Vývoj IS v malých firmách

Pro malé a střední firmy

Lze mnohdy využít i jinde (e-gov)

Informační systémy opakování

-Informační systém (IS) je systém umožňující ukládání, získávání a presentaci informací. IS je systém, tj. strukturovaný komplex technik, nástrojů, a zdrojů umožňující získávání, ukládání a poskytování informací uživatelům a jiným systémům. Výstupem IS mohou být přímo rozkazy osobám a signály procesům reálného světa (avionika letadla, reaktor, …) i kombinace obojího.

-IS nemusí využívat SW, my se budeme zabývat případem, kdy IS využívá softwarovou podporu. IS jsou základním nástrojem

- –globalizace světové ekonomiky,
- – informatizace společnosti a
- –změn ve výrobních procesech a
- –změn ekonomických procesů

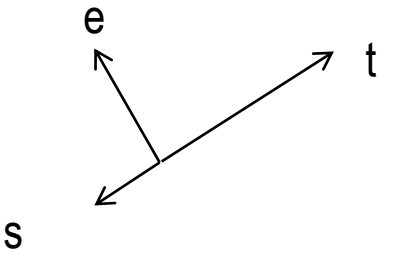
• Informační systém obvykle obsahuje databázový systém, klíčové je poskytování informací!!

•Informační systémy jsou human oriented!

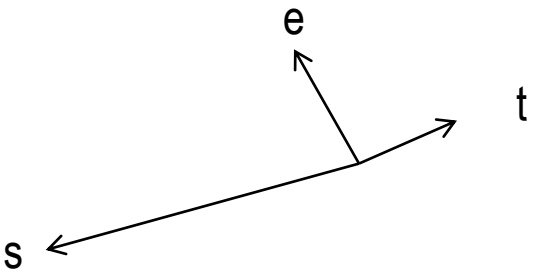
•Tvoří více než 90% SW systémů

Zvláštnosti IT, hlavně IS

- Klasická technika IS na počátku, váha aspektů
  - Ekonomický užitek
  - Jak technicky udělat
  - Sociální aspekt slabý



- Informační technologie
  - Sociální aspekt téměř vždy přítomen, mnohdy zcela zásadní
    - Je zdrojem obtížnosti a problémů
      - Nejasnost cílů
      - Postranní zájmy (viz evaluaci škol)
      - Potřeba globálních propojených měnících se systémů
      - Existence sociálních a zdravotních rizik
    - Zásadním způsobem ovlivňuje procesy vývoje

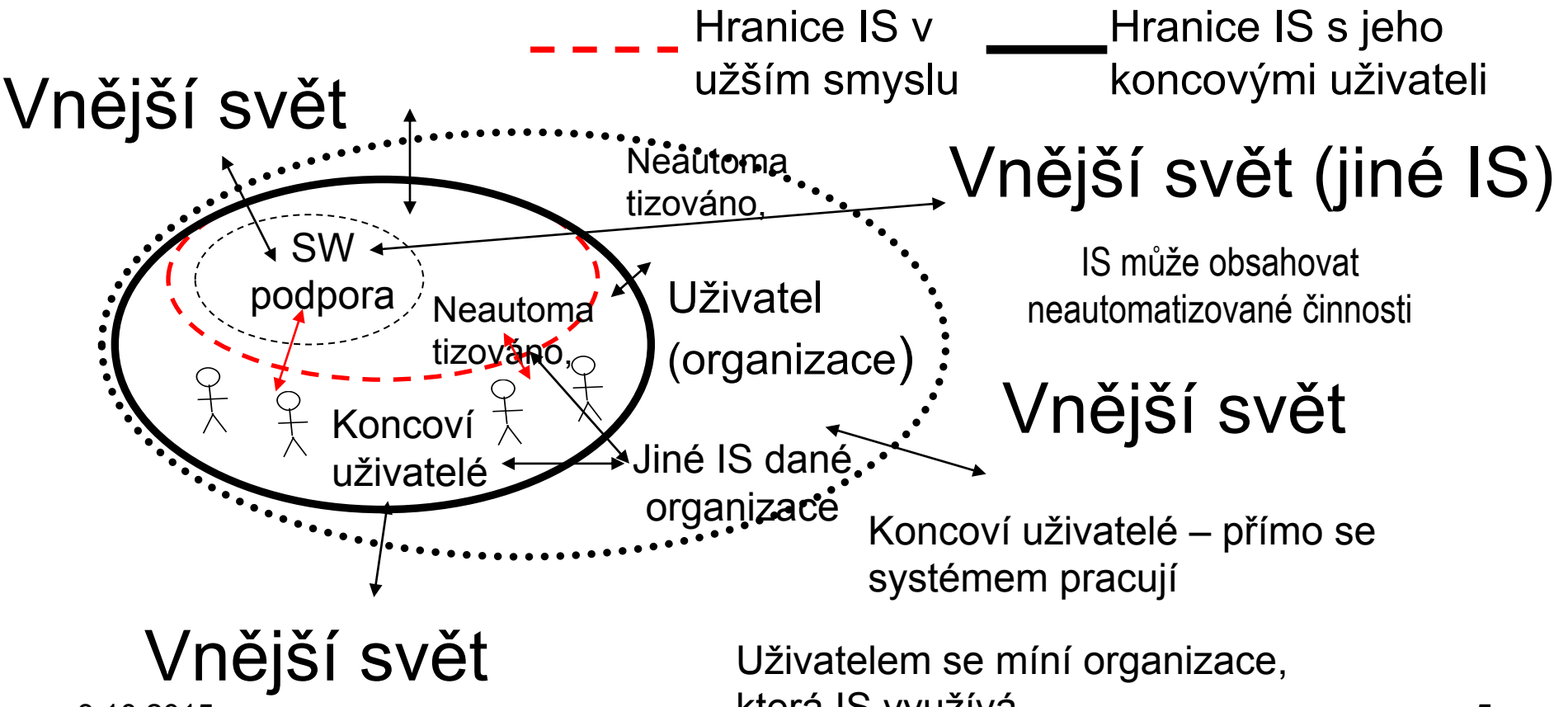


  - Ekonomický i společenský užitek
  - Jak technicky udělat, už se to umí, jsou nástroje

Systém opakování

- Zdroje (lidé, materiál,data)
- Prostředky (stroje, nástroje, znalosti a dovednosti)
- Struktura: Vazby mezi částmi (komponentami)
- Procesy umožňující za daných podmínek dosahovat určité cíle, u IS poskytovat informace, doporučovat opatření

Informační systém je vždy součástí většího systému zahrnujícího i lidi, neautomatizované činnosti **a jiné systémy,**



Hlavní problémy IS opakování

- **Požadavky nejsou zřejmé a navíc se mění**
- **Zahrnuje přímo lidi a nepřímo další ovlivňuje**
- Může být v zájmu i proti zájmům lidí, ty nemusí být zřejmé
- Specifikace požadavků, efekt bývá často jinde, příklady
  - Automatizované dílny,
    - Nikoliv úspora lidí v dílně a vyšší výroba, ale lepší data o výrobě
    - Tím bylo umožněno lepší řízení celé firmy a zvýšení výroby větší, než se čekalo
  - Spotřeba papíru a IT, očekávalo se snížení
    - Spotřeba papíru roste
  - Cíl drahých škol
    - Nejsou to primárně znalosti!!!, jsou to budoucí kontakty s bohatými (⇒ podcenění znalostí a dovedností)
    - Cílem jsou osobní kontakty
- Řízení projektu, představa, že se projektuje jediný kus (tj. IS) a ten se vylepšuje
- Měření přínosů, opomíjené problémy
  - Nezapomínat, že osobní styk je obvykle nenahraditelný (řeč těla, citové vazby, často i rychlost procesu..)
  - Přeceňování efektů automatizace, dělání alibi (dělají to tak jinde, udělám to stejně)

Zásadní požadavky

- IS může obsahovat i neautomatizované části, Automatizace jen když je to smysluplné
  - je to často porušovaná samozřejmá zásada, SW podpora by proto měla být co nejmenší a budována postupně, to není v souhlasu s obchodními zájmy výrobců SW
  - Je nutné umožnit ruční kontrolu a ruční zásahy, chceme-li vyžadovat odpovědnost lidí za podnikové procesy, to se propaguje jako agilita,
    - Je nutno často ad hoc řešit průšvihy
    - Je zpravidla nutné a je to i výhodné umožnit místo automatizovaných akcí i jejich ruční alternativu
    - Je to výhodné pro provoz, agilitu vývoje i byznys procesů i pro údržbu

Zásadní požadavky

- Podnik ani stát nelze nikdy plně automatizovat
  - Lidé ani společenství lidí nejsou počítače
  - Ani svět není deterministický
  - týká se hlavně strategie
  - Všechny možné průšvihy nelze detekovat
  - I ty co známe nelze v IS zohlednit - moc ty to stálo a příliš dlouho trvalo, svět se mění
- IS spolupracuje s lidmi i s jinými IS a SW systémy v organizaci i mimo ni, to je nutné zohlednit
- Správné zapojení lidí a neutomatizovaných procesů je klíčová vlastnost byznys procesů, především pro SME uživatele

Byznys je vždy spojen s rizikem

- Ne každé riziko je žádoucí mzahrnout do podpory IS
- Byznys procesy zvláště v SME je obtížné, není smysluplné, plně automatizovat
- Enhancive maintenence pro potřeby provozu je vhodné svěřit především „těm dole“

Základní varianty architektur IS

1. Monolit:
   - IS = jedna aplikace z hlediska OS, „jeden program“
   - V podstatě bez černých skříněk Přesněji takové skříňky plní okrajové funkce (volání webovských služeb) a mají malou autonomii.
     - Příklad: Program v objektovém jazyce
2. Komponentově orientovaná architektura, autonomní artefakty
3. Servisně orientovaná architektura (SOA **Základní vlastnost:** p2p síť autonomních černých skříněk (je známo jen jejich rozhraní) komunikující asynchronně (výrobci doplňují další vlastnosti z komerčních důvodů),
4. SOA je zvláštní případ komponentové architektury

IS jsou základním nástrojem globalizace světové ekonomiky, informatizace společnosti a změn ve výrobních procesech a vztazích lidí, opakování

- Většina současných SW projektů se týká vývoje IS, IS jsou integrální částí technologií a dokonce i domácích spotřebičů
- Vývoj IS je složitý i když jsou jednotlivé funkce jednoduché, protože
  - IS mají mnoho funkcí, není jasné, které jsou z funkci důležité,
  - požadavky na funkce se mění
  - IS jsou velké a mění se
  - Obtíže při odhadu toho, jaké mají efekty a tedy i potíže při specifikacích toho, co mají dělat
  - Jsou otevřené
  - Dotýkají se zájmů lidí a zahrnují činnosti lidí
    - Lidé dělají chyby, mají různé zájmy, IS může ohrožovat jejich prac. místa
    - Lidé se musí se zaučovat a jejich práce kontrolovat
    - Systém k nim musí být vstřícný a nevyvolávat snahy o sabotáž
- Mnohdy nelze používat klasické formy vývoje monolitů
  - Musím použít to co je a nemám možnost to přepsat
  - Velké systémy musí mít komponetovou strukturu

IS jsou základním nástrojem globalizace světové ekonomiky, informatizace společnosti a změn ve výrobních procesech a vztazích lidí doplnění

