

---

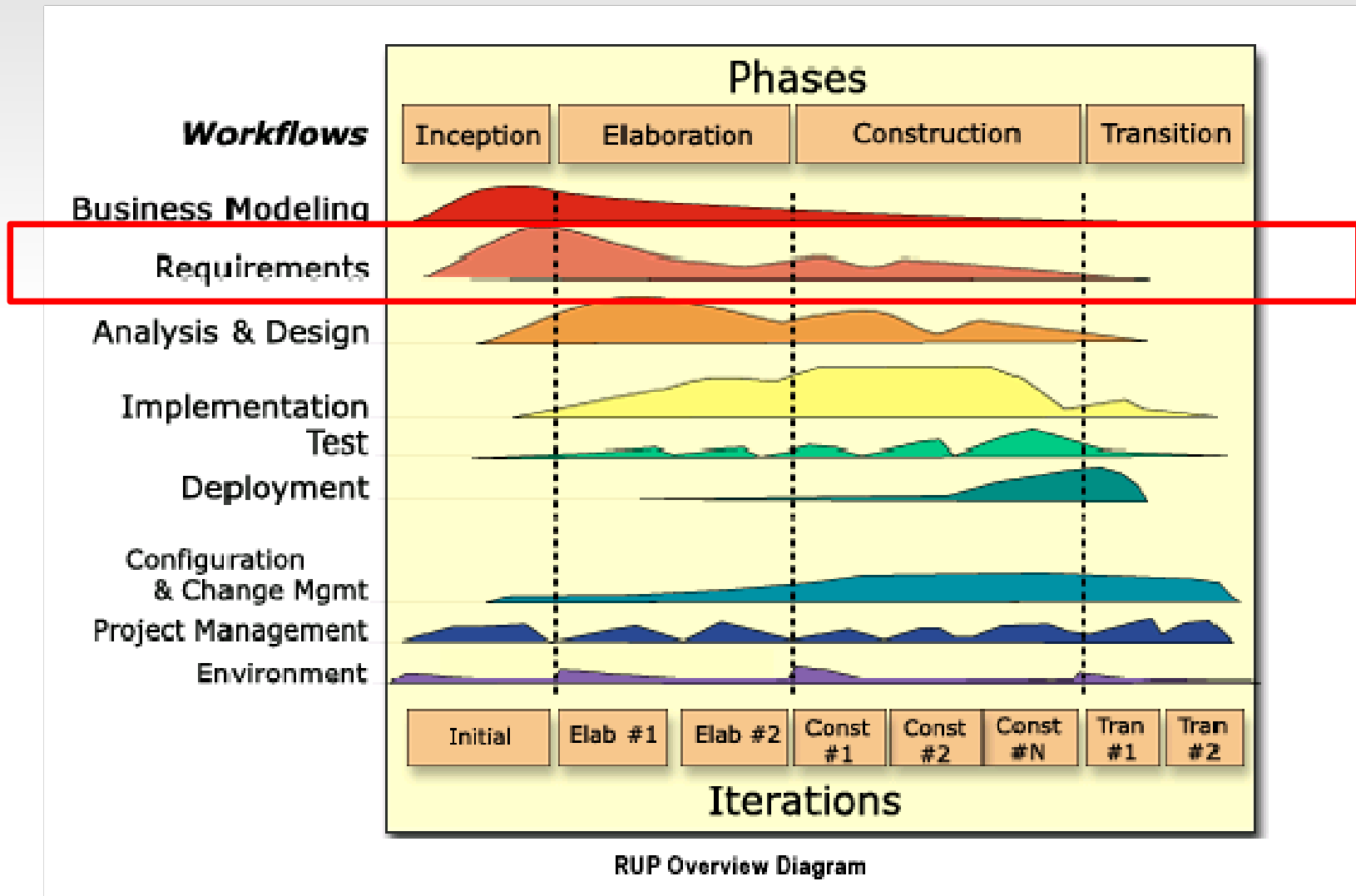
# **Požadavky na systém**

**requirements model, use case modeling**

**© Radek Ošlejšek**  
**Fakulta informatiky MU**  
**oslejsek@fi.muni.cz**

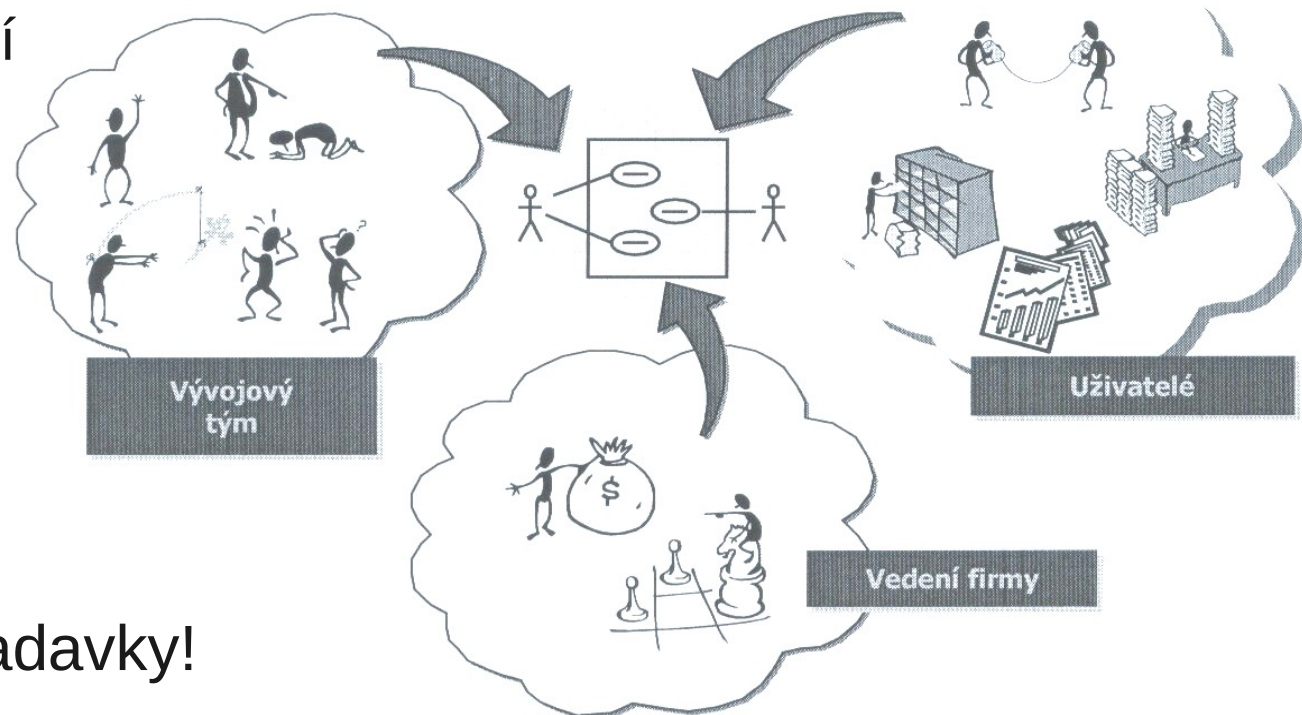
# Requirements workflow

- Převážně ve fázích zahájení a rozpracování, tj. na začátku projektu
- Business modeling – viz specializované předměty



# Problém zachycení požadavků

- Požadavky vycházejí z kontextu modelovaného systému:
  - uživatelé, kteří budou přímo používat systém
  - další zainteresované osoby (manažeři, správci, ...)
  - jiné systémy, se kterými bude náš systém interagovat
  - hardwarová zařízení, se kterými bude náš systém interagovat
  - zákonné a regulační omezení
  - technická omezení
  - obchodní cíle



- Často protichůdné požadavky!

# Funkční vs. nefunkční požadavky

- RUP a UML se zaměřují především na modelování funkčních požadavků (co bude systém dělat)
- Velmi důležité jsou ale i nefunkční požadavky (výkonnost, spolehlivost, ...)!
  - Proto se používá
    - *requirements model* – většinou neformální model zachycující i nefunkční požadavky
    - *use case model* – formální modelování funkčních požadavků pomocí UML
- Funkční i nefunkční požadavky by měly být formulovány ve smyslu **co** bude systém dělat, nikoliv **jak** to bude dělat
  - V praxi je problém toto dodržet, protože někdy je snadnější požadavek popsat pomocí konkrétní technologie, implementačních omezení apod.

---

# Requirements Model

# Metody získávání požadavků: Interview

---

- Interview

- Rozhovor nejlépe mezi 4 očima.
- Nechte si své nápady pro sebe, jen se ptejte na představy zpovídaného, jinak nezjistíte, co skutečně potřebuje.
- Kladte otázky, které nepredikují odpověď („Kdo používá systém“ vs. „Používá systém i Karel?“).
- Naslouchajte. Nechte zpovídaného povídat.
- Nepředvídejte. Nevěřte tomu, že víte, co si zpovídaný myslí, jak se cítí apod. Raději se ho zeptejte.
- Buďte trpěliví.

# Metody získávání požadavků: Dotazník

---

- Nemohou nahradit interview
  - je těžké předem zformulovat otázky pokud nevíte na co přesně se ptát
- Mohou být vhodným doplňkem k interview
  - klíčové otázky, které vyplynuly z interview, mohou být převedeny do formy dotazníku a předloženy širšímu okruhu dotazovaných

# Metody získávání požadavků: Workshop

---

- Setkání zainteresovaných osob
  - doménoví experti, requirements engineers, klíčoví uživatelé, ...
- Brainstorming
  - Vysvětlete, že se bude postupovat podle metody brainstormingu
    - jakákoliv myšlenka je dobrá myšlenka
    - každá myšlenka je zapsána bez diskuze
  - Vyzvěte zúčastněné, aby vyjmenovali svoje klíčové požadavky na systém
    - každý požadavek napište na papírek na nalepte na tabuli
  - Iterujte přes identifikované požadavky (papírky) s cílem doplnit je, upřesnit apod.
  - Převeďte výsledek do podoby soupisu požadavků



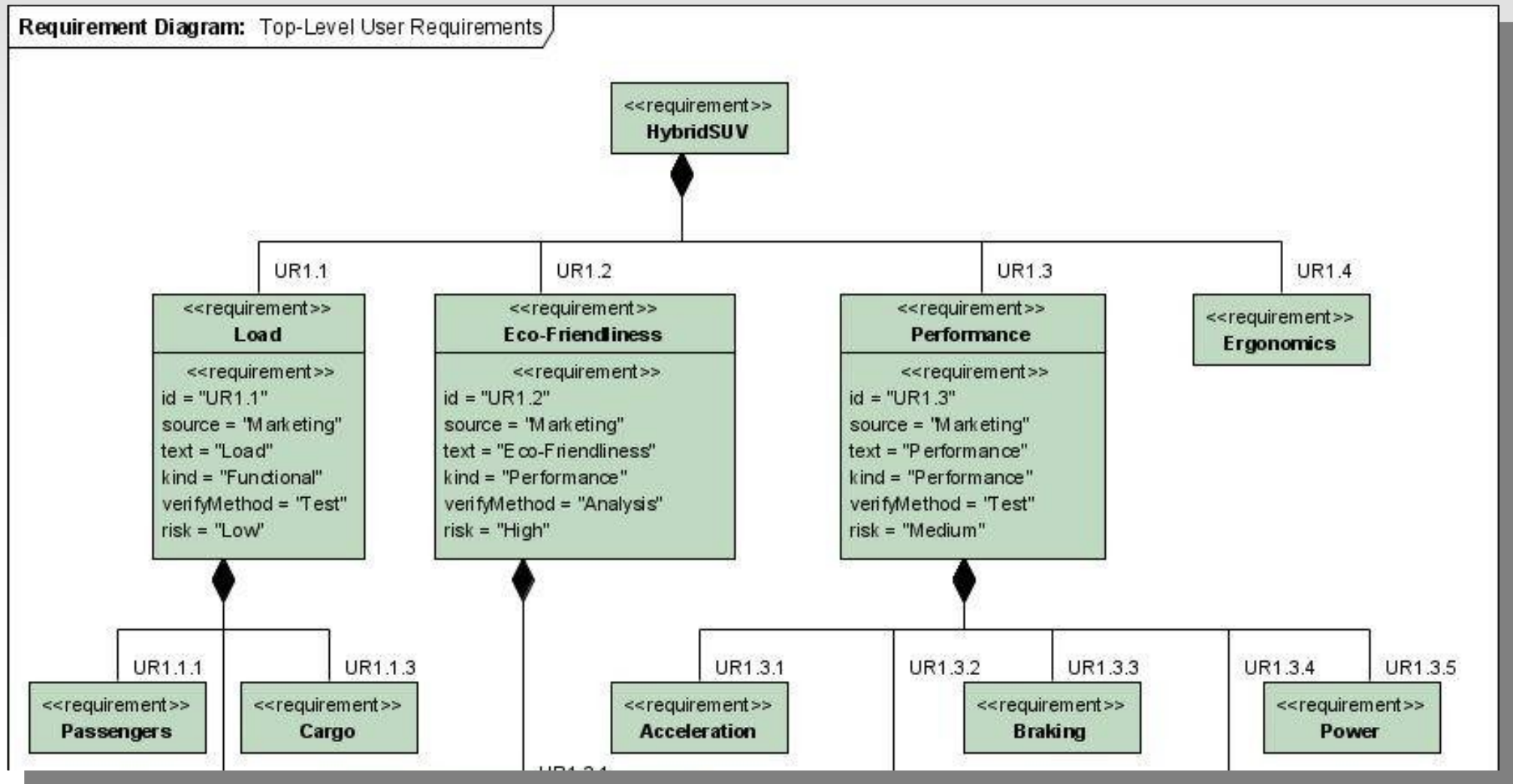
# Requirements model - příklad

---

- Např. věty „<id> <system> <klíčové-sloveso> <function>“
- Funkční požadavky na bankomat:
  1. Bankomat ověří platnost vložené kreditní karty.
  2. Bankomat ověří PIN zadaný zákazníkem.
  3. Bankomat nevydá více jak 20.000,- Kč během 24 hodin.
- Nefunkční požadavky na bankomat:
  1. Bankomat bude naprogramován v jazyce C++.
  2. Bankomat bude komunikovat s bankou pomocí 256bitového šifrování.
  3. Bankomat ověří platnost kreditní karty do tří sekund.
  4. Bankomat ověří PIN do tří sekund.
- Atributy požadavků: priorita, status, riziko, ...

