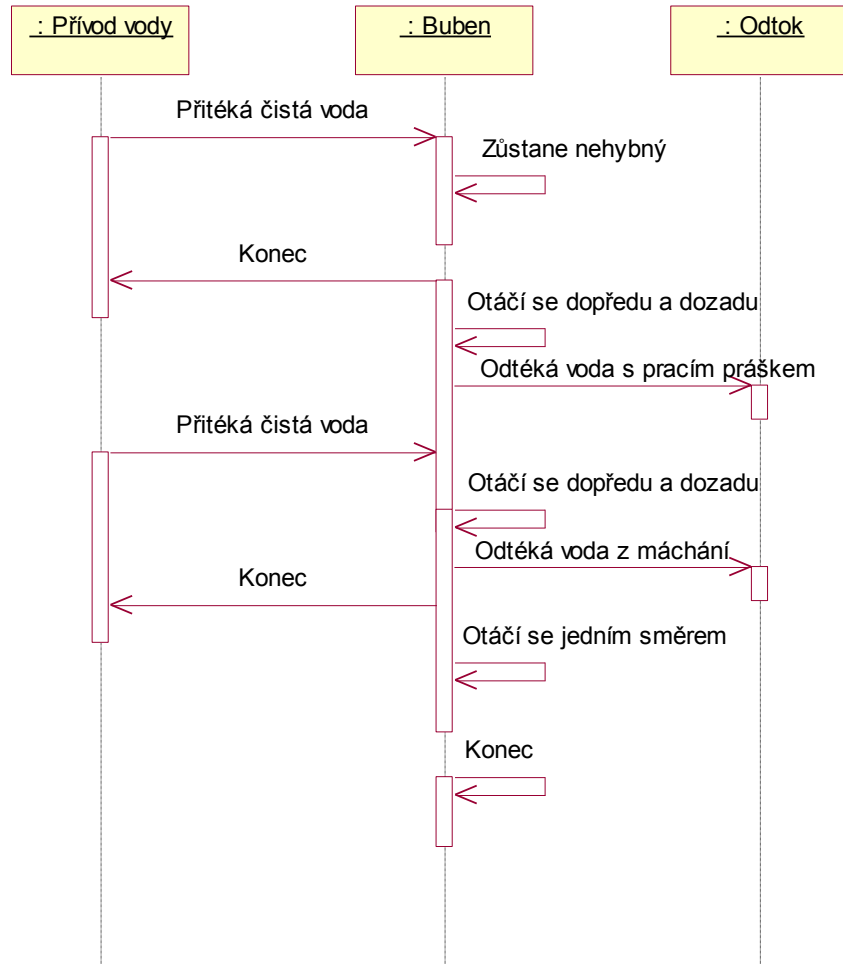
Vytvořte sekvenční diagram pro praní prádla v pračce.

Předpokládejte interakce mezi těmito objekty:

- přívod vody
- buben
- odtok

1. Voda se přívodní hadicí napustí do bubnu
 2. Buben se 5 min nebude pohybovat
 3. Voda se přestane napouštět
 4. Buben se bude asi 15 min otáčet oběma směry
 5. Voda s pracím práškem se vypustí
 6. Voda se znovu začne napouštět
 7. Buben se znovu otáčí oběma směry
 8. Voda se přestane napouštět
 9. Voda z máchání se vypustí
 10. Buben se začne otáčet pouze jedním směrem a rychlost otáčení se na 5 min zvýší
 11. Buben se přestane otáčet, čímž praní končí
-

Příklady - opakování



Q&A

Děkuji za pozornost