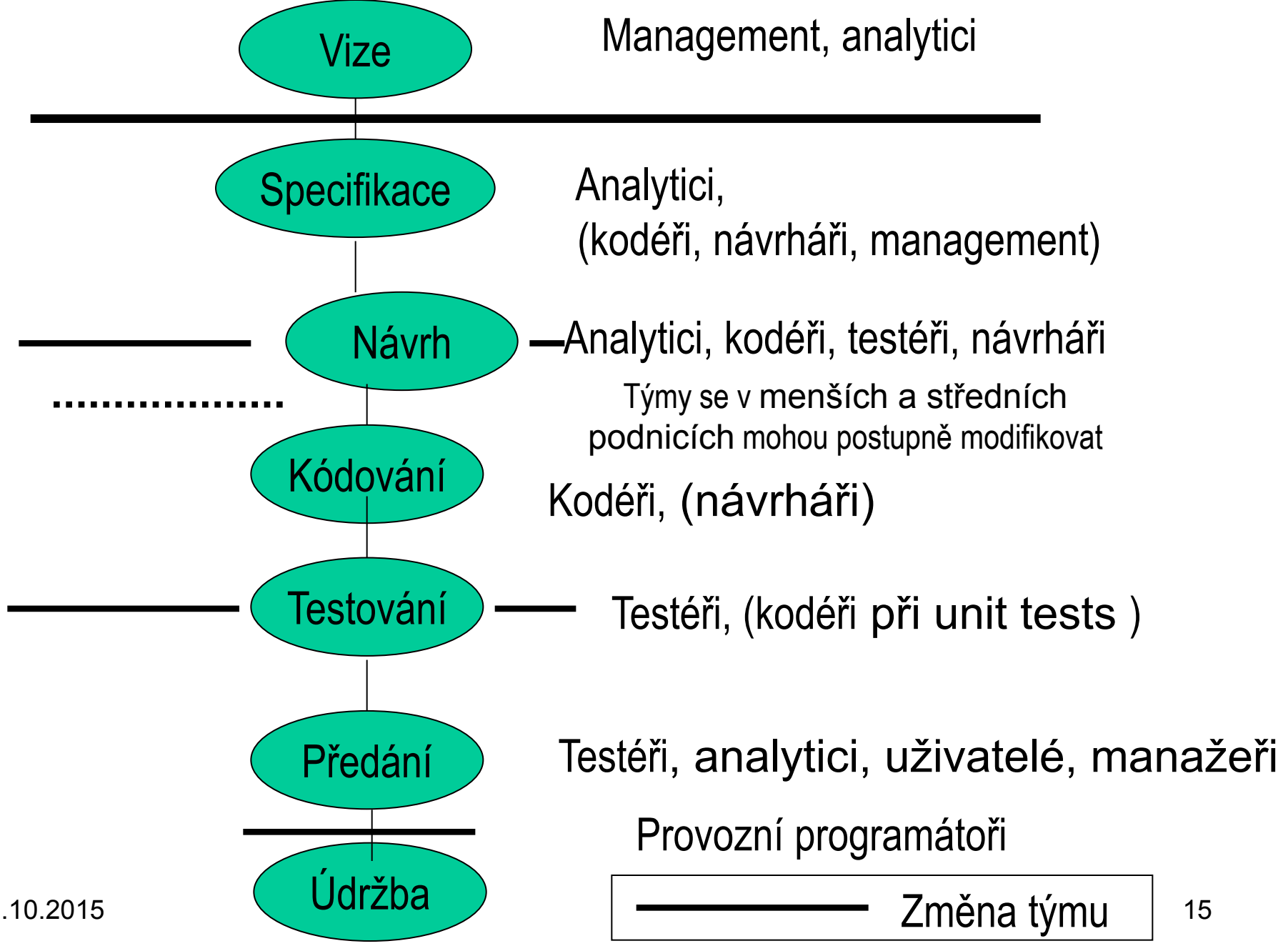
- IS jsou současně součástí procesů reálného světa
  - Některé akce nelze vrátit (?transakce)
  - Akce mohou vyvolat ztráty či katastrofy (radar v Grónsku)
- IS se musí vyrovnávat s vlivem světa a jeho proměnlivých požadavků a leckdy skrytých nebo špatně pochpených zájmů
- Efekty IS jsou často skryté a jiné než se čekalo (dílna), mění se s časem
- IS se proti zájmům rozhodujících uživatelů (stakeholders) obtížně prosazuje (info o kvalitě škol)
- Blbost kvete někdy i při vývoji IS,

Budeme se nejprve věnovat vývoji v SME od začátku metodou vodopádu jako základu pokročilejších technologií a procesů vývoje

Jaké profese jsou potřeba

Je důležité, aby v případě, že pracovník má více rolí, hrál v každém okamžiku jen jednu,

Při testování neopravoval


Vize
Management, analytici
Specifikace
Analytici,
(kodéři, návrháři, management)
Návrh
Analytici, kodéři, testéři, návrháři
Týmy se v menších a středních
podnicích mohou postupně modifikovat
Kódování
Kodéři, (návrháři)
Testování
Testéři, (kodéři při unit tests)
Předání
Testéři, analytici, uživatelé, manažeři
Provozní programátoři
Údržba
Změna týmu

Důvody rekódování malých programů

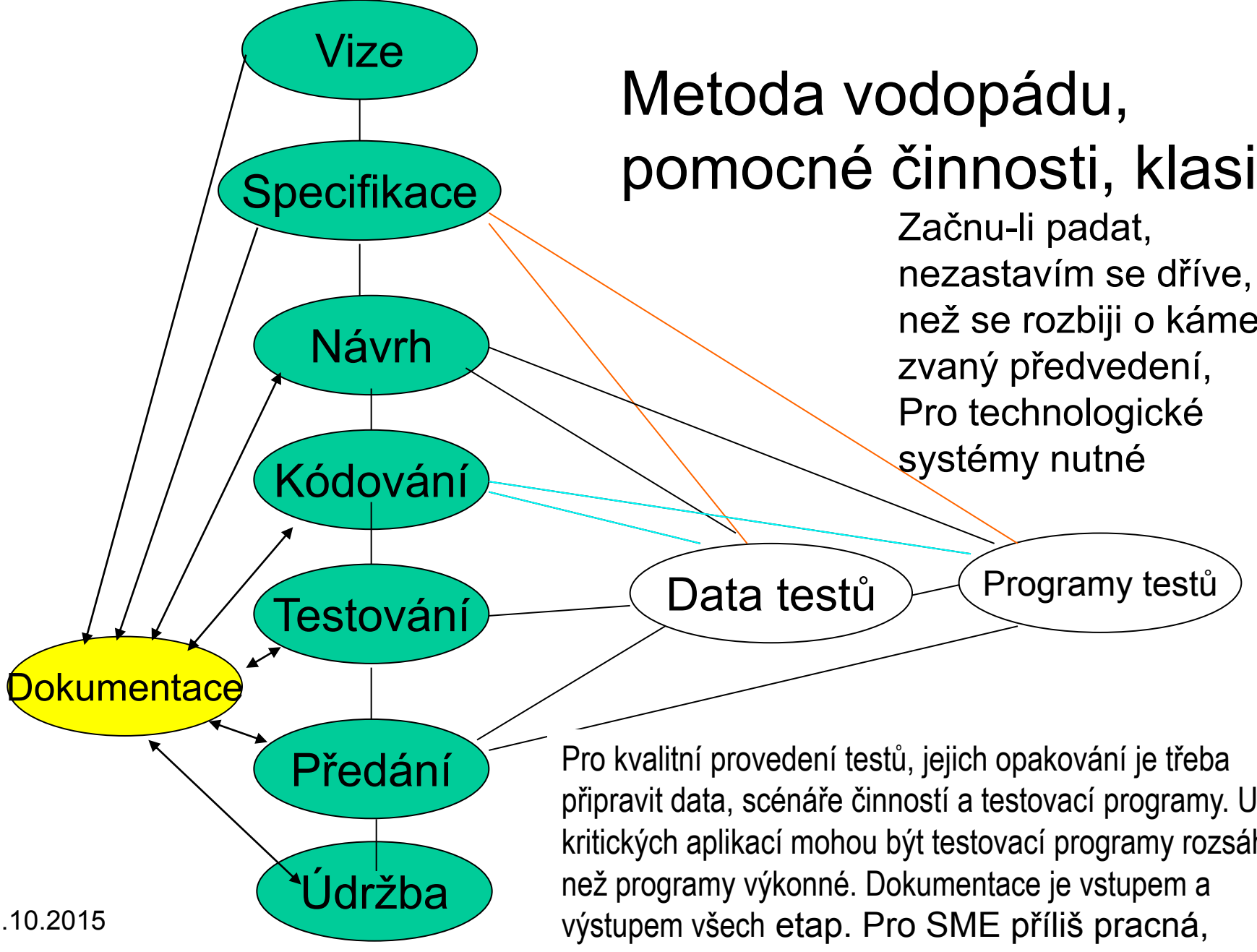
- Skryté (nedokumentované) předpoklady 30%
- Nesprávné požadavky 27%
- Problémy s důvody požadavků (traceability inconsistency) 24%
- Nepřesná terminologie 16%
- Logické chyby 3%

Source: "Experiences using Formal Methods for Requirements Modeling." Easterbrook, et al.

Úzké místo je specifikace požadavků a často také kvalita vize, vize se často ani nedokumentuje!!!

Nedostatek je analytiků, odborníků na IT a byznys a SW architektů. Ti ale rozhodují o úspěchu i neúspěchu

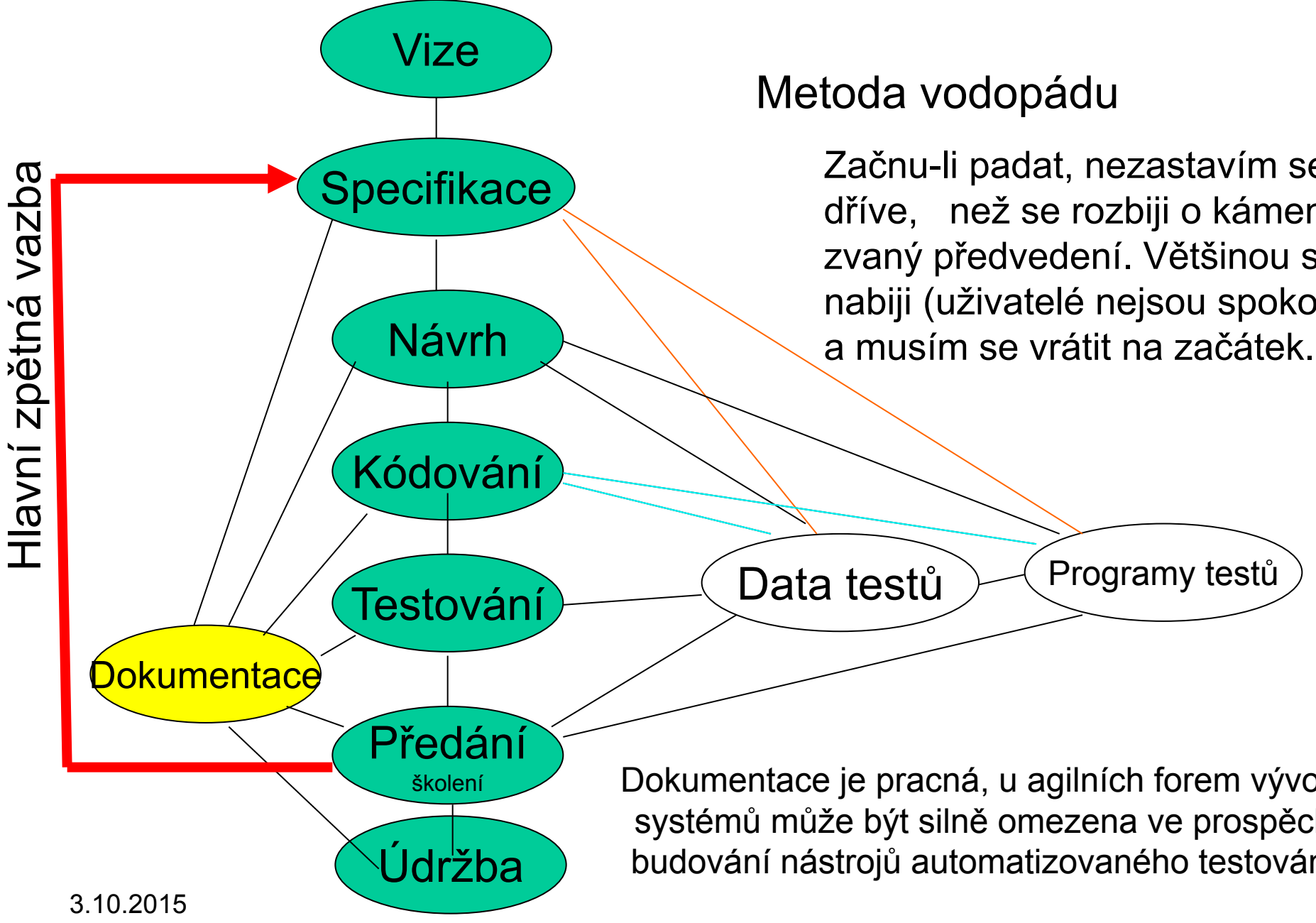
Metoda vodopádu, pomocné činnosti, klasika