# Requirements model – příklad 2

- Requirements diagram ze **SysML** (viz např. [sysml.org](http://sysml.org))

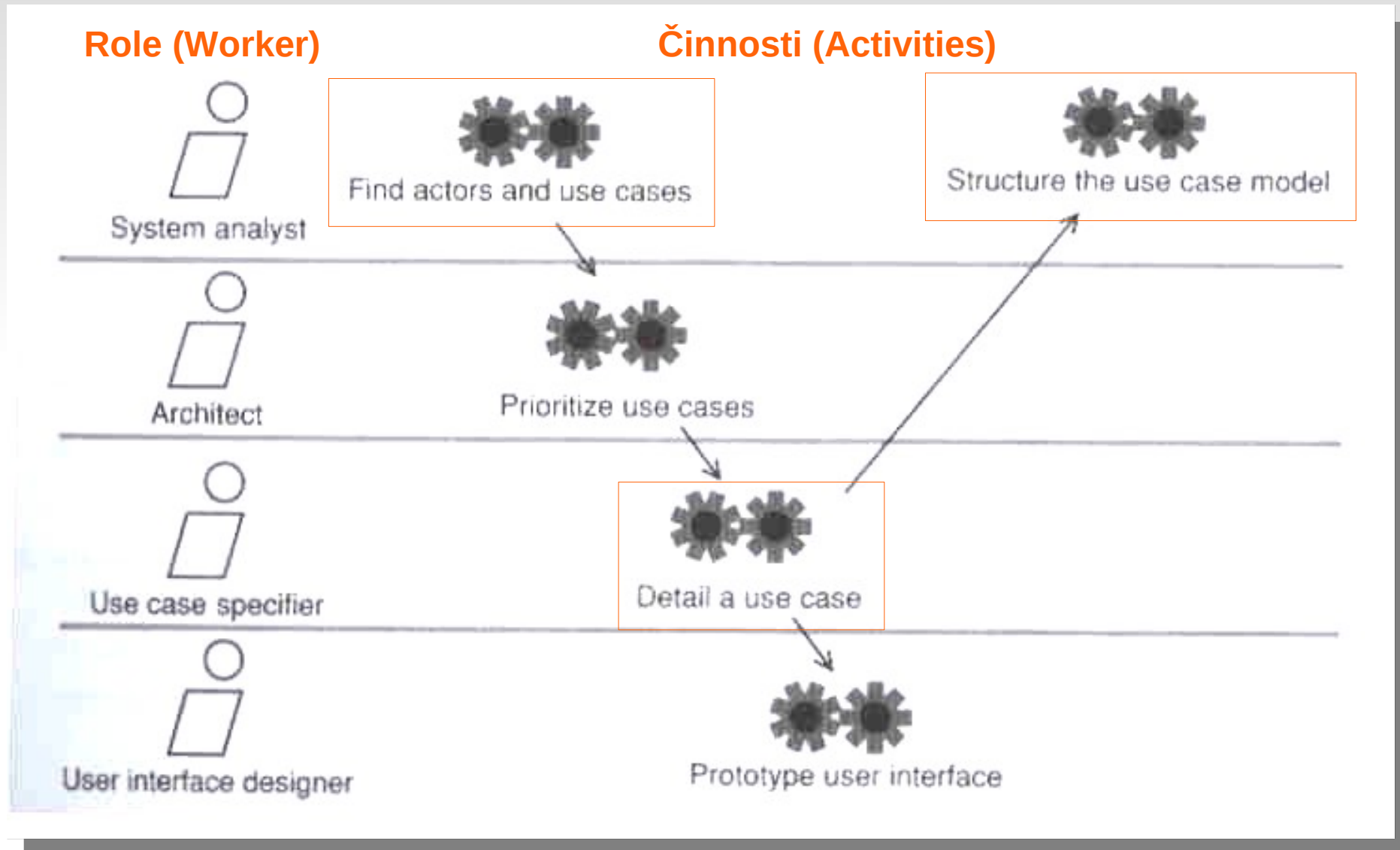


---

# Use case model

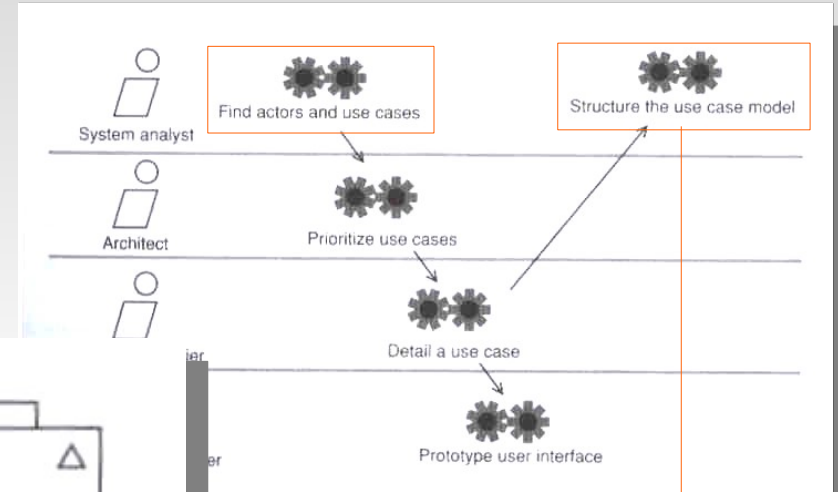
# UP: Requirements workflow detail

- Vyznačené aktivity budou podrobně probrány v následujícím výkladu.

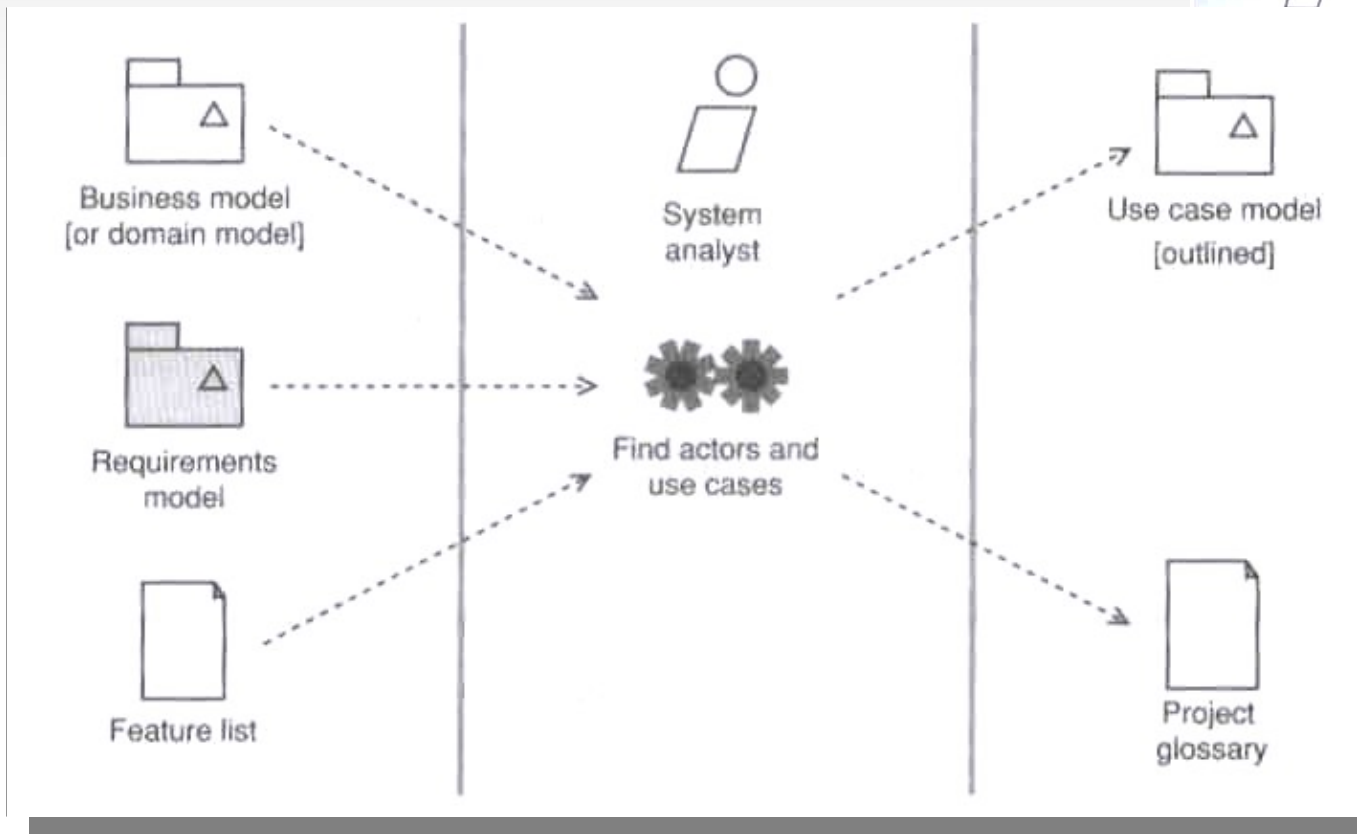


# UP aktivita: Nalezení aktérů a případů užití

- *Business model* – ideální zdroj pro hledání use casů
- *Requirements model* – vysvětleno
- *Feature list* – např. dokument vize
- *Project glossary* – slovník problémové oblasti



Restrukturalizace  
složitých modelů



# Dokument *Vize*

---

- Vytvářený systémovým analytikem během fáze zahájení (inception)
  - Obecný popis systému
  - Přínosy pro zadavatele
  - Důležité cíle z pohledu zadavatele
- 
- Př.: viz systém Pojišťovna ve studijních materiálech

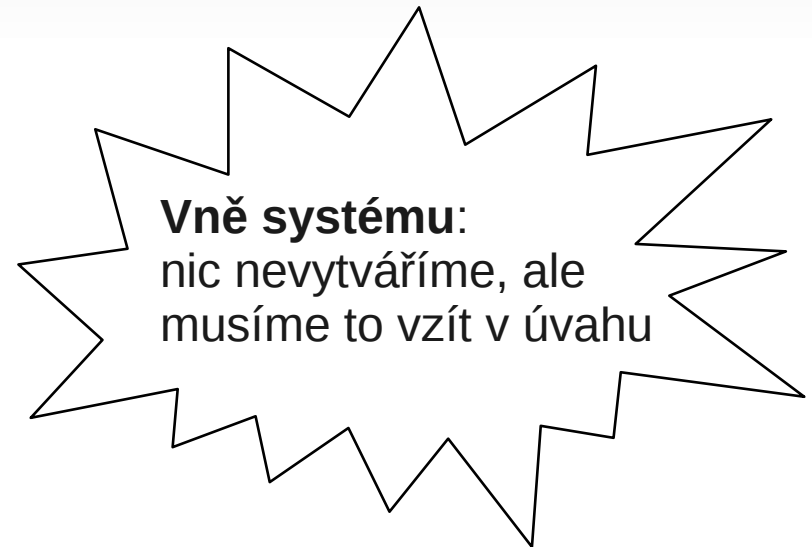
# Diagram případů užití

---

- Další názvy: *Use Case Diagram, UCD*
- Co potřebujeme:
  - Chceme zachytit potřeby a požadavky uživatelů
  - Grafické znázornění dostatečně jednoduché a čitelné i pro laiky
  - Potřebujeme efektivní mechanismus komunikace mezi uživateli a vývojáři
- Postup:
  - Iniciální vymezení hranice systému
  - Nalezení aktérů
  - Nalezení případů užití
    - specifikace případů užití
    - identifikace klíčových alternativních toků
  - Iterace přes případy užití, aktéry a hranici systému, dokud se nestanou stabilní

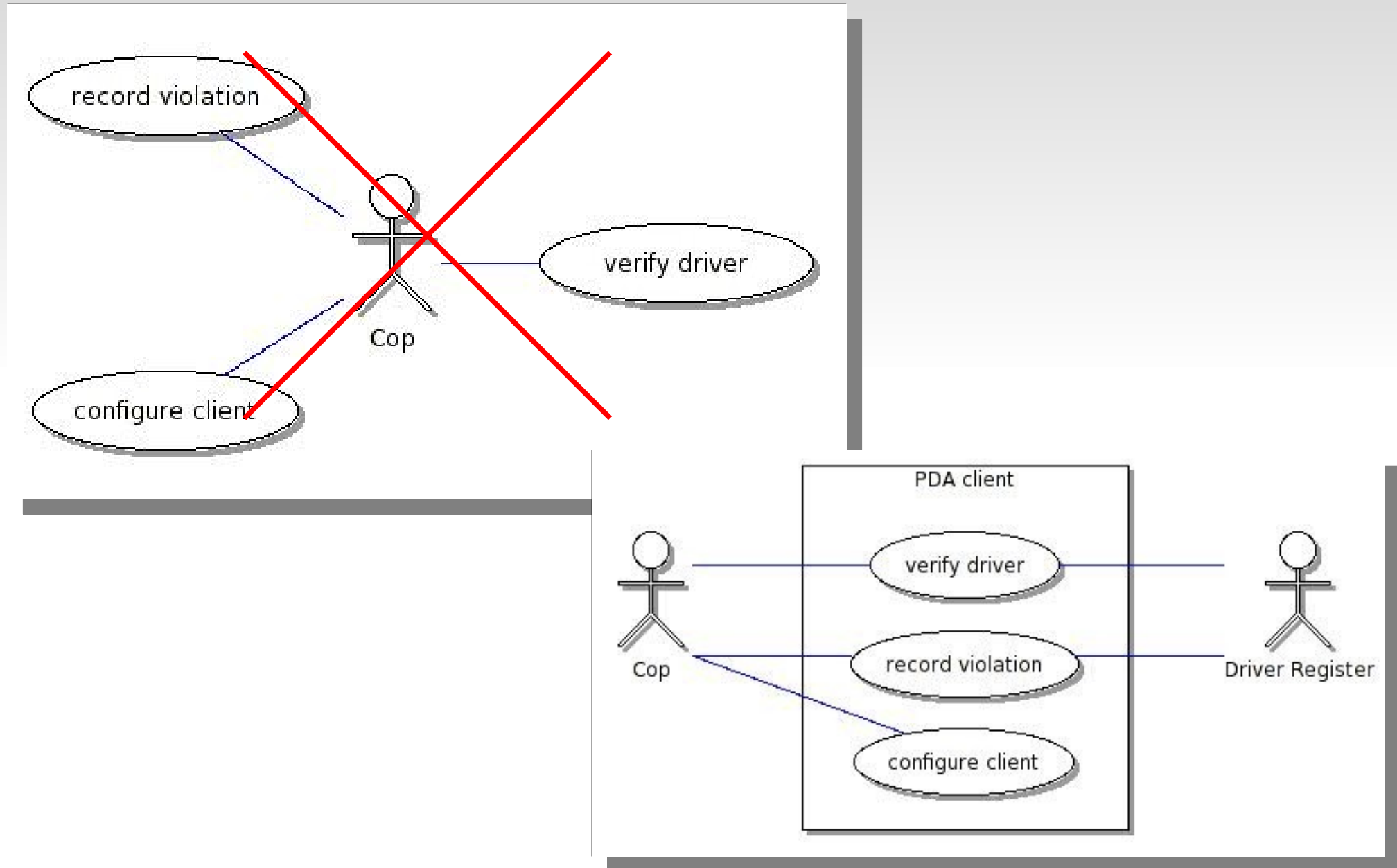
# UCD: Subjekt (hranice systému)

- Angl.: *Subject, System Boundary* (používáno před UML 2.0)
- Je možné vytvářet více UCD pro různé úrovně abstrakce a různé části systému a propojovat je
- Často si ale vystačíme s jediným UCD (důležitá je přehlednost a čitelnost)





# UCD: Subjekt (II)



# UCD: Aktéři

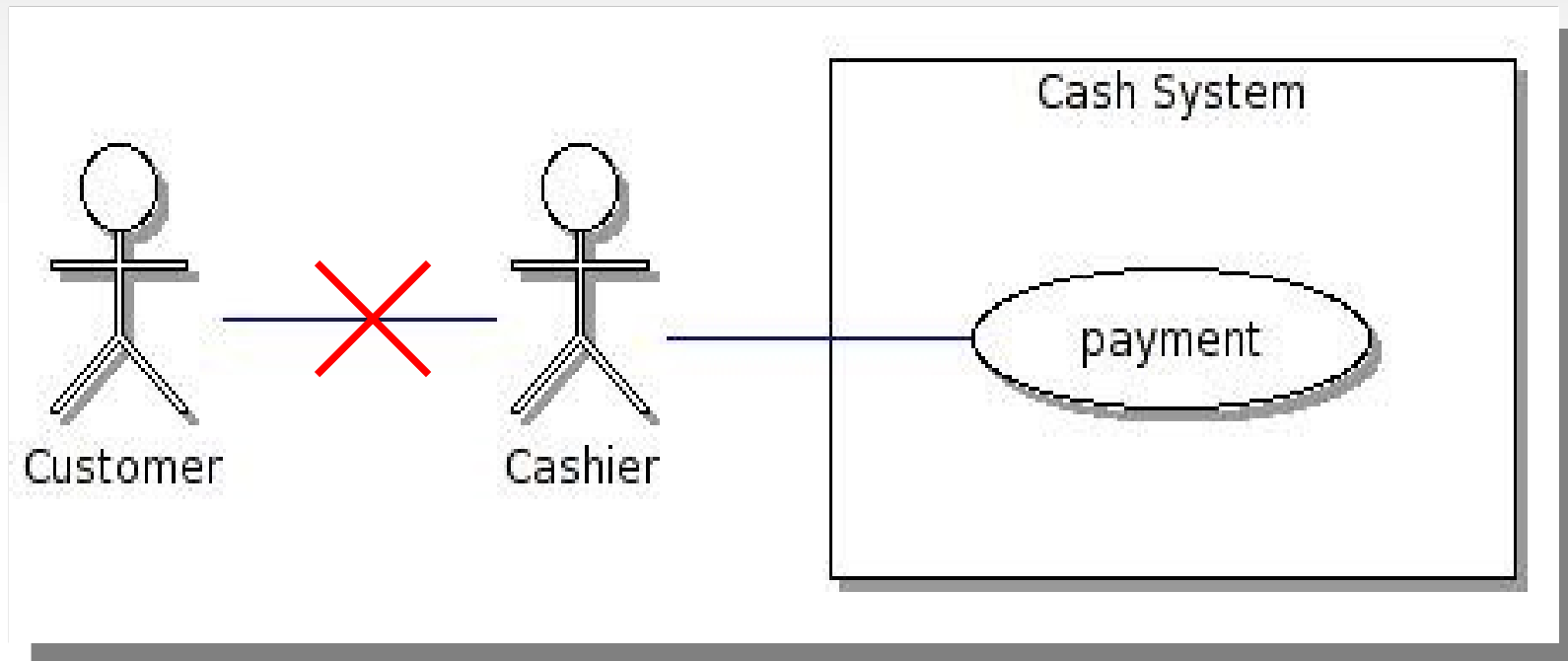
- Další názvy: *účastník*, *actor*
- Entity vně hranic systému, které **přímo** interagují se systémem.
- Dvě možnosti grafického znázornění („panáček“ je nejpoužívanější):



- Modelují **role**, ne konkrétní osoby!
  - v jedné roli může vystupovat více osob
    - např.: Jan Novák a Karel Zeman jsou oba *zákazníci*.
  - jedna osoba může hrát v systému více rolí
    - např.: Jan Novák je zároveň *vedoucí skladu a kontrolor*.
- Nejen uživatelé, ale i **externí systémy**, **části hardwaru** apod.
- Od UML 2 i **ostatní subjekty** => propojení UC modelů
- Aktérem může být i **čas**
  - např.: aktér s názvem „automatická záloha každý večer“, „at 6 am“ apod.