Začnu-li padat, nezastavím se dříve, než se rozbiji o kámen zvaný předvedení, Pro technologické systémy nutné

Pro kvalitní provedení testů, jejich opakování je třeba připravit data, scénáře činností a testovací programy. U kritických aplikací mohou být testovací programy rozsáhlejší než programy výkonné. Dokumentace je vstupem a výstupem všech etap. Pro SME příliš pracná,

Metoda vodopádu



Začnu-li padat, nezastavím se dříve, než se rozbiji o kámen zvaný předvedení. Většinou si nabiji (uživatelé nejsou spokojeni) a musím se vrátit na začátek.

Dokumentace je pracná, u agilních forem vývoje systémů může být silně omezena ve prospěch budování nástrojů automatizovaného testování

Trochu terminologie

Verifikace: *Ověření správnosti nějakého dokumentu důkazem či oponenturou.*

Revize (review): Verifikace většího celku formou blízkou oponentuře v běžném smyslu (jiz obhajoby diplomek). Používá se i u feasibility study (studie uskutečnitelnosti)

Inspekce (inspection): Přísně formalizované oponentní řízení pro menší dokumenty s řadou činností a rolí. Probereme později

Walkthrougs: poloformální pročítání dokumentu v malé skupině, variantou je *čtení kódu* prováděné obvykle ve dvojici

Dobře provedená verifikace je velmi účinná - až 80%, odhalí i problémy obtížně detekovatelné testováním

Vyžaduje to ale stoprocentní individuální nasazení, disciplinu a koncentraci, které se obtížně kontrolují a ne všichni jsou jich schopni, Pokus o řešení - inspekce

Trochu terminologie 2

- **Validace:** Ověření správnosti systému nebo jeho částečně funkčního modelu (prototypu) pokusem (testem), někdy virtuální testování
  - Předvedení prototypu (= částečně funkčního modelu systému)
  - Testy částí (programátor)
  - Testy integrační (testér)
  - Testy systému (testér, koncový uživatel)
  - Regresní testování (opakované provedení testů na používaném systému)
  - Test funkcí (testuje se nově integrovaná funkce)
  - Testy předávací, zkušební provoz

Trochu terminologie (norma IEEE std. 1044)

- **Pochybení (error, fault)** - mentální selhání člověka
- **Defekt (defect) –** místo v nějakém dokumentu, které není správně a může případně v kombinaci s jinými defekty způsobit špatnou funkci systému, ať existujícího tak projektovaného.
- **Selhání (failure) –** případ, kdy systém nepracuje správně.
- Pochybení → defekty → selhání
  - Jak by se to popsalo konceptuálně?
- **Chyba** bude označovat pochybení nebo defekt nebo selhání

Vztahy mezi typy chyb

- Selhání způsobené SW mají v podstatě vždy příčinu v pochybení.
  Pochybení způsobí určitý člověk, nebo skupina lidí
- Pochybení způsobí jeden nebo více defektů (ty mohou být závislé) v určitých dokumentech.
- Defekt v určitém dokumentu může být příčinou jiného defektu, obvykle v jiném dokumentu (specifikace → návrh)
- Defekt způsobí jeden nebo více typů selhání
- Ladění a oponentury jsou procesy, které zjišťují, zda nemůže dojít k selhání s cílem odstranit je. Pro každé selhání je nutno najít defekty, které je způsobily. Je žádoucí zjistit, kdo a jak je odpovědný za pochybení

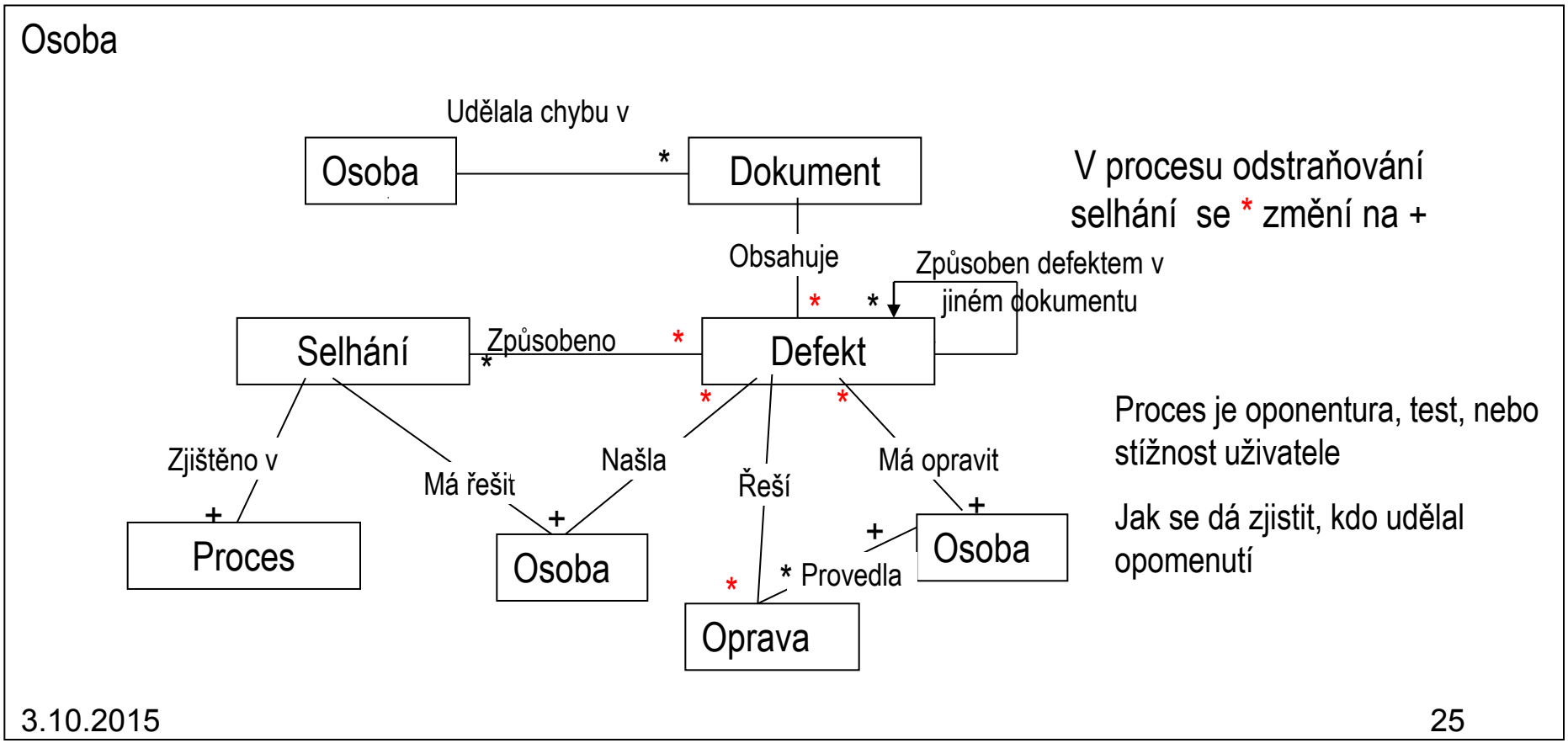
Výstupy verifikací a validací

- Je důležité, aby i verifikace měly výstupy obdobné jako validace (testy), tj. aby se učinil závěr, že jsou důvody k tvrzení, že při platnosti daného dokumentu (dokumentů) nebude vyvíjený systém pracovat správně. Až na výjimky se proto při oponenturách **nehledají defekty** (často stejně budou jinde, než je právě zkoumané místo v dokumentu), nalezení defektu je věc následných prací. **Jinými slovy nesnažím se hned o opravu.**
- Je rovněž žádoucí zpracovávat stížnosti na práci systému od uživatelů obdobně jako výsledky testů (jako seznam selhání).

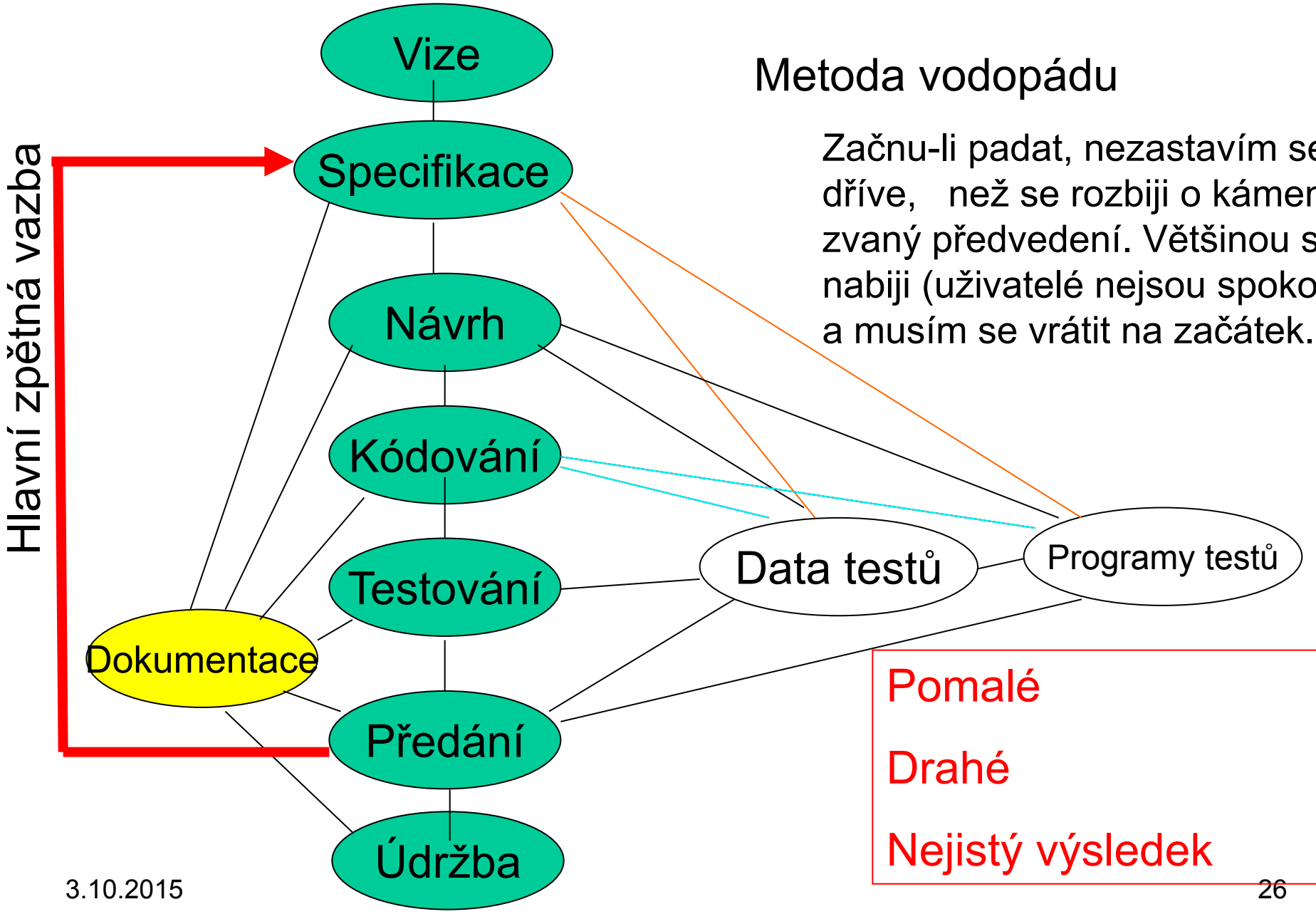
Tyto zásady zvětší účinnost příslušných procesů, mj. tím, že lze vytvořit kvalitní IS pro kontrolu vývoje systému, včetně kontroly kvality oponentur a testů.

Pozor: Tyto informace se nesmí používat k okamžitému postihu pracovníků, snížilo by to účinnost validací a hlavně verifikací

Konceptuální schéma záznamu chyb
ER diagram



Metoda vodopádu



Začnu-li padat, nezastavím se dříve, než se rozbiji o kámen zvaný předvedení. Většinou si nabiji (uživatelé nejsou spokojeni) a musím se vrátit na začátek.

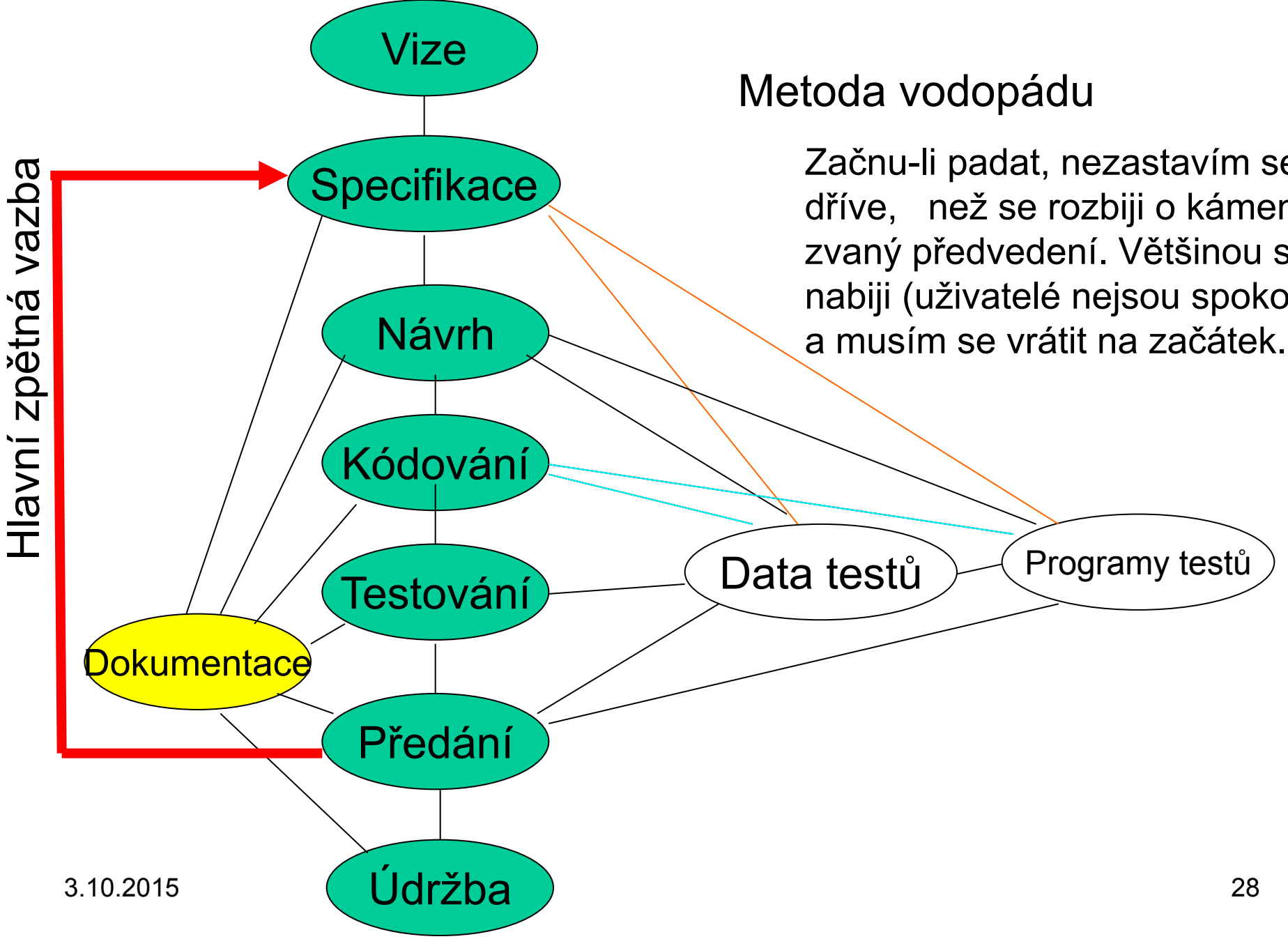
Pomalé

Drahé

Nejistý výsledek

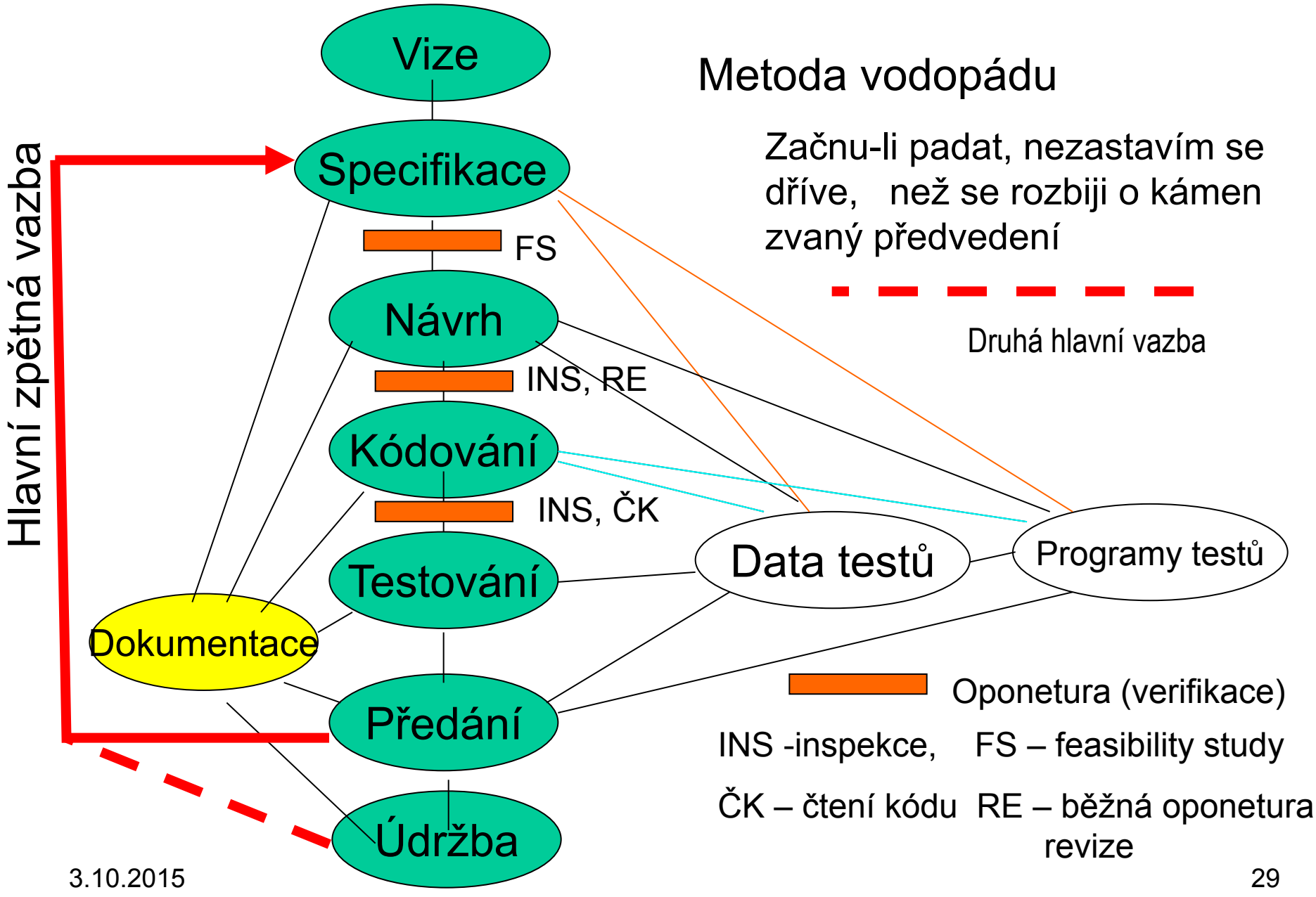
Léčba metody vodopádu pro systémy se sociálním aspektem: Oponentury (verifikace) Oponentní tým: Moderátor, oponenti, zapisovatel, presentátor, „veřejnost“

Metoda vodopádu



Začnu-li padat, nezastavím se dříve, než se rozbiji o kámen zvaný předvedení. Většinou si nabiji (uživatelé nejsou spokojeni) a musím se vrátit na začátek.