# UCD: Aktéři (II)

- Aktéři jsou externí, často se ale objeví i jako objekt uvnitř systému
  - např.: aktér *zákazník* komunikuje se systémem, objekt *zákazník* si uvnitř systému pamatuje detaily (jméno, adresa, ...)
- Modelujeme jen interakci mezi účastníky a systémem, nikoli interakce mezi účastníky



Otázka: Kterého aktéra ponechat?

Odpověď: Kdo je zodpovědný za správnost dat?

Kdo fyzicky interaguje (komunikuje) se systémem?

# UCD: Aktéři (III)

---

- Více aktérů pro jeden případ užití? Ano, ale:
  - Příklad užití je „přístupný“ více uživatelským rolím, pak je vhodné zamyslet se nad
    - vhodnějším rozdělením rolí
    - dědičností mezi aktéry
  - Na řešení případu užití se podílí více rolí (uživatelů) současně
    - ve skutečnosti se za tím často skrývá několik nezávislých navazujících interakcí (=> dekompozice při následném upřesňování hranic systému)
- Jednosměrná komunikace mezi aktérem a případem užití
  - Nejčastěji ze systému směrem ven
  - Vhodné pro „pasivní“ zařízení/systémy a zajištěné spolehlivé komunikační protokoly
  - Používá se velmi zřídka, většinou se jedná o obousměrnou interakci

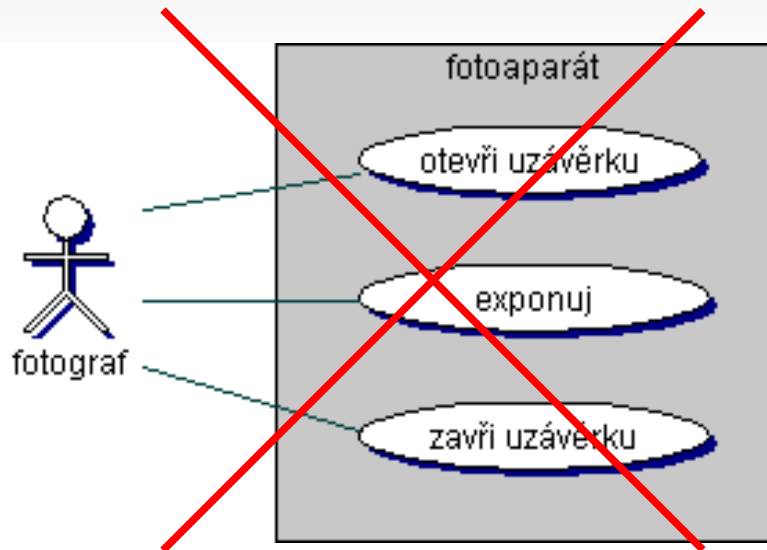
# Jak najít aktéry

---

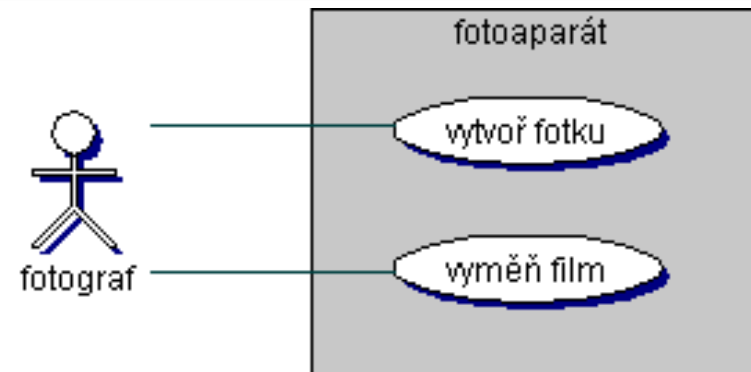
- Kdo nebo co používá systém?
- Jakou roli v této interakci hraje?
- Jaké další systémy interagují s tímto systémem?
- Kdo nebo co plní informace do systému?
- Kdo nebo co informace používá?
- Děje se něco pravidelně v předem daných časech?

# UCD: Případy užití

- Další názvy: *Use Case*, *typová úloha*, *uživatelská transakce*, ...
- Koherentní jednotky funkcionality poskytované systémem
  - Případ užití = konkrétní interakce, která vede ke splnění cíle
- Jsou vždy **zahájeny účastníkem**, vždy **napsány z pohledu účastníka!**
  - případy užití spuštěné časovými událostmi => účastník čas
- Jméno by mělo **vyjadřovat cíl**, kterého chce aktér dosáhnout



**špatně** – fotograf tyto činnosti nedělá izolovaně, ale jsou mu poskytnuty systémem jako jedna funkcionality “vytvoř fotku”



**správně** – fotograf vyvolává tyto činnosti a systém na ně reaguje

# Jak najít případy užití

---

- Záleží na použité metodice, obecně můžeme použít tyto strategie:
  - Každému elementárnímu procesu z business modelu (česky procesního modelování – viz přednáška *PV165: Procesní řízení*) odpovídá jeden případ užití.
  - Procházíme seznam účastníků a zvažujeme způsob, jakým bude každý z nich systém používat.
    - Jaké funkce jednotliví účastníci od systému očekávají?
    - Bude systém uchovávat a poskytovat informace? Pokud ano, kdo a jak bude tyto činnosti vykonávat/aktivovat?
    - Jací účastníci budou upozorněni na změnu stavu systému?
    - Existují nějaké vnější události, které ovlivňují systém? Co upozorní systém na tyto události?

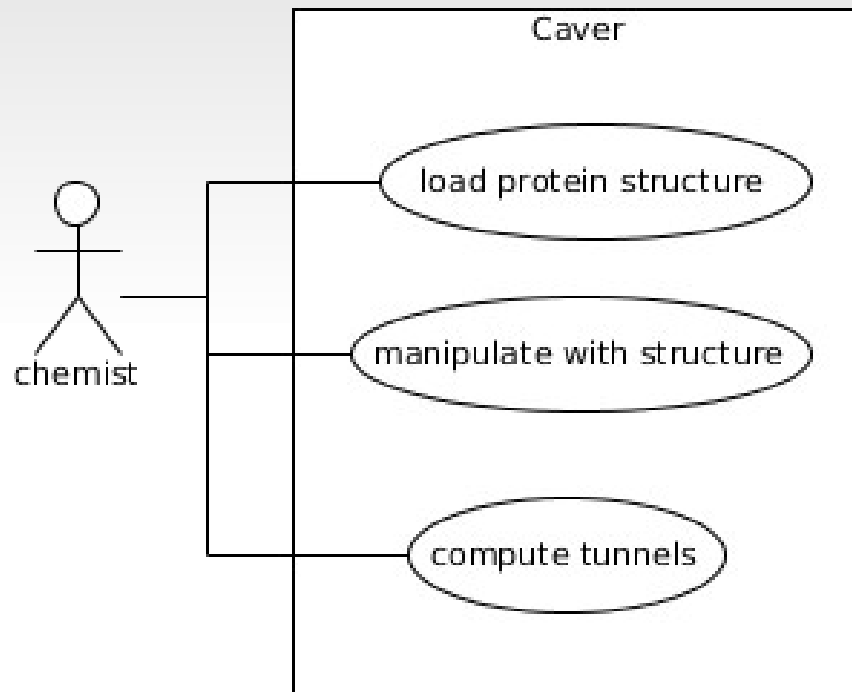
# Problémové případy užití

---

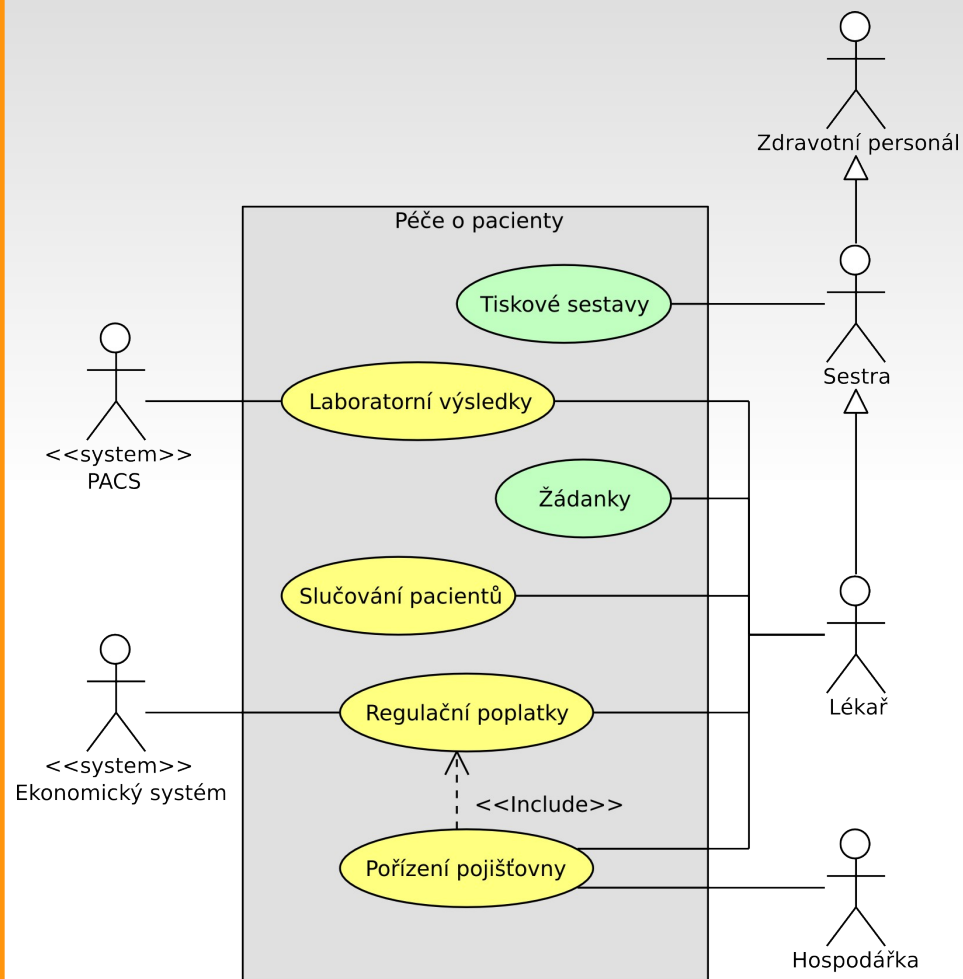
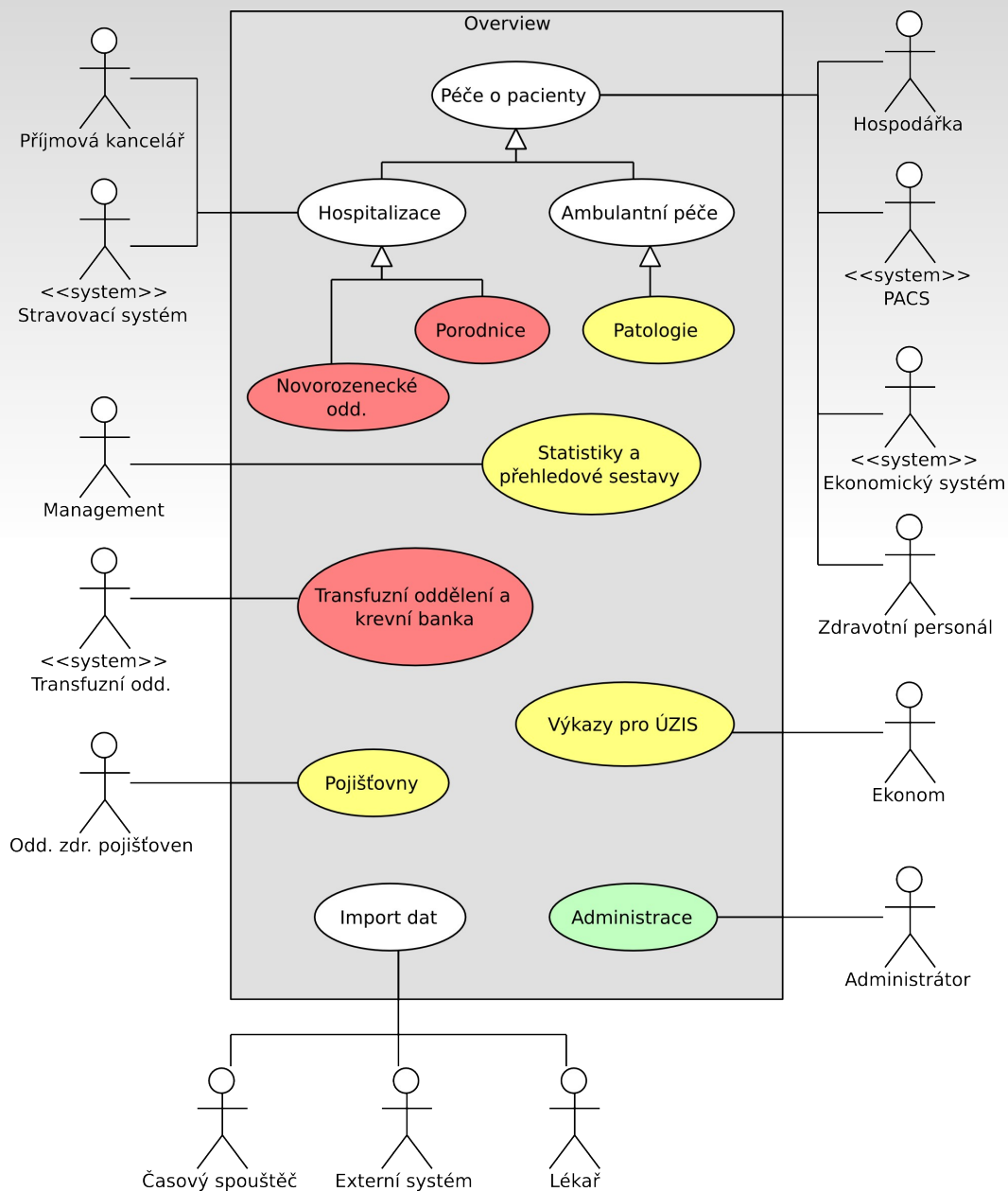
- Cross-cutting concerns = aspekty systému, které lze těžko zapouzdřit/dekomponovat, protože jsou rozprostřeny přes celý systém.
  - Např. autentizace a autorizace, logování, transakce, synchronizace a paralelizmus, ...
- Těžko se izolují do separátních případů užití, balíků, tříd apod., zejména v počátečních obecných modelech.
- Řešení:
  - Z počátku nemodelovat, nechat na později (např. přihlašování modelovat až v okamžiku, kdy řešíme autentizační mechanismus)
  - Použít tzv. aspect-oriented metodiky a frameworky
    - Spring MVC vs. Spring AOP
    - Vhodné zejména pro systémy s převahou cross-cutting concerns



# Příklad z praxe: Jednoduchý UC model



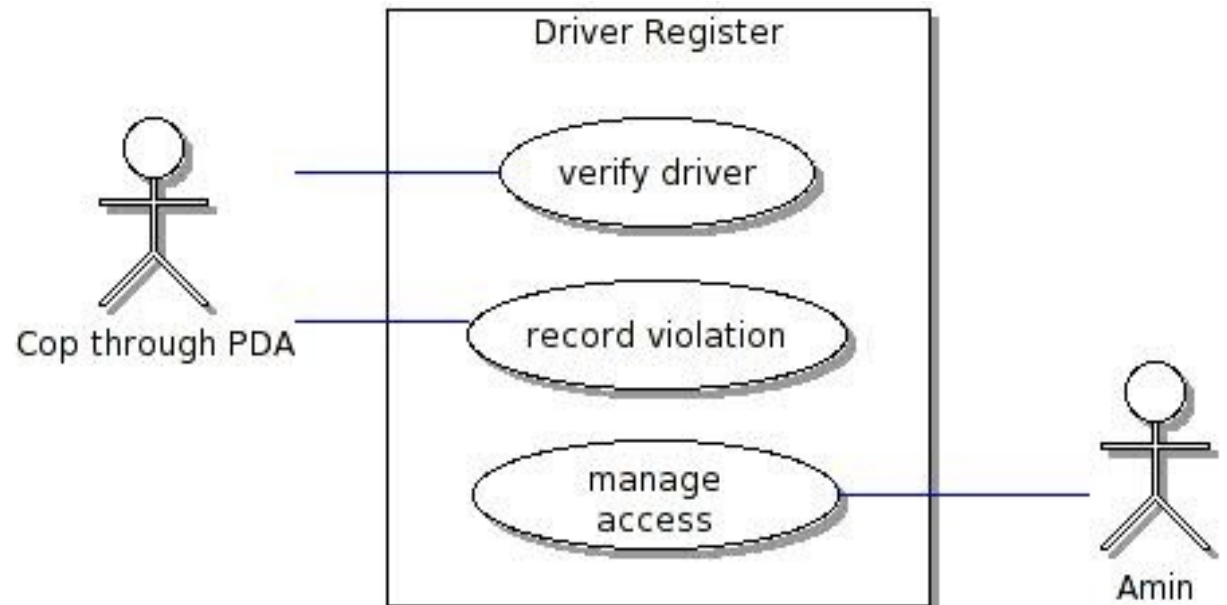
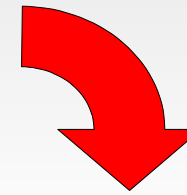
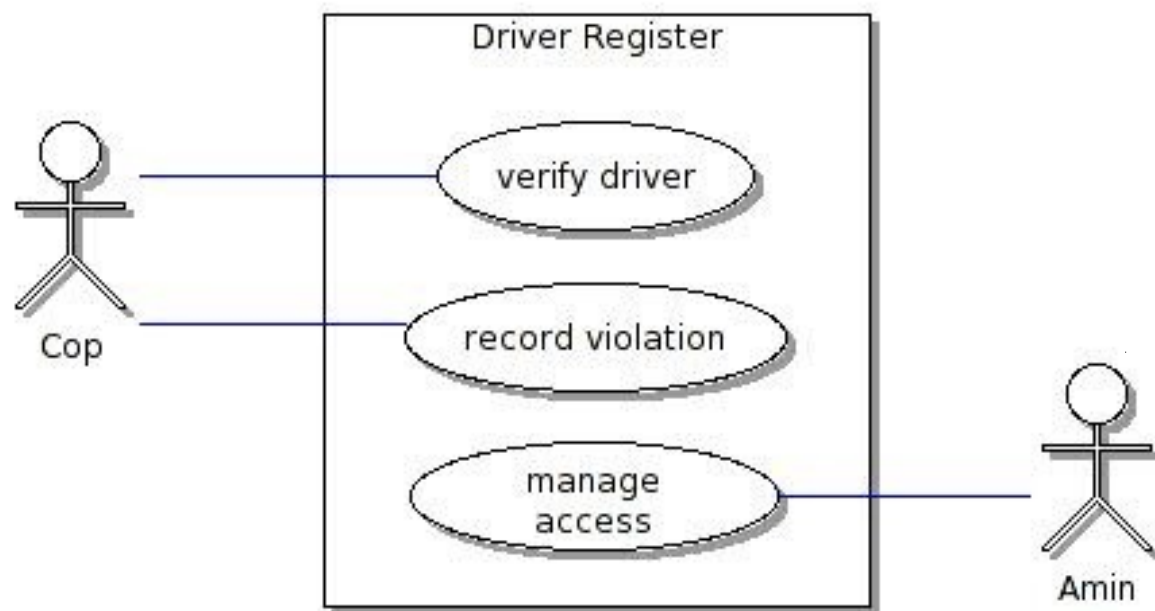
# Příklad z praxe: Složitý UC model



# Vymezení hranic systému

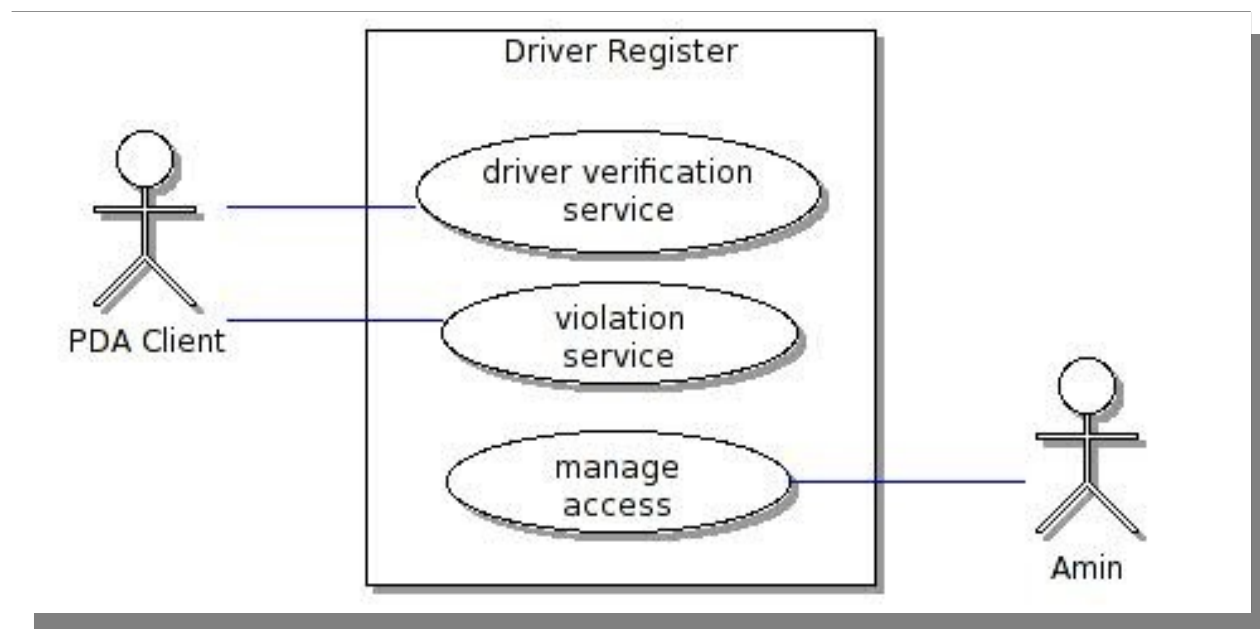
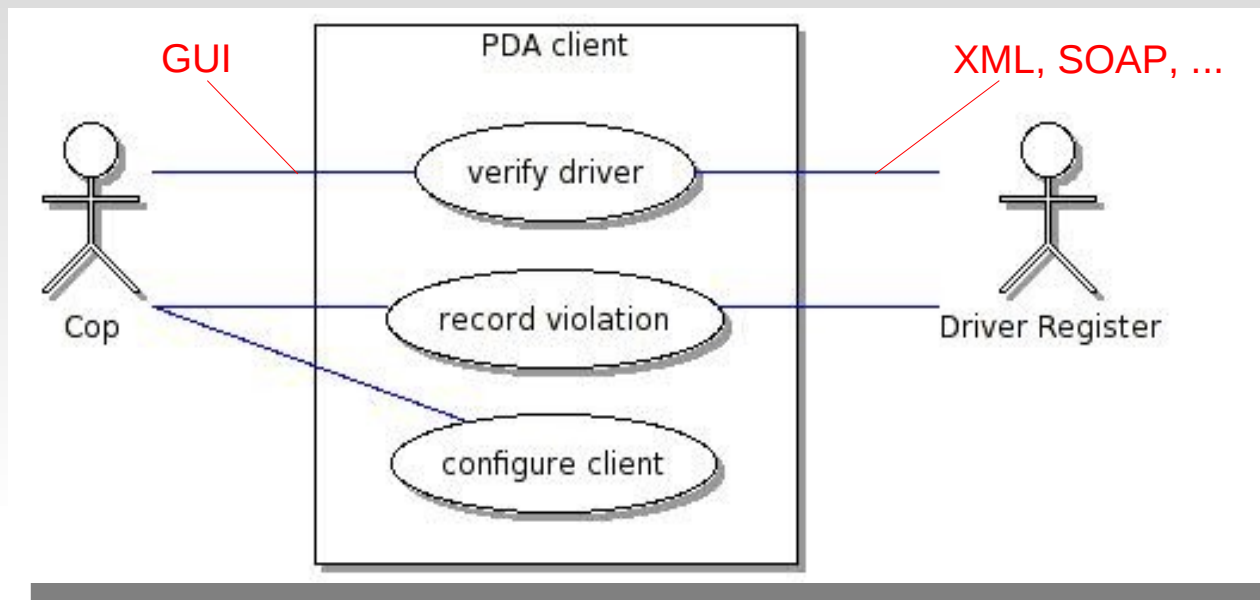
- Definice hranic systému probíhá iterativně.
- Hranice systému mohou být definovány na několika úrovních abstrakce, účastníci interakcí se mění podle úrovně abstrakce!
  - vysoká úroveň bez žádných reprezentantů
  - střední úroveň, která bere v úvahu kdo/co skutečně vkládá za informaci
  - nižší úroveň, která bere v úvahu, jaká data opravdu vstupují
- **Př. 1: platby za provedené služby**
  - Na začátku víme pouze to, že náš systém musí řešit platby za provedené služby.
  - Později si vyjasníme, že platby jsou evidovány v externím systému, který zadavatel používá. Náš systém tedy bude platby řešit přes tento externí systém.
  - Nakonec dospějeme k tomu, že komunikace bude probíhat prostřednictvím XML zpráv. Náš systém bude mít modul, který se o komunikaci bude starat.
- **Př. 2: zálohování dat**
  - Zákazník požaduje zálohování všech dat.
  - Později zjistíme, že data jsou uložena na externím databázovém serveru, který je automaticky zálohovaný a proto se o tuto činnost náš systém vůbec nemusí starat.

# Př: Vymezení hranic systému (II)



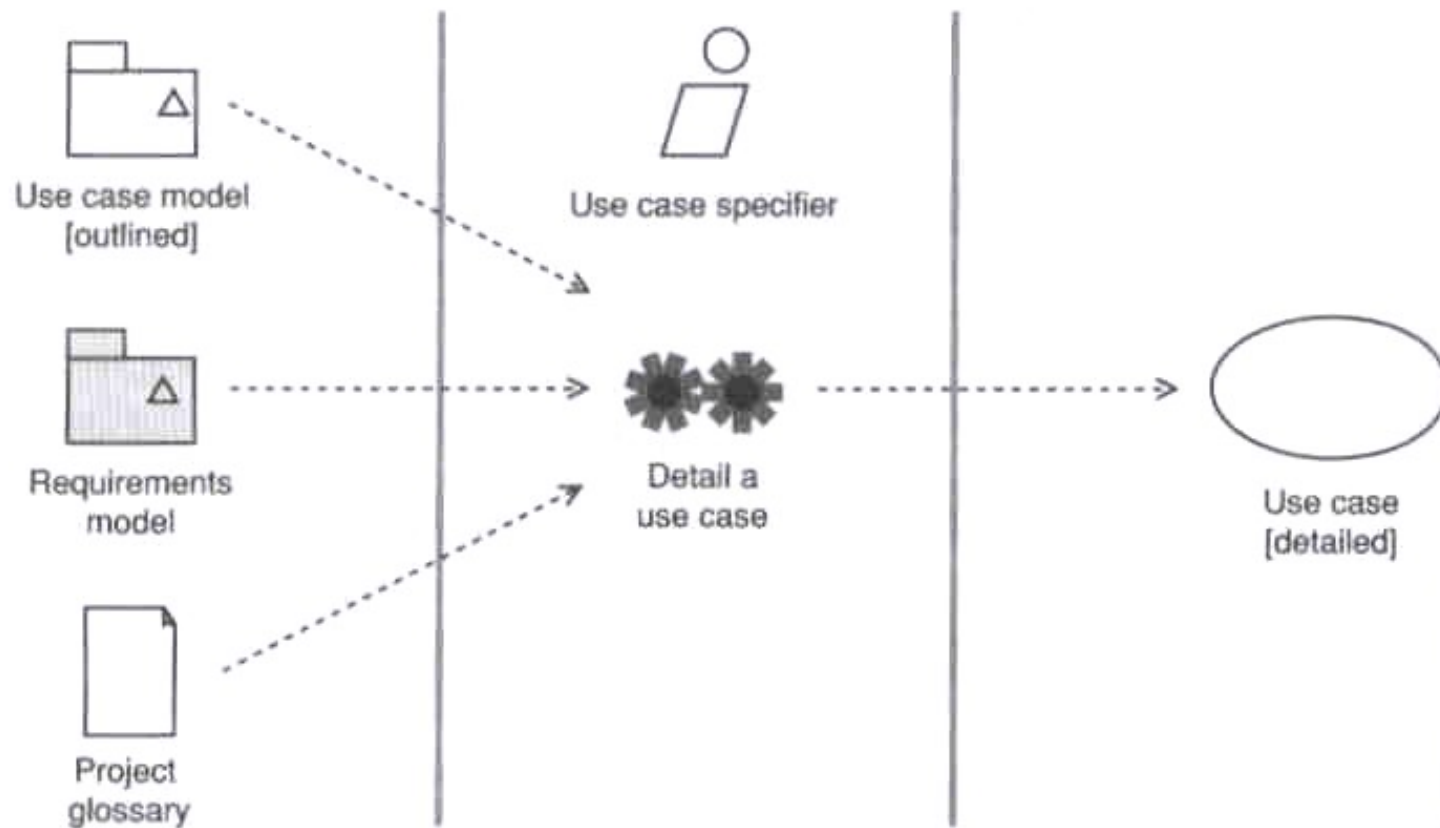
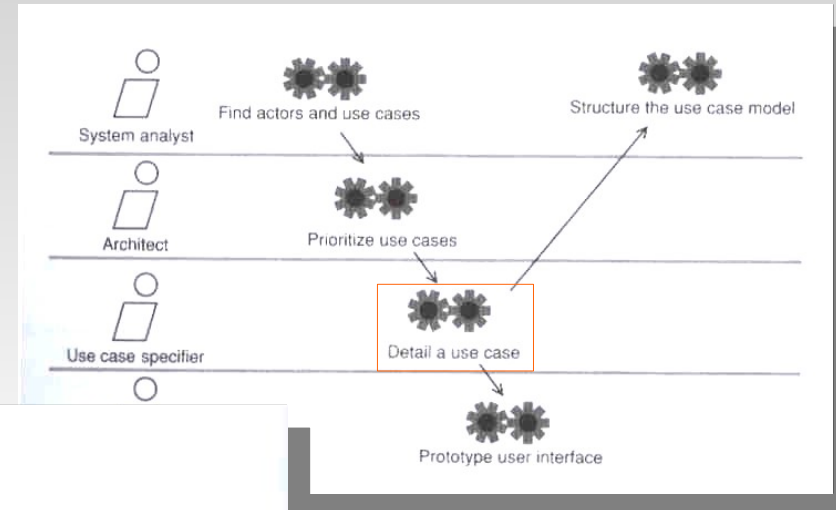
# Př: Vymezení hranic systému (III)

Na PDA běží speciální klientská aplikace, kterou musíme také vyvinout



# UP aktivita: Specifikace případů užití

- *Project glossary* – slovník modelované problémové oblasti (klíčová slova, synonyma, homonyma, ...)