Metoda vodopádu



Začnu-li padat, nezastavím se dříve, než se rozbiji o kámen zvaný předvedení

Druhá hlavní vazba

Oponetura (verifikace)

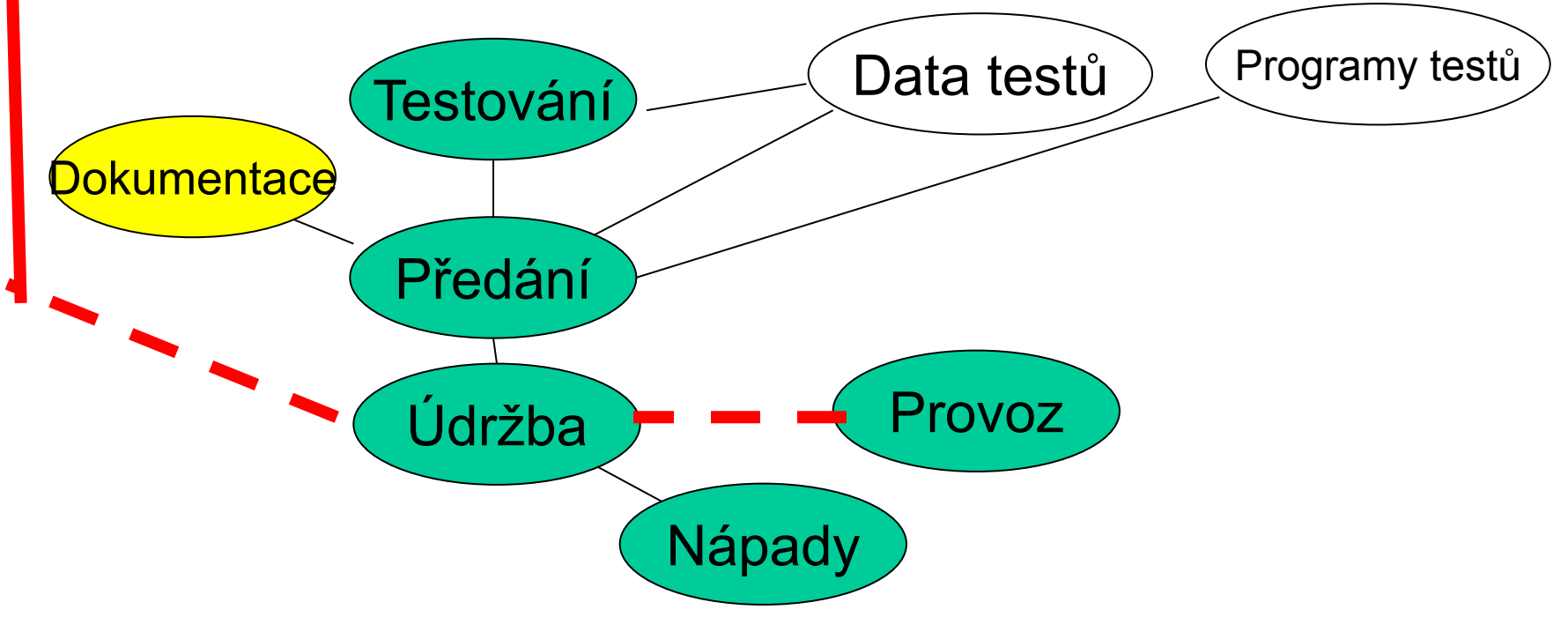
INS -inspekce, FS – feasibility study

ČK – čtení kódu RE – běžná oponetura = revize

Provoz a údržba,

zkušenosti provozu jsou nedostatečně využívány (co se vyplatí)

- Zednodušený vývoj (část nekvalitní)



Účinnost testování

- Unit test 24%
- Systém 36%
- Regresní 23%
- Integrační 24%

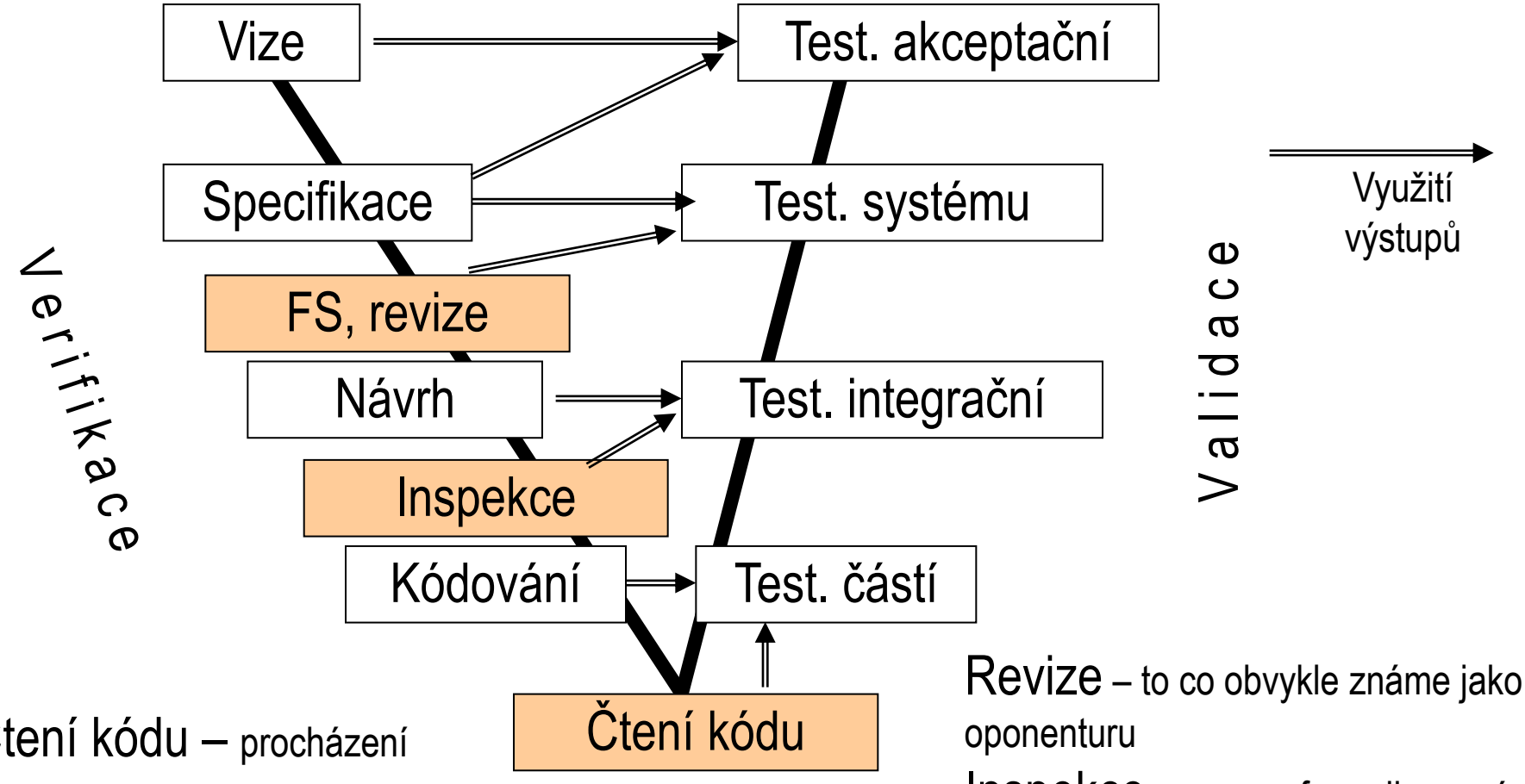
Source: “Software Quality Analysis and Guidelines for Success”, Capers Jones, 12/96

Procenta výskytu a cena odstranění chyb daného typu

	Četnost	Náklady
Chyby ve specifikacích	56%	82%
Chyby v návrhu	27%	13%
Chyby v kódování	7%	1%
Jiné chyby	10%	4%

Source: “An Information Systems Manifesto”, James Martin

V-schema



Čtení kódu – procházení textu ve dvou

Revize – to co obvykle známe jako oponenturu

Inspekce – vysoce formalizovaná oponentura

Efekt verifikace

Pokud je dobře provedena je stejně účinná jako testování. Může ale snáze odhalit některé typy chyb.Tvrdí se že účinnost oponentur je až 80%, dobré je i 25%. Účinnost ladění tří etap prováděním verifikací (účinnost třikrát 0.25) a tří validací (v předchozím slajdu jsou účinnosti unit test 0.24, integration test 0.24 a systém test 0.36) se zvýší až na 85%. To většinou vyhovuje, protože zůstávají skryty defekty, které jen výjimečně vyvolají při provozu selhání.

Efekt verifikace

Skutečně podle vzorce

$$P_{odhalí} = 1 - \Pi_i (1-P_{odhalí-i})$$

takže

$$P_{odhalí} = 1-0.75*0.75*0.75*(1-0.24)*(1-0.24)* (1-0.36)=0.84$$

Problém je v kvalitě provedení verifikace. Verifikace se obtížně kontrolují a závisí na kvalitě nasazení a na pozornosti účastníků oponentur. Možná je kontrola kvality oponentur po jisté době v testech, viz konceptuální schéma pro tytpy chyb

Efekt validace

Oponentury y nalezených závad na tisíc řádek

Testy x závad na tisíc řádek

Y/X co největší

Kde je úzké místo při vývoji SW

Víme, že problém jsou specifikace,
ukažme si příklady

Vlastní zkušenost přednášejícího

Hlavním zdrojem problémů jsou *nedokumentované a hlavně opomenuté předpoklady*, *nejasné požadavky* a požadavky u nichž se neví, jak se k nim došlo (nelze je vystopovat). Ty vedou i k neúplným a neadekvátním specifikacím a jsou odpovědné i za nekonzistence a ztrátu znalostí o tom, proč se věci řeší tak a ne jinak (traceablity). Tj. podobně jako v předchozím slajdu odpovídaly za 80% problémů ve specifikacích.

Kdy to neplatí

- Když nejsou problémy se specifikacemi a systém není příliš velký (nelze big bang)
  - To obvykle znamená malou důležitost zapojení lidí
  - Příklady:
    - Komunikační protokoly
    - Řízení technologických procesů, např. avionika, řídící SW auta
    - Dosti často hry
  - Někdy se musí dělat klasický vodopád i ze zákona (podle normy), případně s oponeturami

Další cesty jak zlepšit vodopád

- Zlepšit specifikace, inkrementální vývoj, hotové zlepší specifikací toho, co se má dodělat
- Snížit pracnost dělením na autonomní části
  - Rozdělením se sníží pracnost vývoje
  - Využít hotové
  - Nakoupit cizí
- Zlepšit údržbu (kontinuální vývoj)
- Vyžaduje to úpravy procesů vývoje a SW architektury

Landauer 1993, The trouble with computers

Opakování

- Makroekonomicky se nedá vliv počítačů detekovat, průměrný přírůstek HDP 1973-1993 nižší než v předchozích stejně dlouhých obdobích (ropná krize?)
- Záporný přírůstek produktivity v odvětvích, kde se hodně používá IT (publikační činnost, do jisté míry i v bankách, nejlepší výsledky v materiální výrobě)
- Vysvětlení
  - Těžko měřitelné efekty a nové funkce (platební karty, rychlé vydávání)
  - Ještě se neprojevil vliv (viz třífázový motor, vliv se projevil po 40 létech), brzy dojde k úspoře pracovníků na přepážkách
  - Neumíme měřit (viz nejasnost požadavků), v bankách se objevily nové služby (platební karty, on-line bankovnictví, rychlá publikace knih v editaci)
  - Ovlivněno recesí na počátku devadesátých let (krize i informatiky)

Landauer 1993, The trouble with computers nová fakta dnes

- Další příčiny malého efektu IT
  - Nevhodně používáme IT (zátěž e-mailu, spamy, mnoho hnoje na www, automatizujeme kde co)
  - Špatně využíváme potenciál dat
  - Efekty jinde, než čekáme, a jež proto nedovedeme využít (výrobní systém a obchodní efekty, nové služby bank, rychlé publikování), nové typy služeb
  - Nevhodná legislativa (ÚOOÚ)
  - V r. 1990 byla zrovna recese (to tvrzení, že IT je a nic, jenom zmírňuje, nevyvrací)
- Pravděpodobnost, že se všichni, co do IT investují, mýlí je velmi malá, IT je významný faktor pokroku
  - Všichni se nemohou mýlit pořád, inteligence lidí není právě malá, takže podpora IT není chybná, problém je, že někdy jsou efekty zakázány, jindy jinde než se čekalo, jindy dlouhodobé, …

Projektování IS je obtížné

Některé důvody jsme uvedli

Hodně projektů IS dopadne všelijak

Ztráty v americké ekonomice v r. 1997. Bureau of statistics

- American companies spent $81 Billion (USD) for cancelled software projects
- $59 Billion (USD) was spent on projects which significantly exceeded time and budget estimates
- Studies show failure rates to up to 85%
- 65% of all projects become “runaways” and exceed budget by at least 200%

opakování V roce 1994 skončilo v USA SW projektů v soukromé sféře podle průzkumu Standish Group:

- 16 % v termínu a podle rozpočtu (tj rozpočet nebyl překročen o více než 20%), tj. úspěšných
- 32 % zrušeno před dokončením
- Více než polovina neúplná nebo dražší až třikrát, termín až dvaapůlkrát

Zdroj *Standish group* 1996, soubor velikosti 8000 projektů

2003 podle aktualizovaného průzkumu již 30% úspěšných, neúspěšných stále mnoho (30%). Možné důvody:

  - Uživatelé mají více zkušeností s IT a tedy rozumnější požadavky
  - Existující systémy slouží jako prototypy vyvíjeného systému
  - I vývojáři se poučili
  - Obě strany lépe spolupracují, chyby ve vizích ale přetrvávají, prima to zatím není
  - Lepší celkové know-how

- Celkově nic moc, v době následující krize byla zahozena až čtvrtina projektů a podíl problémových je skoro polovina a málo se mění

Sledování vlivu různých příčin problémů s vývojem v průzkumu Standish Group

Dotaz ve tvaru: Pro projekty, které skončily úspěchem zaškrtněte ty faktory z následujícího seznamu, které byly rozhodující pro takový výsledek (tj. úspěch)

Obdobně postavené dotazy pro ohrožené projekty a pro projekty, které byly zrušeny.