# Specifikace případů užití

---

- Každý případ užití musí být dokumentovaný
- Dokumentace je psána z pohledu aktéra
- Obsahuje detaily, které musí systém poskytnout aktérovi při provádění případu užití
- Neexistuje žádný standard UML pro psaní specifikace p.u.
- ..., ale typicky obsahuje:
  - **název** případu užití
  - **aktéři** účastníci se případu užití
  - speciální (nefunkční) požadavky, např. časová omezení
  - priority a stav vývoje ...
  - **popis interakce** pomocí **toků událostí (scénářů)**

# Popis interakce pomocí toků událostí

Dokumentace pomocí toků událostí */flow of events/* nejčastěji obsahuje:

- **vstupní podmínky** */preconditions/*
  - kritéria, která musí být splněna před spuštěním p.u.
  - aktér nemůže p.u. spustit, dokud nejsou podmínky splněny
- **výstupní podmínky** */postconditions/*
  - kritéria, která musí být splněna na konci p.u. (popisují stav systému po ukončení p.u.)
- **normální tok** událostí */main flow*, jinak také *primární scénář – primary scenario/*
  - jednotlivé kroky interakce v případě užití
  - interakce za ideálních podmínek
  - je jediný v p.u. a většinou ho začíná primární aktér
  - může obsahovat jednoduché větvení
- **alternativní a výjimečné toky** událostí */alternative flows, secondary scenarios/*
  - náhrada za složité větvení
  - ošetření chyb, málo pravděpodobných situací apod.



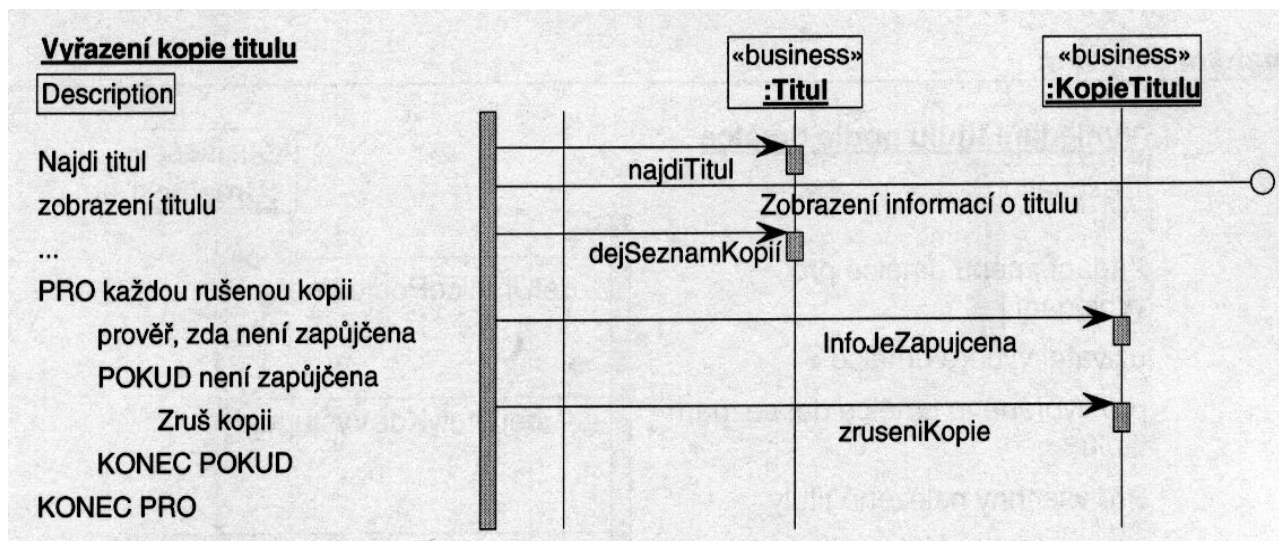
# Jak popsat toky událostí (I)

- Volný text
  - „slohové cvičení o tom, jak agent pojišťovny vyplňuje informace o pojistné události“
  - až příliš mnoho volnosti, zapomenou se důležité věci a naopak je tam mnoho zbytečného
- Číslované kroky
  - dobrý kompromis, nejpoužívanější
- Pseudokód
  - obvykle až příliš podrobný (není naším cílem popsat program textově)
- Diagram aktivit
  - grafická alternativa k číslovaným krokům
- Diagramy interakcí s pseudokódem nebo textem na levé straně
  - vhodné pro etapu rozpracování případů užití */use case realization/*, viz pozdější přednáška

# Jak popsat toky událostí (II)

1. p.u. začíná, když zákazník chce zobrazit obsah košíku
2. KDYŽ je košík prázdný
  - 2.1 systém zobrazí uživateli, že košík neobsahuje žádné položky
  - 2.2 p.u. končí
3. systém zobrazí seznam všech položek v košíku
- ...

*číslované kroky – první krok popisuje okolnosti zahájení činnosti aktérem*



*diagram interakcí s pseudokódem*

# Toky událostí – jak podrobně?

- Závisí na fázi projektu (např. nejprve číslované kroky, později diagram interakcí)
- Záleží na významu a složitosti p.u. (kompromis: **hlavní tok událostí** podrobně, **alternativní toky** stručně nebo dokonce jen vyjmenovat)
- Ne moc podrobné: zachycení požadavků neznamena popis konstrukce systému ve volném textu!

1. jsou zadány údaje o rušených položkách objednávky

Špatně:

- kdo údaje zadává?
- kam je zadává?
- co to jsou “údaje o rušených položkách”

1. => p.u. začíná, když uživatel zadá do formuláře číslo objednávky
2. <= systém vypíše seznam položek objednávky
3. => uživatel vybere rušené položky...

Správně, doporučený vzor kroků: <číslo> [i/o] <kdo> <akce>

# Alternativní toky událostí

- Zjednodušují a doplňují hlavní tok.
- Náhrada za větvení, zejména pokud není jasné, kam větvení umístit.
- Musí začínat “booleovskou” podmínkou, která daný alternativní tok spouští!
- Měl by být co nejjednodušší.
- Každý alternativní tok musí mít svoje vlastní výstupní podmínky!

**Případ užití:** zobrazení nákupního košíku

**Vstup. podmínky:** zákazník je přihlášen do systému

**Tok událostí:**

1. p.u. začíná když zákazník chce zobrazit obsah košíku
2. KDYŽ je košík prázdný
  - 2.1 systém zobrazí uživateli, že košík neobsahuje žádné položky
  - 2.2 p.u. končí
3. systém zobrazí seznam všech položek v košíku

**Výstup. podmínky:**

**Alternativní tok 1:**

1. zákazník může *kdykoli* opustit obrazovku košíku

**Výstup. podmínky:**

**Alternativní tok 2:**

1. zákazník může *kdykoli* opustit systém

**Výstup. podmínky:**

# Dokumentace pomocí diagramů aktivit

Diagramy aktivit patří do skupiny dynamických diagramů UML; mohou být také použity pro dokumentaci případů užití. Podrobněji budou probrány později.

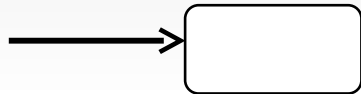


**začátek** případu užití

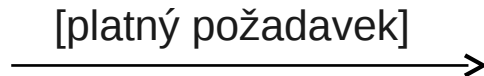
**konec** případu užití



**krok** případu užití (reprezentovaný jako stav obsahující akce), nebo **několik kroků** podrobněji popsanych na zvláštním diagramu

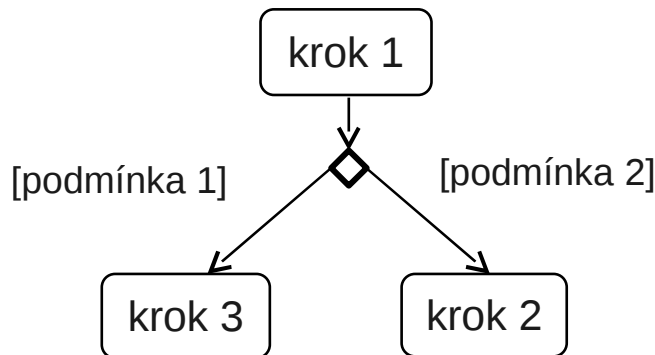


**přechod** mezi kroky

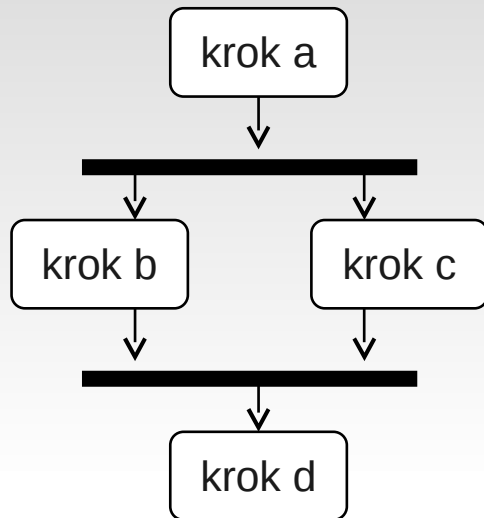


**podmínka** přechodu

rozhodnutí s **rozhodovacím bodem**



# Dokumentace pomocí diagramů aktivit pokr.



## paralelní kroky (fork & join)

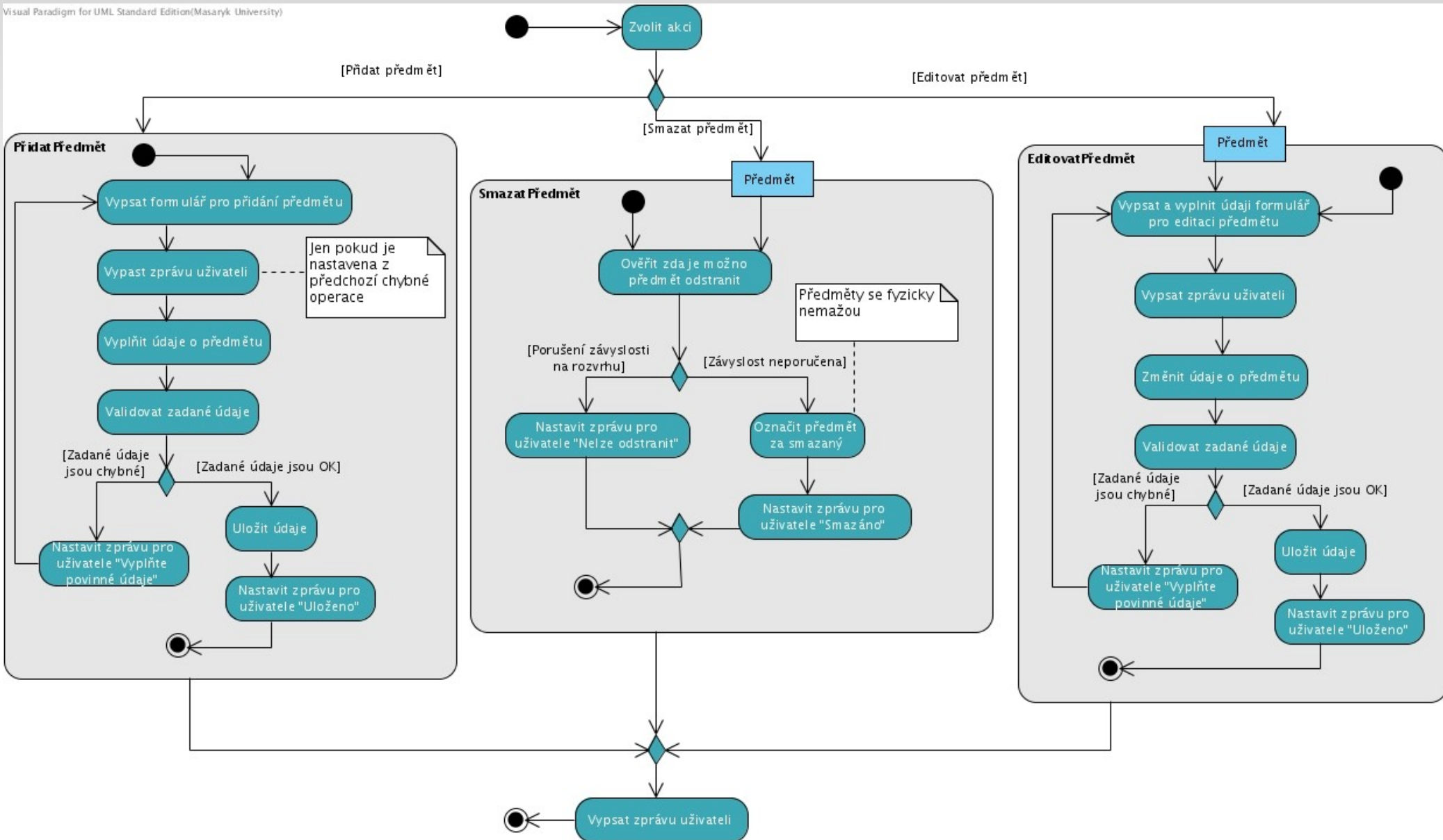
Diagramy aktivit obsahují více modelovacích prvků, které zde neuvádíme, protože nejsou relevantní pro popis případů užití

## Nedostatky:

- » grafická reprezentace zabírá velký prostor, textový popis je kompaktnější
- » nejsou znázorněny vztahy „užívá“ a „rozšiřuje“ (pokud si je sami nevytvoříte a nepřidáte do notace)
- » diagramy mohou být velmi složité ==> pro každý p.u. je nutné rozhodnout, zda je vhodné použít textový popis, diagram aktivit nebo obojí.