Pro každou skupinu projektů (úspěšné, zrušené, částečně úspěšné) se pak vyhodnotilo v kolika procentech případů byl daný faktor uveden.

Příčiny úspěchů v procentech (Standish, průzkum v roce 1994)

Faktor úspěchu	*% co zaškrtli*
Zainteresovanost uživatelů	18
Podpora managementu uživatele	16
Jasné požadavky	15
Realistická očekávání	10
Správná dekompozice úkolu	9

Všimněme si, že mezi příčinami úspěchu není technická kompetentnost vývojářů. Jistě by se pochlubili, kdyby si to mysleli. Problém je v tom, co udělat a jak to manažersky podpořit

Příčiny nedosažení cílů v procentech

Faktor potíží ohrožujících projekt	*% zaškrtlo*
Uživatel neschopen říci, co chce	13
Neúplnost požadavků	12
Mizerná podpora managementů	8
Technologická nekompetentnost	7
Chyběly zdroje	6
Přehnaná očekávání	6
Termíny	4

Příčiny krachů v procentech, Standish 1993

Nekompletní a nejasné požadavky	22
Nezájem + malá podpora uživatelů	12
Chybějící zdroje (krátké peníze i termíny)	11
Nerealistická očekávání	10
Management na to házel bobek	9

Pozorování

- Vliv technických faktorů klesá
- Vliv manažerských faktorů a spolupráce s uživateli roste
- Důležitost omezení úkolu na rozumné jádro
- 2000 – nově se objevil vliv architektury,

Problém metodologie

Pozorování 2003

- V roce 2003 podobný průzkum. Hlavní výsledky:
  - Kvalita řešitelů není problém i nadále, význam kvality řešitelů se dále zmenšil (je samozřejmostí)
  - Dále zesílil význam managementu, nástrojů a technik.
  - Důležitá je standardní architektura systému a *minimalizace jeho rozsahu*
- Hlavní problém je stále ve specifikacích požadavků, v jednání s uživateli a na straně managementu,
- Význam kvalitního managementu roste a stává se klíčovým problémem
- Malá role kompetentnosti řešitelů znamená, že je kompetentnost řešitelů standardem a možná se úkol řešitelů poněkud usnadnil

V roce 2009 skončilo v USA SW projektů v soukromé sféře podle průzkumu Standish Group:

- 32% úspěšných projektů (včas a za plánovanou cenu, nárůst proti 1993)
- 44% ohroženo (méně funkcí, vyšší cena, nedodržení termínu)
- 24% zrušeno (selhalo), to je více, než v r. 2003, asi i důsledek krize
- Poznámka, Metodologie průzkumu je kritizována, hlavní závěry jsou ale asi OK

Průběh v čase, Standish

	2004	2006	2008	2010	2012
ok	29	35	32	37	39
zahoď	18	19	24	21	18
problém	53	46	44	42	43

Důvody úspěchu 2012
Chaos Report 2013

Factors of Success	Points
Executive management support	20
User involvement	15
Optimization	15
Skilled resources	13
Project management expertise	12
Agile process	10
Clear business objectives	6
Emotional maturity	5
Execution	3
Tools and infrastructure	1

Závislost úspěchu na velikosti projektu, chaos report 2013

	Úspěch	Problémy	Průšvih
Malé	76	20	4
Velké	10	62	38

Další důvody: těžko se měří

- Velikost úkolu je PROBLÉMEM
  - Potřeba nových technologií
- Nejasné a nepřímé efekty (globalizace, kvalita dat)
- Existenční ohrožení (střední management) a ztráta mocenských pozic (včetně ztráty informačního monopolu)
- Skrytá rizika (ztratí se staré znalosti, závislost na dodavateli)
- Nevhodná kombinace ručního a automatizovaného
- Nutnost reakce na změny, mentální lenost

Pomohou formalizace v IS? Jen někdy

- Důvody většiny problémů
  - Není jasný cíl projektu, nutnost zapojení uživatelů
  - Neschopnost si uvědomit všechny důležité požadavky, zamlčené požadavky
  - Neschopnost je intuitivně zformulovat-nebezpečí chybné formalizace
  - Často i neexistence vhodného nástroje formalizace,

Částečné řešení v modelech,

- Diagramy, sítě
- Nevím-li co chtít, nemohu to nikdy dostat, model nemodel
- Totéž platí pro zamlčené předpoklady

Stížnosti na vlastnosti vývojářů

Výroky vedoucích SW firem

- Já o ty nafoukané informatiky nestojím. Snáze doučím strojaře programovat, než informatika spolupracovat s uživateli.(Bochum)
- Já nemohu ty arogantní programátory pustit k uživatelům. Hned je svou nafoukaností a neschopností se vyjádřit naštvou a ohrozí tím celý projekt. S uživateli ale musí někdo spolupracovat. (Brno)
- Já jsem celkem s informatiky spokojen, především s tím, co se naučili mimo rámec běžného programování, programování jsme celkem sami schopni je doučit. (Brno, ale v tomto případě jednají se zákazníky jen někteří vývojáři)

Hackerský syndrom
Pozorováno u mnoha informatiků,

- Raději práce s počítači než diskuse s lidmi, nemilují práci v týmu
- Tendence k černobílému uvažování
- Přeceňování čistě informatických znalostí a schopnosti programovat
  - Ve schopnosti kódovat se může málokdo našim programátorům vyrovnat, snadno seženou dobrý job,je to cenné, nemusí ale stačit na celý život (tak do 40)
- Práci považují hlavně za fascinující intelektuální hru
  - Positivní – tvorba volně šiřitelného softwaru
  - Negativní – tvorba virů, trojských koní a někdy přímo kriminalita
  - Statistika a postupy empirických věd jim nevoní
    - Takže nejsou schopni strávně využívat data mining jako funkci vyvíjeného systému a také je použít pro hodnocení vlastní práce

Silně vyvinuto u hackerů – **hackerský syndrom**

Hackerský syndrom, další symptomy

- Podceňování neinformatických oborů, znalostí a schopností koncových uživatelů a významu spolupráce s nimi
- Odpor k filosofii experimentálních věd a zvláště k matematické statistice jako nástroji analýzy dat
  - To je ale nutné k podpoře managementu uživatele i vlastní firmy
  - Bez toho nelze kvalifikovaně hodnotit kvalitu softwaru a zlepšovat procesy vývoje SW
  - Je to nutný předpoklad k porozumění potřebám zákazníků
  - Kvalita dat zásadním způsobem ovlivňuje specifikaci požadavků (kritický řetěz při řízení projektů, rozvrhování, atd. bude diskutováno níže) ISO250xx

Hackerský syndrom, další symptomy

- Neochota pracovat v týmu
- Odpor k dokumentování (někdy i oprávněný - viz zásady agilního vývoje)
- Tendence jít hned na věc a nebabrat se se specifikacemi požadavků (a tedy strategie pokus-omyl)
  - Při moderních principech vývoje, např. při servisní orientaci, lze takový způsob snáze používat (agilní formy vývoje)
- Obtíže při přijímání filosofie moderních směrů v softwaru, např. servisní orientace
  - Neochota aplikovat p2p přístup
  - Neochota používat existující aplikace a produkty třetích stran.
  - Neochota používat „zastaralé“ technologie a kombinovat datový a příkazový přístup a existující aplikace
- Snaha neopouštět kyberprostor (svět počítače)
  - Extrém – rande u počítače s videem,

Rande u počítače

Holka přijde za přítelem, který se věnuje své zábavě na počítači. Přítel jí pustí video a po přehraní filmu se oba rozloučí.

Udělal se následující pokus. Chatovačům se simuloval chyba na serveru. Nakonec se to provalilo a děvčata viníkovi děkovala, že se mohla s kamarády také někdy procházet a vyrazit si s ním do města

Zvláště ostré jsou projevy závislosti na počítačích u pařanů. Závislost na kyberprostoru je pro profesi informatika nežádoucí

- Nedisciplinovanost a nesoustředěnost
- Tendence k přetěžování organizmu až k vyhoření
- Neschopnost spolupráce s neinformatiky,
- Často neschopnost pracovat v týmu

Pokus zorganizoval doc. Jirovský, tehdy na MFF UK Praha

Proč je třeba prevence hackerského syndromu když není pro informatiky problém sehnat dobrý job

- Hackerský syndrom blokuje uplatnění v kvalifikovanějších rolích při vývoji softwaru (analytik, vedoucí projektu, SW architekt)
  - Z toho důvodu se jako vedoucí IT oddělení a IT projektů se často uplatňují lidé, kteří studovali něco jiného než informatiku, nebo ji studovali na VŠE
- Zhoršuje adaptibilitu na změny na trhu práce (dnešní studenti půjdou do důchodu po více než 40 létech, programátorská virtuosita se ztrácí v 35 létech, schopnost analýzy se ztrácí později, viz praxi IBM, kde se zbavují třicátníků, jsou výjimky), je proto velmi žádoucí, aby se neuzavírala možnost uplatnění mimo informatiku. Pro to je dobrou průpravou práce analytika – se uplatní dovednosti jiné než ty, které se uplatní jen při práci s počítačem.
- Délka profesní kariéry mezi 25 až 50 roky. Během této doby dojde k významným změnám na trhu práce.