# Diagram aktivit – příklad

Visual Paradigm for UML Standard Edition(Masaryk University)



---

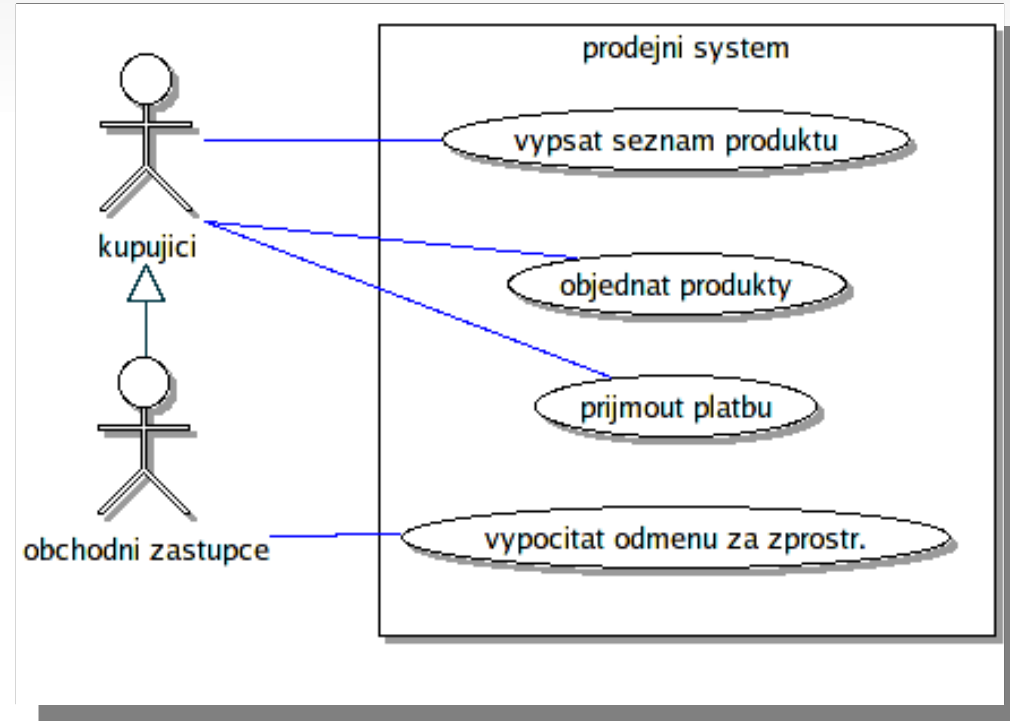
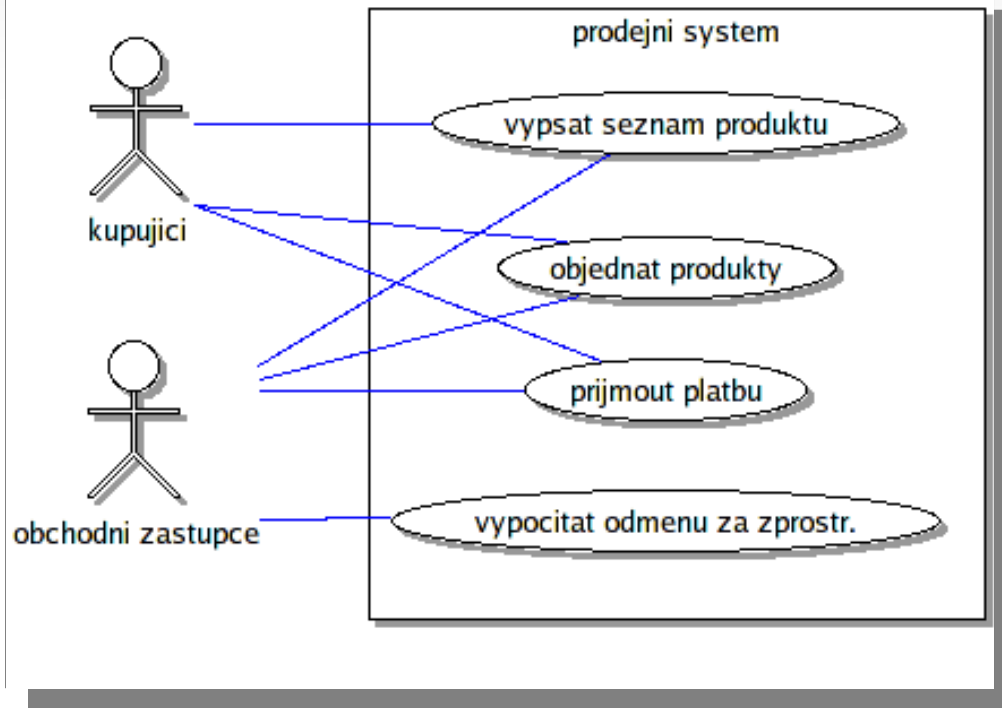
# **Pokročilé modelování případů užití**

## **aneb co používat s rozmyslem**

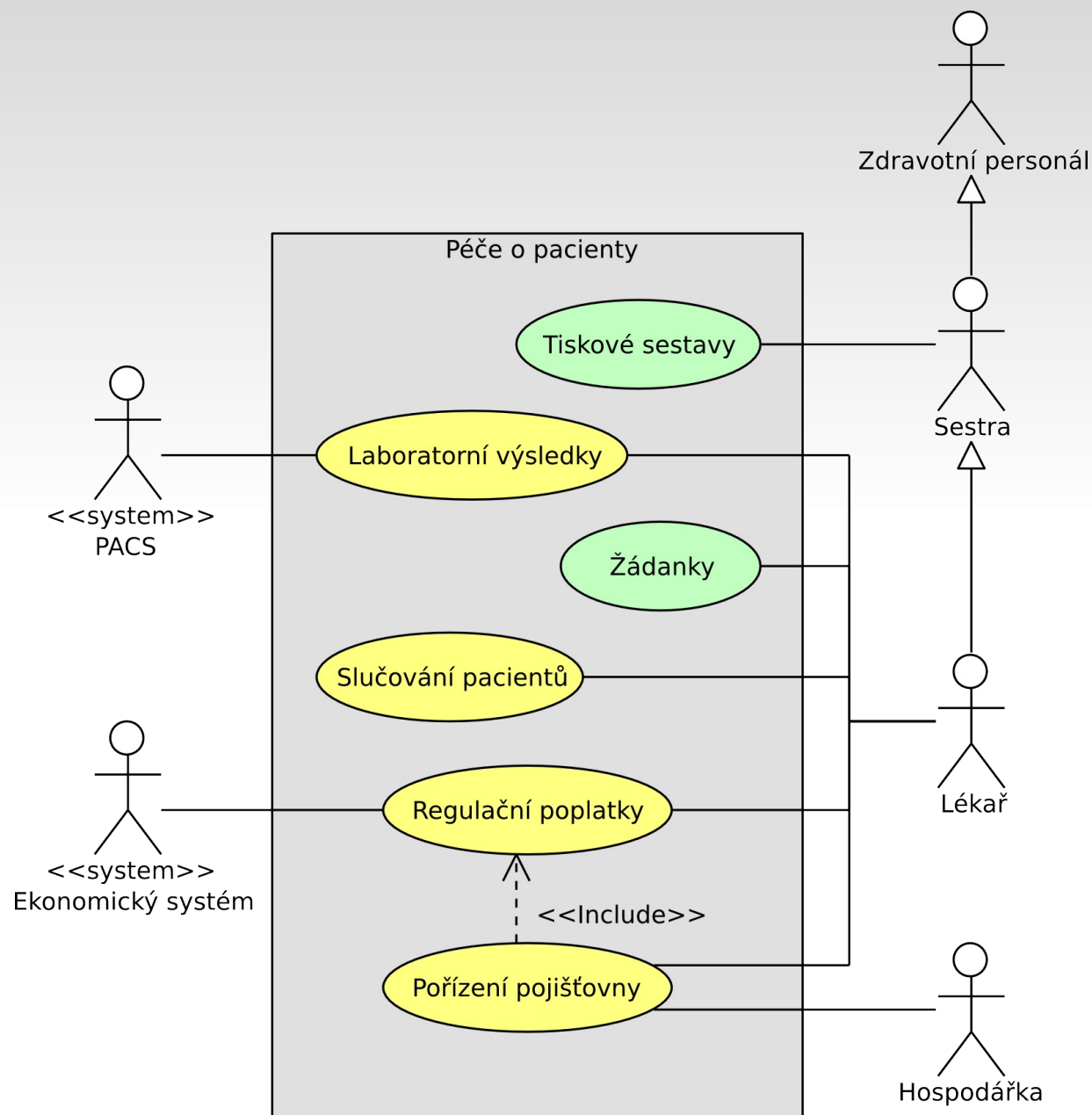


# Dědičnost aktérů

- Aktér *obchodní zástupce* je podtypem aktéra *kupující*
- Specializovaný aktér plní stejnou roli jako obecnější aktér, a pravděpodobně další roli navíc
- Specializovaný aktér se účastní ve všech případech užití, ve který se účastní obecnější (dědí spouštění případů užití „nadaktéra“)

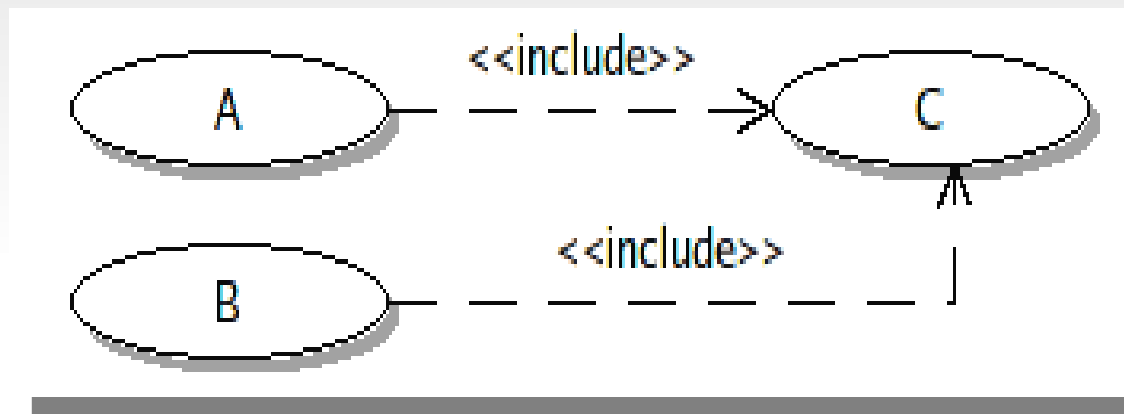


# Příklad z praxe: Dědičnost aktérů



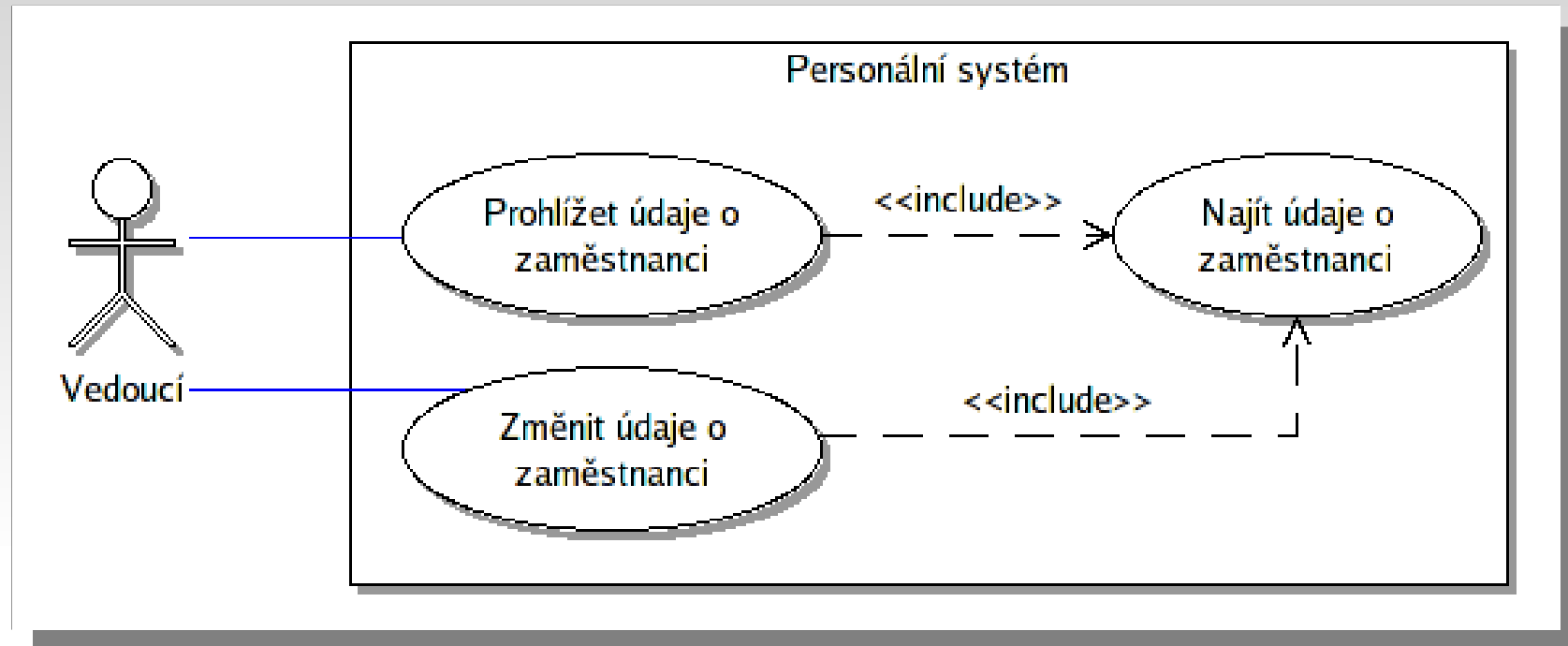
# Vztah <<include>>

- Další názvy: *užívá*
- Vztah mezi dvěma případy užití
- Společné chování dvou nebo více p.u. může být vyčleněno do zvláštního případu užití (vyhýbá se opakování a kopírování společných kroků)



- A a B nejsou kompletní bez C
- C je odkazováno alespoň jednou v A i B
- C může být (a často je) použito ve více případech užití (jinak <<include>> postrádá smysl)
- Pokud je C úplný p.u., může být vyvolán i přímo aktérem
- C je popsáno stejně, jako jiné případy užití
- **I C představuje interakci, nikoliv modul/knihovnu/... !!!**

# Vztah <<include>> (II)



**Případ užití:** Změnit údaje o zaměstnanci

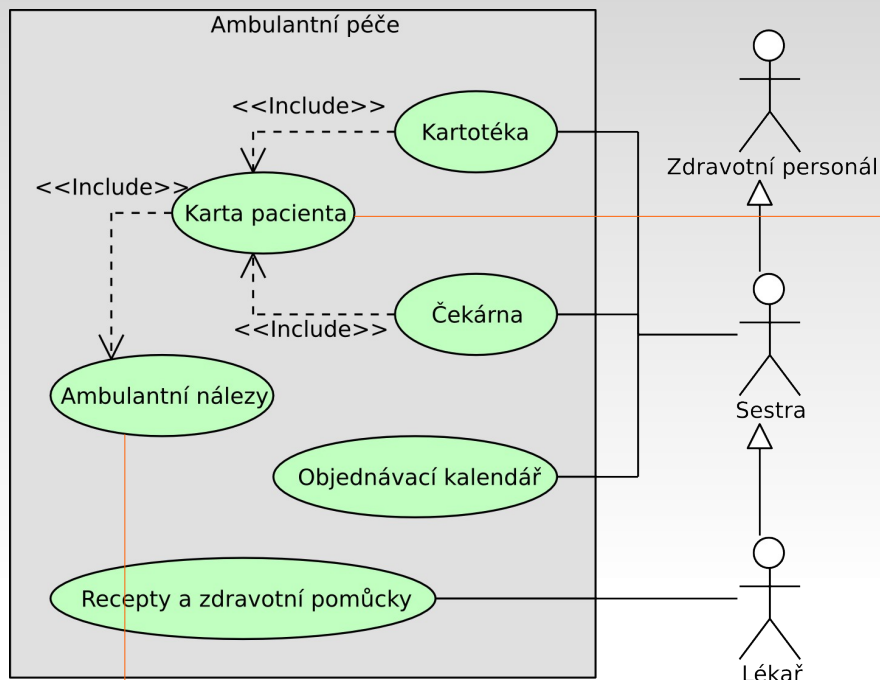
**Vstup. podmínky:** K systému je přihlášený oprávněný Vedoucí.

**Tok událostí:**

1. PU začíná požadavkem na změnu údajů zaměstnance.
2. INCLUDE(*najít údaje o zaměstnanci*).
3. Systém zobrazí aktuální údaje zaměstnance.
4. ...