Proč je třeba prevence hackerského syndromu, když není pro informatiky problém sehnat dobrý job

Těžko se léčí, raději prevence

- Predispozice k HS je občas důvod volby dráhy informatika při rozhodování, co studovat.
  - Nevíme, jaké je rozložení talentů a zda se prevence vyplatí
- Utvrzován běžnou praxí výuky. Ve výuce je obtížné navodit situace ukazující např. potřebu spolupráce s koncovými uživateli nebo dokumentování
- Často spojen s programátorskou virtuozitou – zatím cesta k dobrým rychlým výdělkům (a k podceňováním studia a hlubších znalostí v informatice a hlavně mimo informatiku, studium jen pro jméno a titul).
  - Jak dobře využít? Samostatné podúkoly, hlavně nástřel (ověřování, resp hledání optimální implementace)
- Bacha na takové týpky
  - Rozeštvou tým
  - Nezdokumentují, nedodrží pravidla
  - Odradí partnery

Proč jsou IS složité i když dělají jednoduché věci

opakování

- Mění stav světa
  - Změny nelze vrátit
  - Ohrožení ekonomické a na životech (kritičnost)
  - Blbost lidská (předsudky, mylné předpoklady , nedomyšlenosti)
  - Prvky reálného času a přímého řízeni
  - Ovlivňují zájmy politiků a nakonec i manažerů a řadových pracovníků
    - Sledování kvality škol, dá se využít IT na hodnocení úspěšnosti absolventů, pokud se to neprovede klesne zájem o IT a STEM, ve společnosti
    - Dojde k významné prohře

Proč jsou IS složité i když dělají jednoduché věci

opakování a doplnění

- Fakticky zahrnují i lidi (sociálně politická dimenze)
  - Týkají se jejich zájmů, často podvědomých, vyžadují změnu zvyků
  - Mají politické, pro některé skupiny ne vždy příznivé, důsledky - i celosvětové i místní
  - (Koncoví) uživatelé musí formulovat požadavky společně s vývojáři, to je náročné pro obě strany
    - Obě strany se musí naučit nové věci, je nutná spolupráce různých profesí
    - Potřeba víceoborových znalostí
    - Stálá změna
  - Je nutné kombinovat sílu automatizace s intuicí lidí a balancovat automatizované i neautomatizované činnosti
- Jsou rozsáhlé, dynamické a otevřené

IS jsou složité i když dělají jednoduché věci, doplnění

Někdy se výhody IT přeceňují, jindy nedoceňují

- Slepá víra ve výstupy IS bez ohledu na kvalitu použitých dat(hnůj tam hnůj ven) a chyby ve specifikacích, závislost a počítačích. Význam některých aspektů kvality dat je silně podceňován.
- Snaha o alibi (jinde přece SAP funguje), podceňování nutnosti ručit za svou práci
- Nedohlédnutí efektů nasazení
  - Kvalifikace lidí
  - Organizační změny
  - Efekty jinde, než se čekalo, nevyužití synergie spolupráce lidí a softwarových systémů
  - Efekty se objeví později a závisí na systémových změnách
  - Zvýšení produktivity, možnost na některé práce používat méně kvalifikované síly
  - Ztráta produktivity při nevhodném použití (mail, Internet)

IS jsou složité i když dělají jednoduché věci

Podceňování nevýhod¨používání IS

- Ztráta kontaktu s realitou (IS je velmi často založen na neúplných datech a nezahrnuje celou lidskou zkušenost, potřebu intuice a kritického myšlení)
- Podceňování mezilidské komunikace (v ní až 60% neverbální komunikace, která např. říká jsi mi sympatický/sympatická, city a osobní vazby)
- Ztráta dovedností improvizace a práce ručně při nečekaných při situacích, se kterými vývojáři nepočítali, zpohodlnění až ztráta pracovní inteligence (příklad z půjčovny aut v Texasu)
- Ergonomie a nemoci z povolání

Půjčovna aut v Texasu

- Turisté z Čech přišli platit v hotovosti, pracovníci na to neměli školení a funkci v IS
  - Nebyli schopni vyřídit, nakonec museli půjčit dražší auto proti záloze
  - Při vracení v sobotu odkazovali na pondělí, pomohla pohrůžka, že projde vízum

Přesně natrénovaná činnost je v běžných situacích výhoda, běda při nestandardních akcích, viz hurikán v New Orleansu

IS jsou složité, věcná dimenze

- Běží stále po dlouhá léta ale v důsledku změn v byznysu se musí neustále modifikovat
- Zabezpečení, některé IS mají charakter kritických aplikací (mohou způsobit ztráty životů nebo alespoň peněz)
- Obtíže se specifikací požadavků
  - Nejasnost cílů
    - Nečekané efekty (příklad výrobního systému dílny zlepšující řízení celého podniku)
    - Časté změny celopodnikových informací
    - Nedostatečná analýza potřeb a možností IT a lidí
    - Závisí na principech organizace (jaký typ byrokracie)
- Mnohé IS vyžadují specifické formy vývoje při kterých nejsou využitelné klasické metody a také zavedené nástroje jako je model driven architecture (velké servisně orientované architektury popsané níže - servisní orientace)
- Zvláštní případ O2 nabídla čtvrtinovou cenu než dosavadní provozovatel ICZ, soud řekl, že to O2 lže

Strojová byrokracie

- Přísná hierarchie, „vojenská“
  - Pozice jmenováním
  - Vedoucí má poslední slova a nedílnou pravomoc ve své oblasti
  - Komunikace přes nejnižšího společného vedoucího
    - IS může usnadnit komunikaci podřízených přímo, je ale žádoucí dělat žurnál (log) komunikace

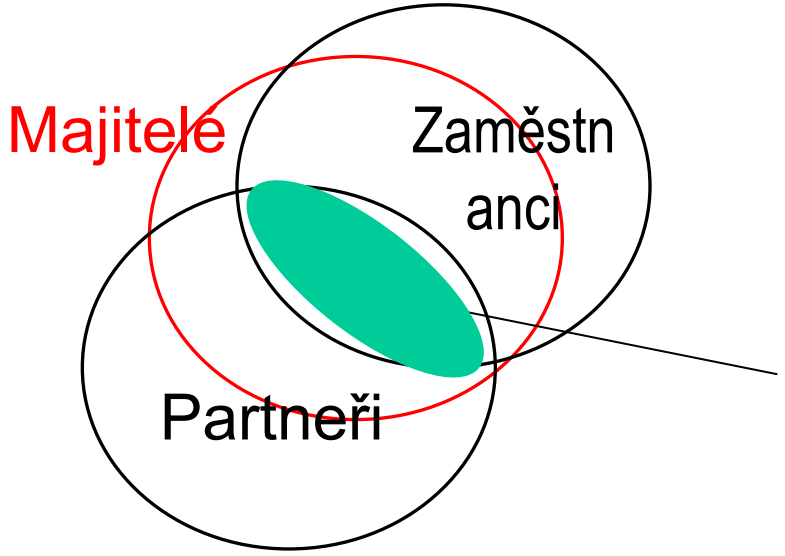
Profesní byrokracie

- Pozice plyne ze vzdělání nebo z volby a je obvykle časově omezena (na pár let)
  - Podmínkou zastávání pozice je „papír“
    - Jmenování profesorem
    - Akreditace, zvolení
  - **Jmenování je na určitou dobu během níž lze obtížně odvolávat, to oslabuje snahu řešit strategické otázky**
  - Univerzita, státní správa , zdravotnictví
  - Vždy spojena se strojovou byrokracií v podřízených složkách
    - Sekce na ministerstvu, úřady, děkanát, ..
  -

Podceňování lidské dimenze
Koalice zainteresovaných v podniku

- **Majitelé**
  - Cíl: maximální zisk
- **Zaměstnanci (i management)**
  - Co nejvíce peněz za co nejméně práce
  - Management bývá i majitelem (vlastní akcie)
- **Partneři v obchodě**
  - Dodavatelé: měkké termíny, vysoká cena, nízká kvalita
  - Odběratelé:Tvrdé termíny, nízká cena, vysoká kvalita
- **Státní orgány** (jen částečně zainteresované a zapojené)
  - Udržet zaměstnanost
  - Mít od koho vybírat daně

Koalice v podniku



Platí pro strojovou byrokracii

podřízeného jmenuje a odvolává nadřízený
komunikace přes společného nařízeného

Společný zájem: dlouhodobá prosperita *IS musí proto být prospěšný pro všechny členy koalice, platí i pro týmy*

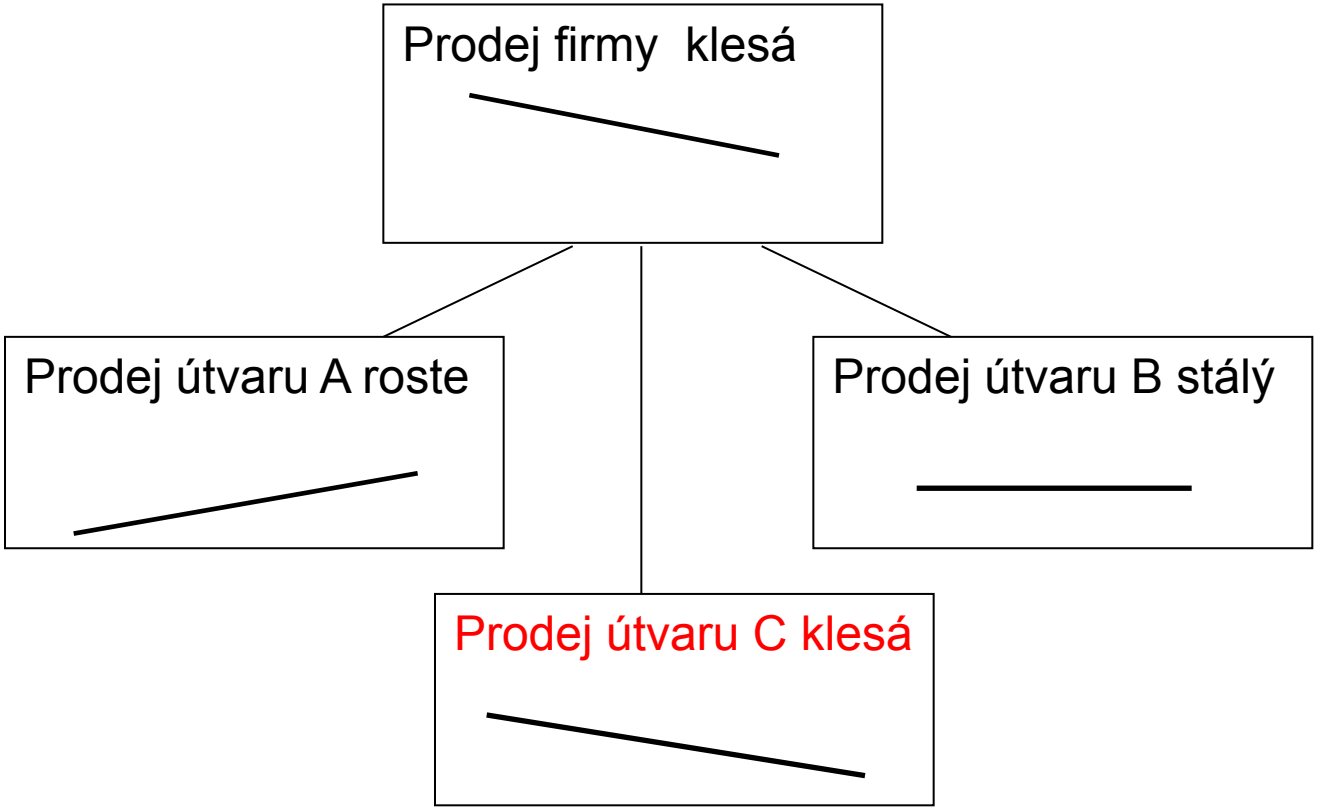
Manažeři akciovek jsou tlačeni investičními fondy ke krátkodobým výhledům preferencí okamžitých výplat dividend, fondy jsou tak trochu na půl cesty k tunelářům