**Výstup. podmínky:**

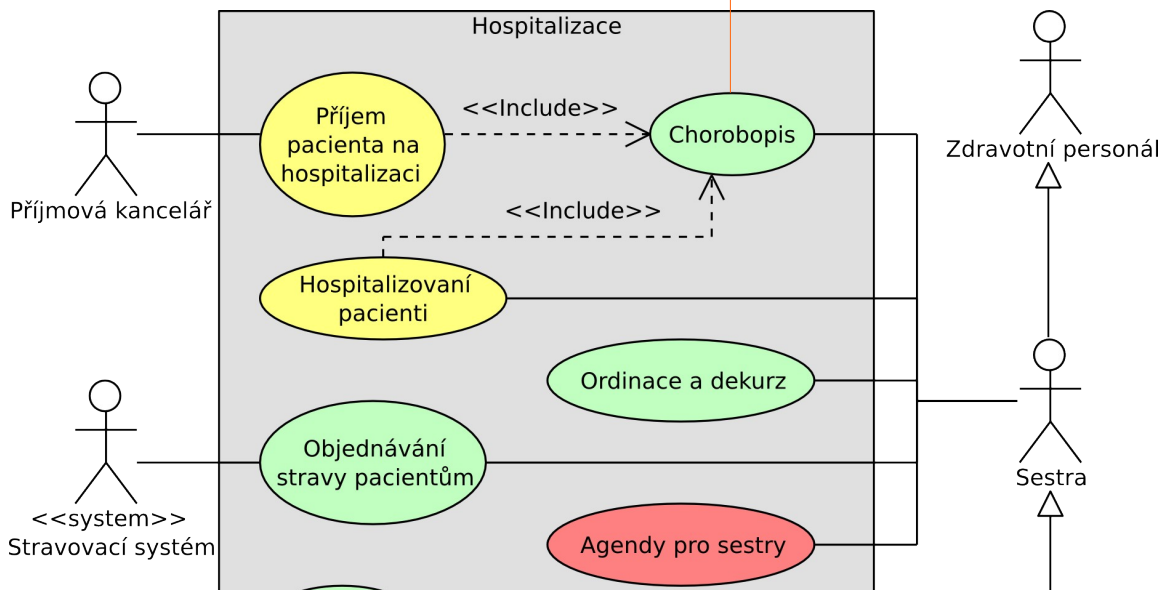
# Příklad z praxe: <<include>>



UC dostupný pouze přes jiné UC

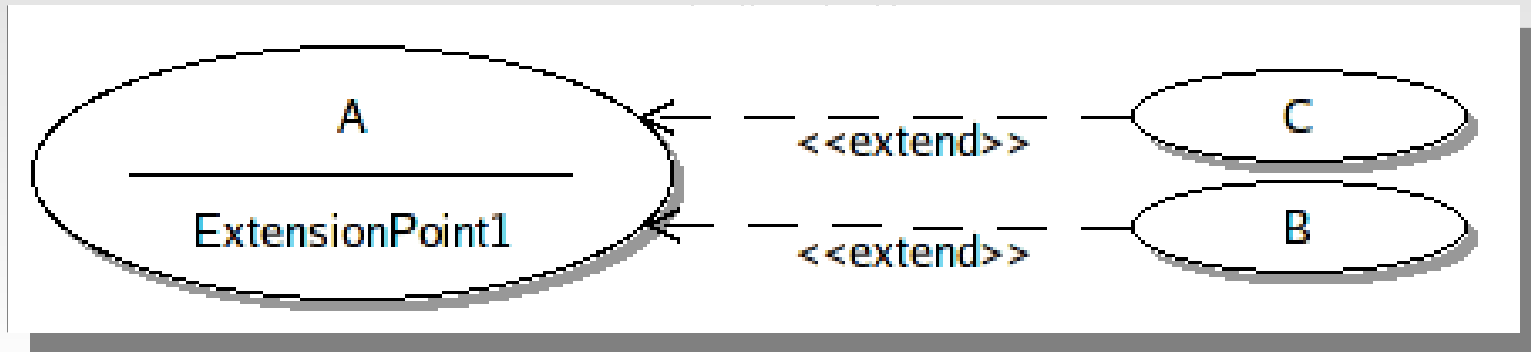
Zvýraznění specifické functionality  
! používat s rozvahou !

UC dostupný i přes GUI



# Vztah <<extend>>

- Další názvy: *rozšiřuje*
- Vztah mezi dvěma případy užití
- Pro vložení **nového chování**, volitelných částí a pro zpracování chyb



- A je úplný p.u., který v ideálním případě neví nic o existenci svých rozšíření B a C  
=> základnímu scénáři je jedno, kdo ho rozšiřuje
- Umístění rozšiřujícího p.u. je označeno *bodem rozšíření* (extension point)  
=> v diagramu a/nebo v popisu rozšiřovaného p.u. “*ve vrstvě nad tokem událostí*”  
=> rozšiřující p.u. ví, jak se přidat do základního scénáře
- Rozšiřující p.u. B a C jsou často *neúplné* p.u. a mohou mít více fragmentů
- *Podmínka pro rozšíření* je uvedena v popisu jednotlivých rozšiřujících p.u. nebo v podmínce rozšiřujícího vztahu (od UML 2.0)
- Rozšíření se často používají v následných verzích systému (následných inkrementech)

# Vztah <<extend>> (II)



## **Případ užití: RezervujKnihu**

### **Preconditions:**

1. Jsou zobrazeny detailní informace o knize

### **Tok událostí:**

1. Čtenář klikne na „rezervovat knihu“.
2. Systém uloží čtenáře do fronty rezervací a zobrazí výsledek.

- Chceme přidat:
  - Odmítnout rezervaci čtenáři, který má nevrácenou knihu
  - Zaslání upomínky těm, kteří již měli rezervovanou knihu vrátit
  - Uložení pokuty těm, kteří navíc již upomínku dostali

# Vztah <<extend>> (III)



## Případ užití: RezervujKnihu

### Preconditions:

1. Jsou zobrazeny detailní informace o knize

### Tok událostí:

1. Čtenář klikne na „rezervovat knihu“.

<existujiPohledavky>

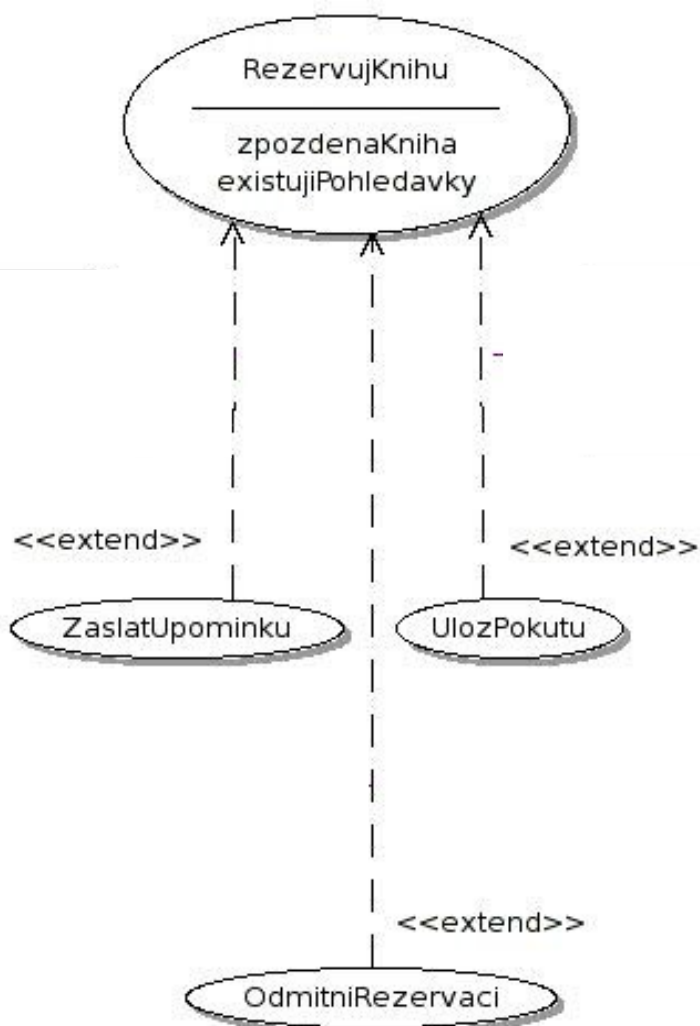
2. Systém uloží čtenáře do fronty rezervací a zobrazí výsledek.

<zpozdenaKniha>

- Chceme přidat:
  - Zaslání upomínky těm, kteří již měli rezervovanou knihu vrátit
  - Uložení pokuty těm, kteří navíc již upomínku dostali
  - Odmítnout rezervaci čtenáři, který sám má v nevrácenou knihu



# Vztah <<extend>> (IV)



## Případ užití: RezervujKnihu

### Preconditions:

1. Jsou zobrazeny detailní informace o knize

### Tok událostí:

1. Čtenář klikne na „rezervovat knihu“.  
<existujiPohledavky>
2. Systém uloží čtenáře do fronty rezervací a zobrazí výsledek.  
<zpozdenaKniha>

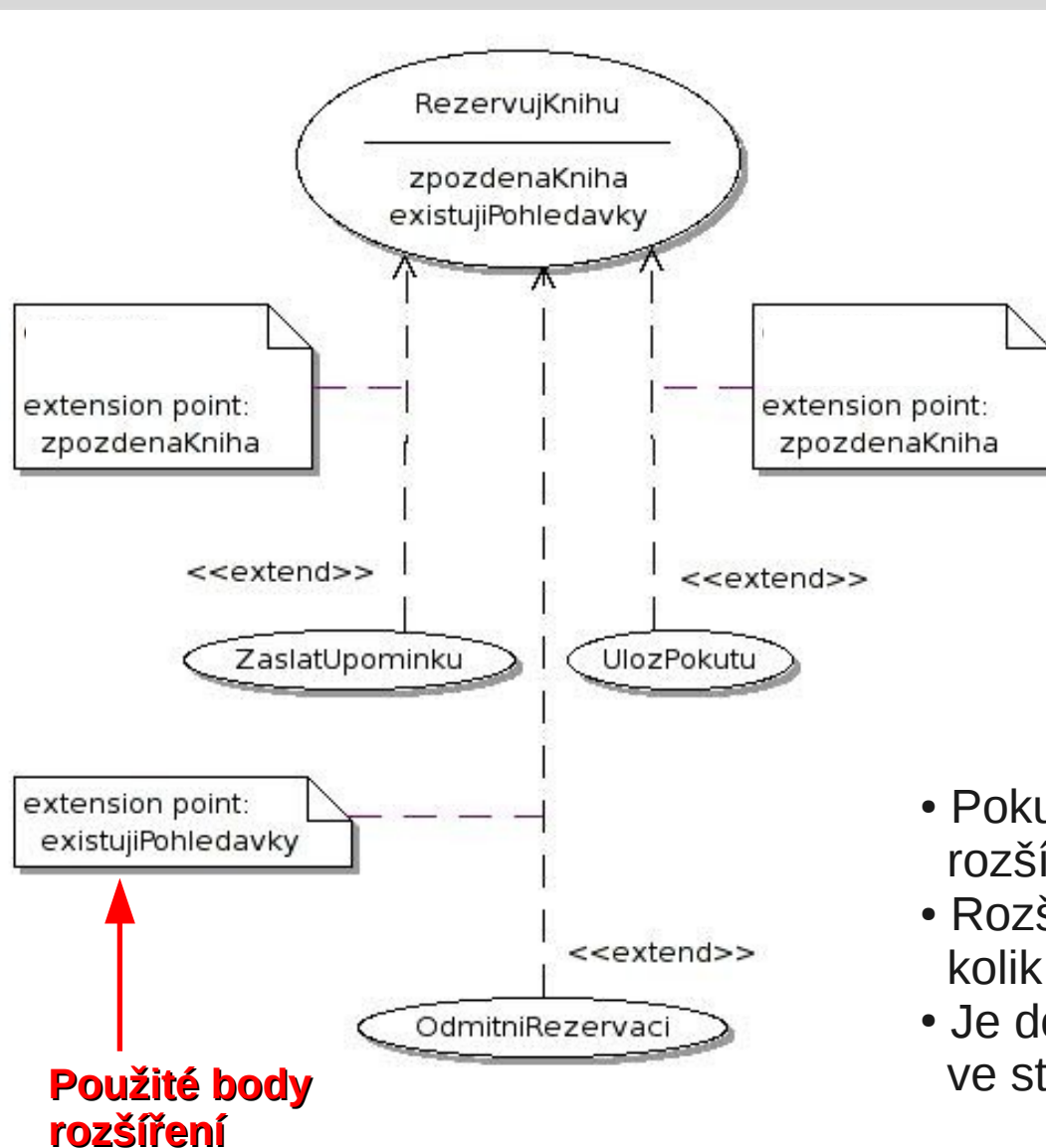
## Rozšiřující případ užití: UlozPokutu

1. PRO VŠECHNY čtenáře, kteří mají knihu půjčenou
- 2.1. IF čtenář měl knihu vrátit AND dostal již upomínku
- 2.1.1. Systém zaznamená u čtenáře výši pokuty
- 2.1.2. Systém pošle čtenáři info o udělené pokutě

## Rozšiřující případ užití: OdmitniRezervaci

1. Systém vyhledá výpůjčky čtenáře
2. IF existuje zpožděná výpůjčka
- 2.1. Systém zamítne rezervaci

# Vztah <<extend>> (V)



## Případ užití: RezervujKnihu

### Preconditions:

1. Jsou zobrazeny detailní informace o knize

### Tok událostí:

1. Čtenář klikne na „rezervovat knihu“.

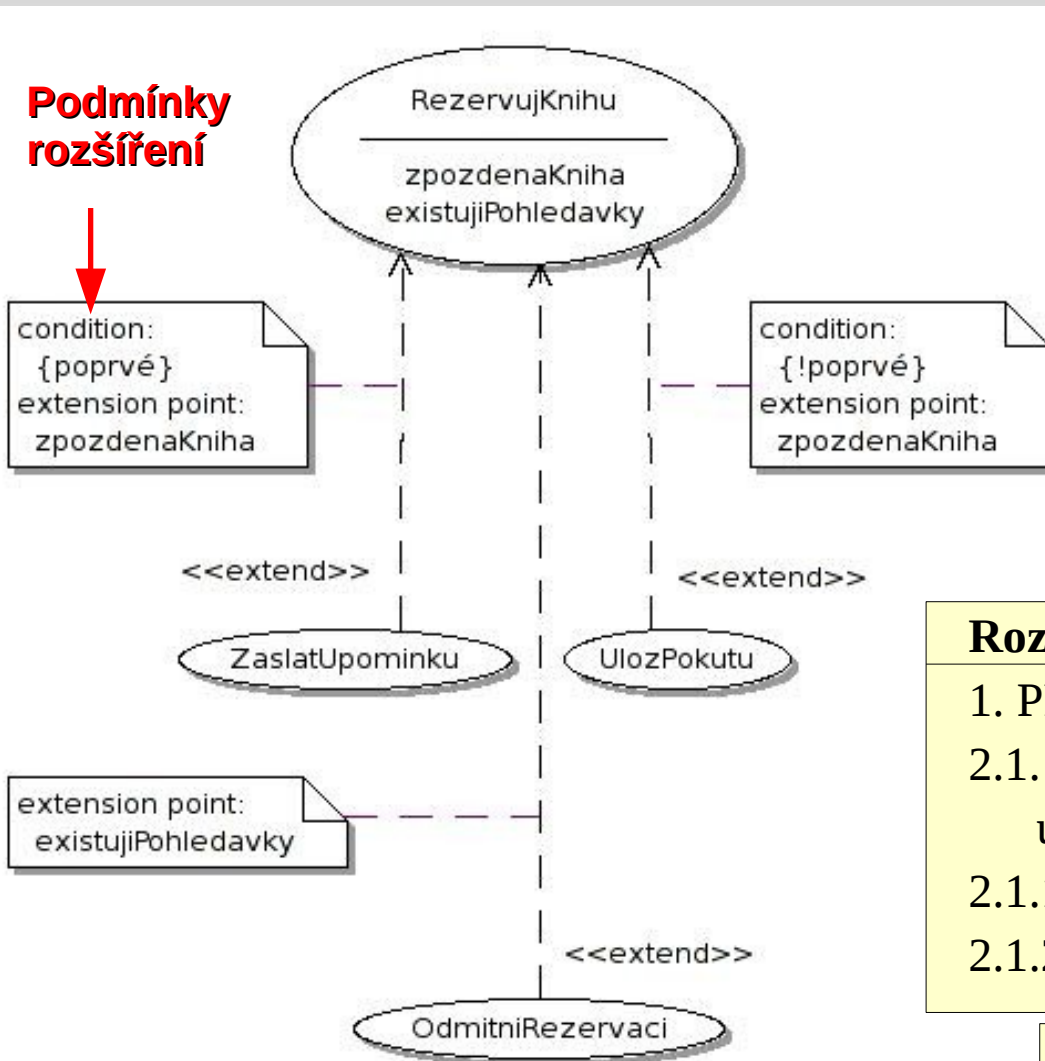
<existujiPohledavky>

2. Systém uloží čtenáře do fronty rezervací a zobrazí výsledek.

<zpozdenaKniha>

- Pokud vztah <<extend>> nespecifikuje body rozšíření, aplikuje se na všechny
- Rozšiřující p.u. musí mít přesně tolik segmentů, kolik bodů rozšíření používá
- Je dovoleno aby dva p.u. rozšiřovali základní p.u. ve stejném bodě, pořadí je pak ale náhodné

# Vztah <<extend>> (VI)



## Případ užití: RezervujKnihu

### Preconditions:

1. Jsou zobrazeny detailní informace o knize

### Tok událostí:

1. Čtenář klikne na „rezervovat knihu“.

<existujiPohledavky>

2. Systém uloží čtenáře do fronty rezervací a zobrazí výsledek.

<zpozdenaKniha>

## Rozšiřující případ užití: UlozPokutu

1. PRO VŠECHNY čtenáře, kteří mají knihu půjčenou

2.1. IF čtenář měl knihu vrátit AND dostal již upomínku

2.1.1. Systém zaznamená u čtenáře výši pokuty

2.1.2. Systém pošle čtenáři info o udělené pokutě

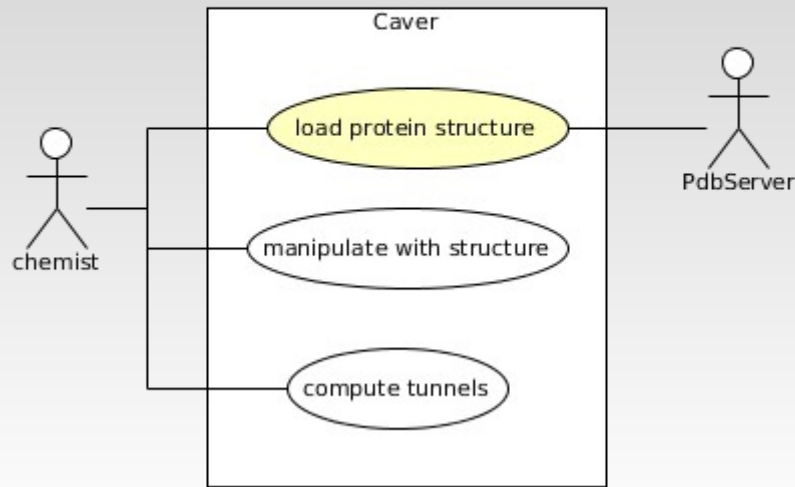
## Rozšiřující případ užití: OdmitniRezervaci

1. Systém vyhledá výpůjčky čtenáře

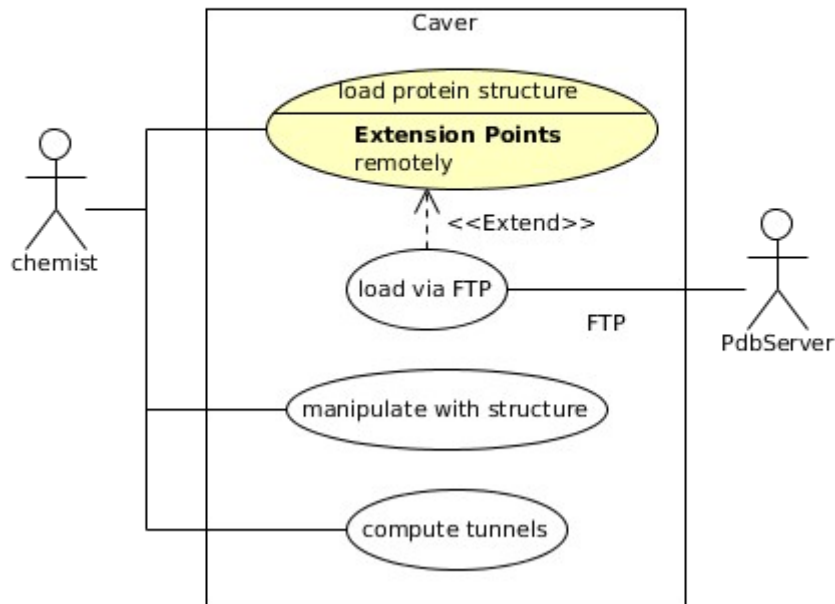
2. IF existuje zpožděná výpůjčka

2.1. Systém zamítne rezervaci

# Příklad z praxe: <<extend>>

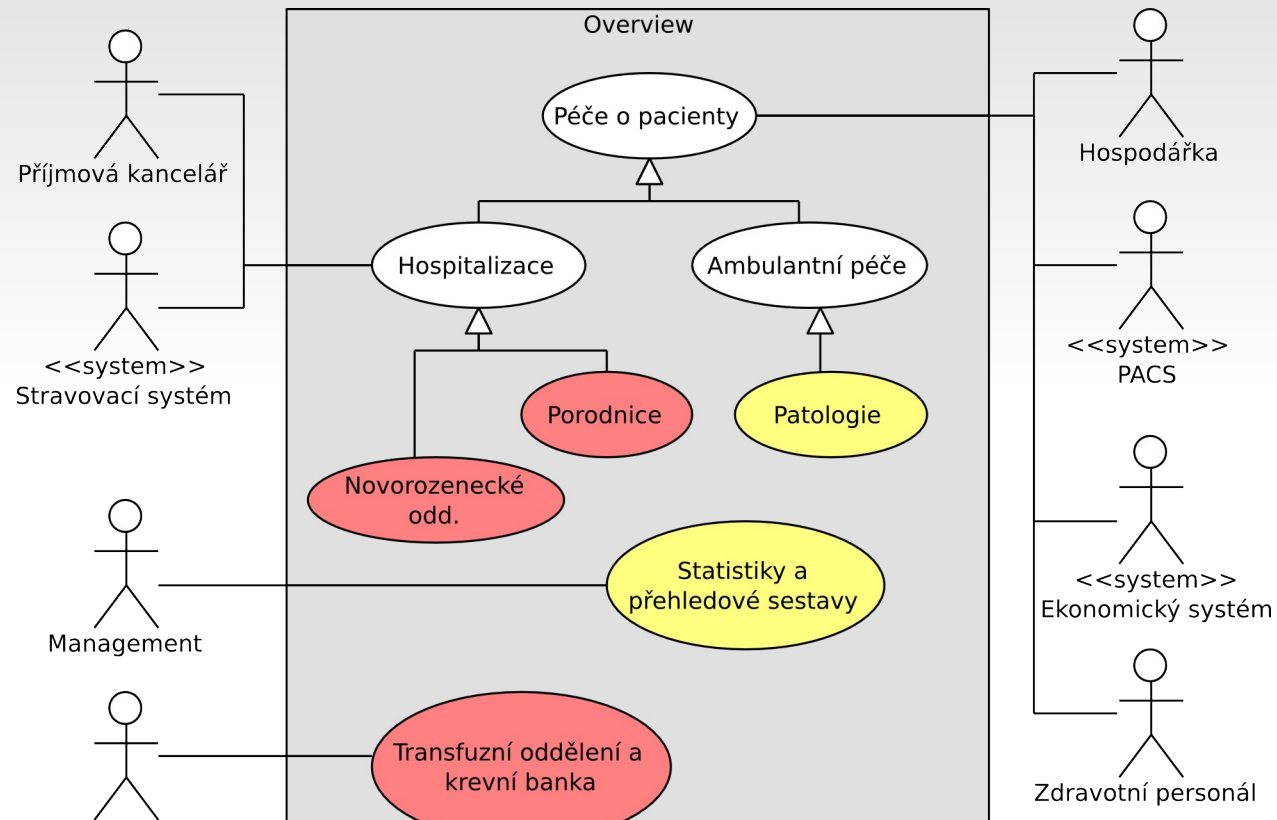


- Primární scénář: nahrání proteinu z lokálního souboru
- Alternativní scénář: Vyhledání a nahrání proteinu ze vzdáleného serveru



- „load protein structure“ obsahuje jediný scénář – nahrání proteinu z lokálního souboru.
- „load via FTP“ obsahuje jediný scénář – vyhledání a nahrání proteinu ze vzdáleného serveru. Tato varianta byla přidána v pozdějším inkrementu.

# Dědičnost případů užití



# Dědičnost případů užití (II)

- Specializovaný p.u. může

- dědit rysy z rodiče
- přidávat nové rysy
- přepisovat (měnit) zděděné rysy

Rys	Dědění	Přidání	Změna
Vztah	A	A	N
Bod rozšíření	A	A	N
Vstupní podmínka	A	A	A
Výstupní podmínka	A	A	A
Krok v toku událostí	A	A	A

- V dokumentaci specializovaného p.u. se doporučuje následující notace, kdy v závorce je číslo, které odpovídá rysu z rodiče

## Rys je ...

## Příklad

Zděděný beze změny

3. (3.) Zákazník vloží požadovaná kritéria.

Zděděný a přechíslovaný

6.2 (6.1) Systém sdělí zákazníkovi, že odpovídající produkt nebyl nalezen.

Zděděný a změněný

1. (o1.) Zákazník vybere „vyhledej knihu“.

Zděděný, změněný a přechíslovaný

5.2 (o5.1) Systém zobrazí stránku s detaily maximálně pěti knih.

Přidaný

6.3 Systém znovu zobrazí vyhledávací stránku.

# Podrobnější model rozhraní

---

Může být užitečné podrobněji dokumentovat rozhraní systému (vymezené pomocí případů užití) např.:

- **Obrazový scénář** (pro grafická uživatelská rozhraní)
  - pouze náčrtky ukazující různé obrazovky
  - diagram ukazující posloupnost obrazovek
  - prototyp uživatelského rozhraní (prověření motivace !!!)
  - Viz aktivita UP „Prototype user interface“
- **Definice protokolu** (pro systémová rozhraní)
  - např. při výměně XML zpráv definujte tyto zprávy
- **Interakční diagramy** na různých úrovních abstrakce
  - mezi aktéry a systémem, asynchronní události
  - popis vlevo
  - strukturovaná angličtina nebo závorky pro různé cesty
  - různé úrovně abstrakce !

# Rady pro modelování případů užití

---

- Odvodte případy užití z firemních procesů nebo procházením seznamu aktérů
- Jako první identifikujte primární scénáře
  - zamyslete se nad vztahy <<include>>
  - odsouhlaste s uživateli
  - **system by měl být optimalizován pro primární scénáře**
- Soupis alternativ provádějte až po akceptaci základních p.u.
  - zamyslete se nad vztahy <<extend>>
  - alternativy nepopisujte detailně
- Diagramy se průběžně zpřesňují
- Pokročilé modelování používejte pouze tehdy, pokud to zjednoduší a zpřehlední model



# Rady pro modelování p.u. (II)

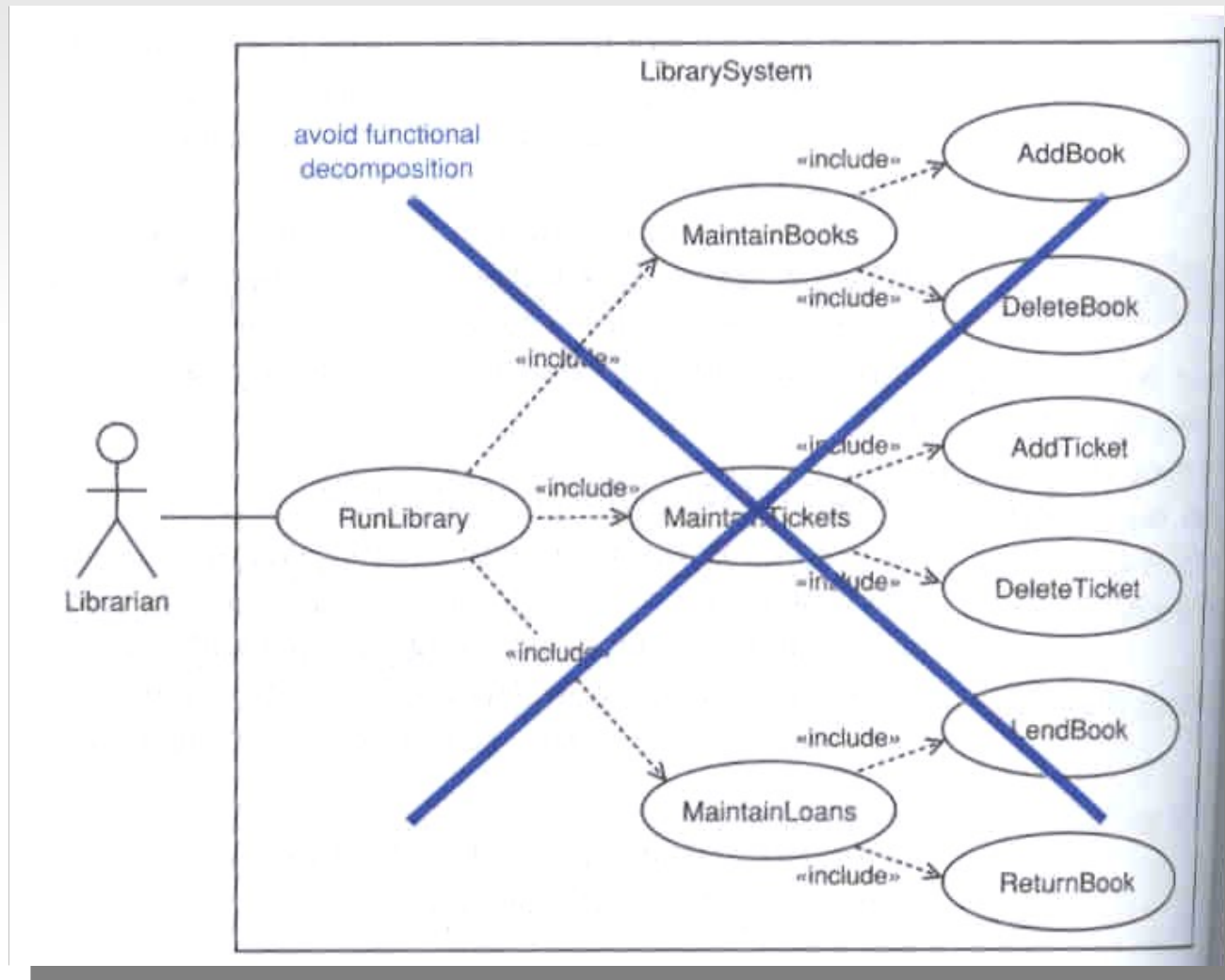
- Vytvářejte krátké a jednoduché p.u.
  - jen nezbytné detaily
  - z praxe: hlavní tok událostí by se měl vejít na jednu stránku A4
- Zaměřte se na **co**, ne na **jak**
  - modelujeme **co** aktéři dělají, ne **jak** to dělají

...  
4. Systém požádá Zákazníka o potvrzení objednávky.  
5. Zákazník klikne na tlačítko OK.  
...

...  
4. Zákazník potvrdí objednávku.  
...

# Rady pro modelování p.u. (III)

- Vyhněte se funkční dekompozici
  - jednoduchý UC model je dobrý model!



# Rady pro modelování p.u. (IV)

- Q: Jak vyjádřím časovou návaznost mezi případy užití na diagramu?
  - Např. mám p.u. „příjem pacienta“ a „propuštění pacienta“ a chci graficky vyjádřit, že nejdříve musím pacienta přijmout a teprve pak ho můžu propustit.
- A: **NIJAK!** Případy užití jsou „okna“ do systému. Časovou návaznost musí řešit kód uvnitř systému. Ten ale teď nemodelujeme.
- Q: Lze vyjádřit časovou návaznost mezi případy užití v dokumentaci případů užití? Jak?
- A1: Implicitně v popisu toku událostí
  - „Propuštění pacienta“ obsahuje např. krok „systém zobrazí seznam hospitalizovaných pacientů“ => nepřijatého pacienta nemůžeme propustit (není v seznamu)
- A2: Pomocí vstupních podmínek
  - Např: „pacient musí být na seznamu hospitalizovaných“

# Shrnutí případů užití

---

- **vymezení problému** (iterativní proces)
- odhalení **hranic systému**
- diskuse **požadavků s koncovými uživateli**
- specifikace **vnějšího chování** systému, funkčních požadavků pro **vývojáře systému**
- výchozí bod pro **návrh** (diagramy tříd, interakční diagramy)
- **rozdělení podrobnější analýzy** požadavků podle případů užití
- **rozdělení konstrukčních** iterací podle případů užití
- odvození **testovacích případů**, odvození **uživatelské dokumentace**
- znovupoužití, architektura (viz literatura)
- velikost projektu a **odhad ceny** (viz kniha G. Schneider)
- .....