Kauzální diagramy

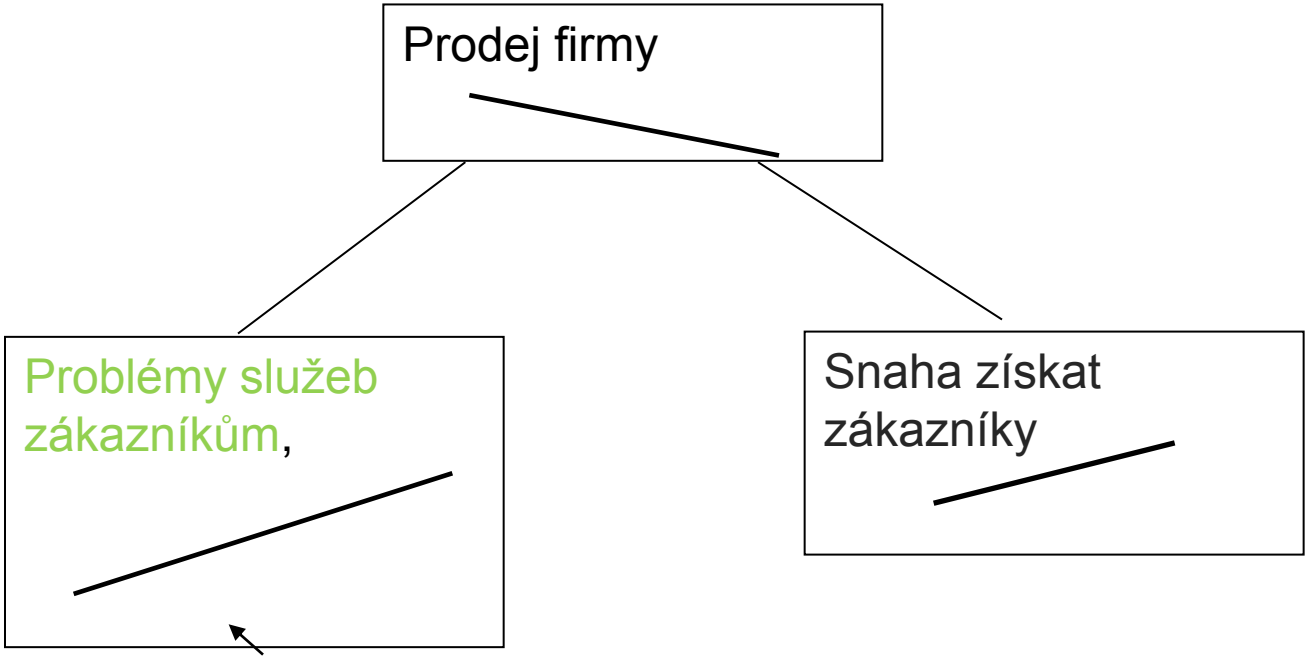
Sledovávání příčin a následků

Předpoklad: Abstraktní objekty se ovlivňují dvěma způsoby: Souhlasně a nesouhlasně. Vlivy tvoří síť

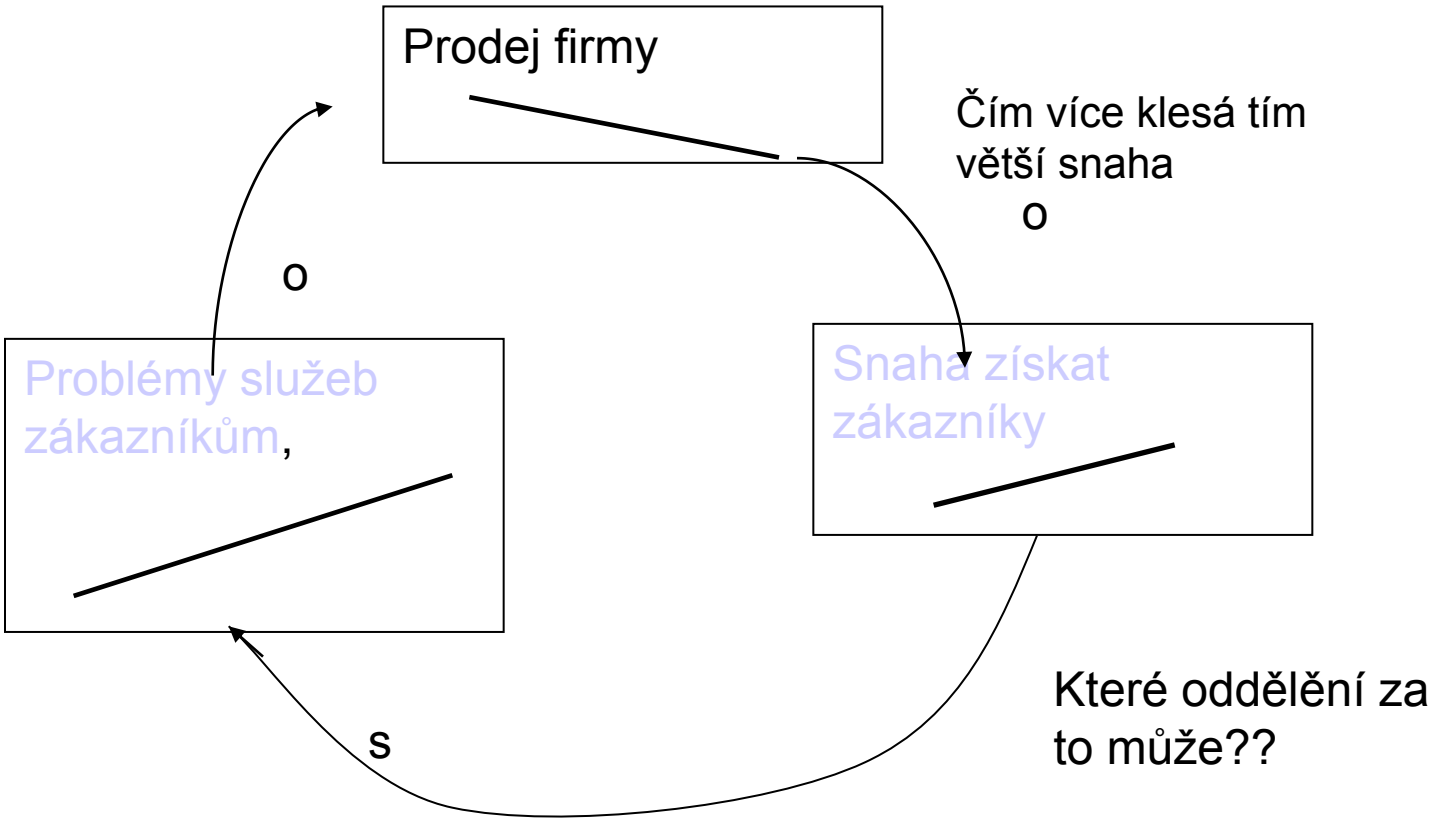
Drill down, příčina neúspěchu je jednoduchá resp. snadno lokalizovaná



Drill around, příčina neúspěchu je komplexní



Drill around, příčina neúspěchu je komplexní

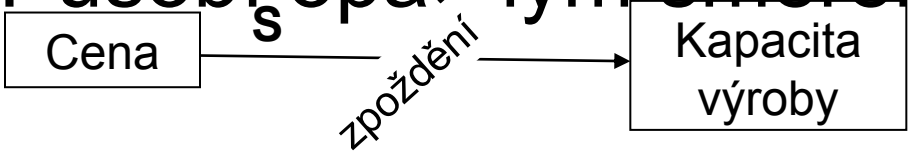


Archetypy

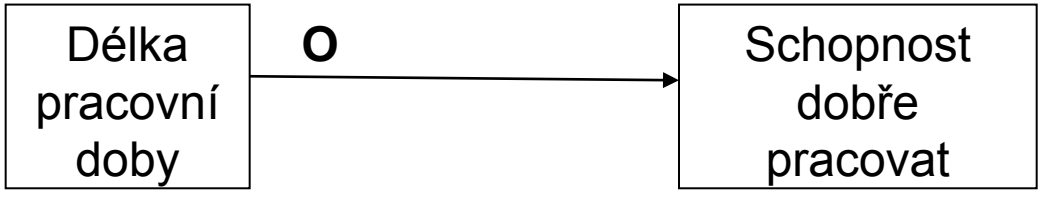
- Kauzální diagramy (KD) jsou ohodnocené orientované grafy. Uzly činnost/skutečnosti, hrany říkají „má vliv na …“ a to buď souhlasný nebo opačný
- V KD je vhodné hledat podgrafy a cykly
- Některé podgrafy se často opakují, ty se nazývají archetypy, je jich dosti málo typů
- W. Chan Kim, Renée Mauborgne, **Blue Ocean Strategy: How to Create Uncontested Market Space and Make Competition Irrelevant,** Harvard Business School Press, **ISBN** 1591396190, February 03, 2005 Henry Mintzberg, Managers not MBA’s, Prentice Hall, 2004
- System thinker

Souvislosti

- Působí hned nebo se zpožděním
- Působí souhlasně,
- Působí opačným směrem

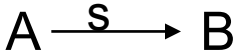


Míněno z hlediska výrobce, čím vyšší cenu si mohu dovolit, tím více investuji, potenciálně pořád

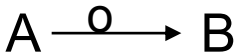


Zatížení pracovníků nelze zvyšovat neomezeně, únava se projeví s poměrně malým zpožděním

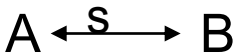
Notace kauzálních diagramů



A a B se vyvíjejí synchronně. Roste-li A má B tendenci růst, klesá-li A má B tendenci růst



A a B se vyvíjejí opačně, růst A stimuluje pokles B pokles A stimuluje růst B

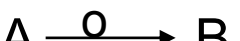


Zkratka pro

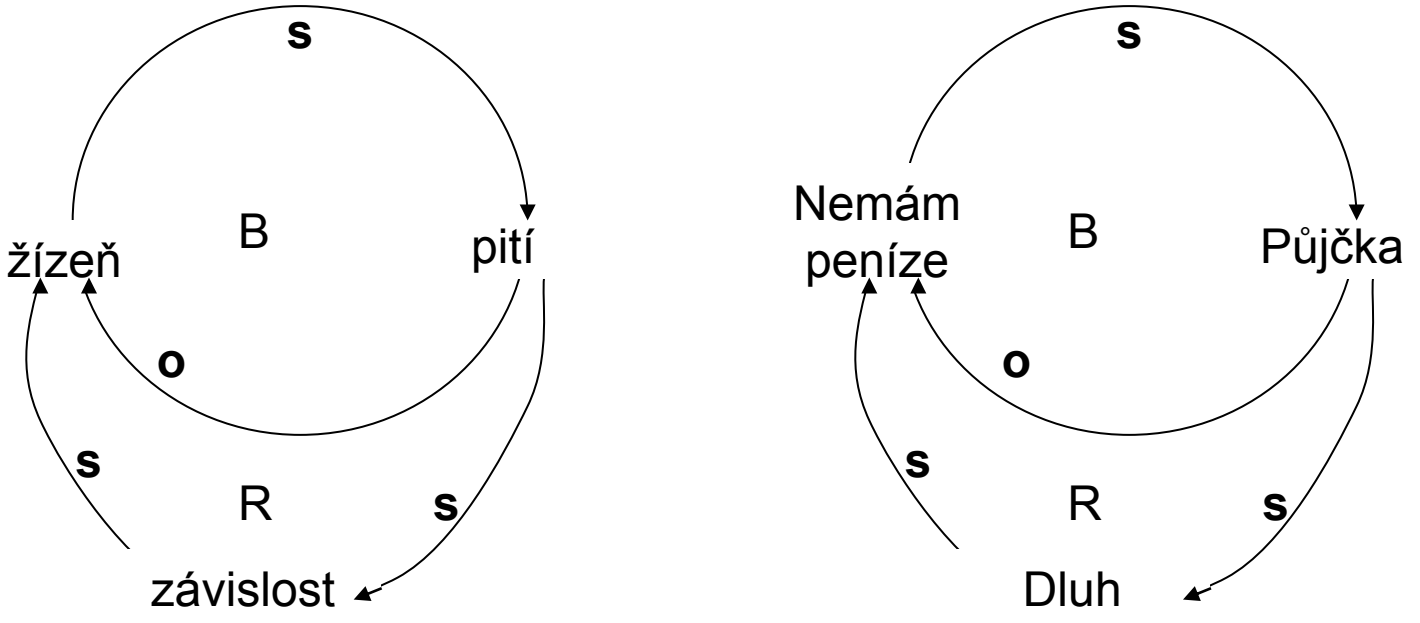
a

Zkratka pro

a



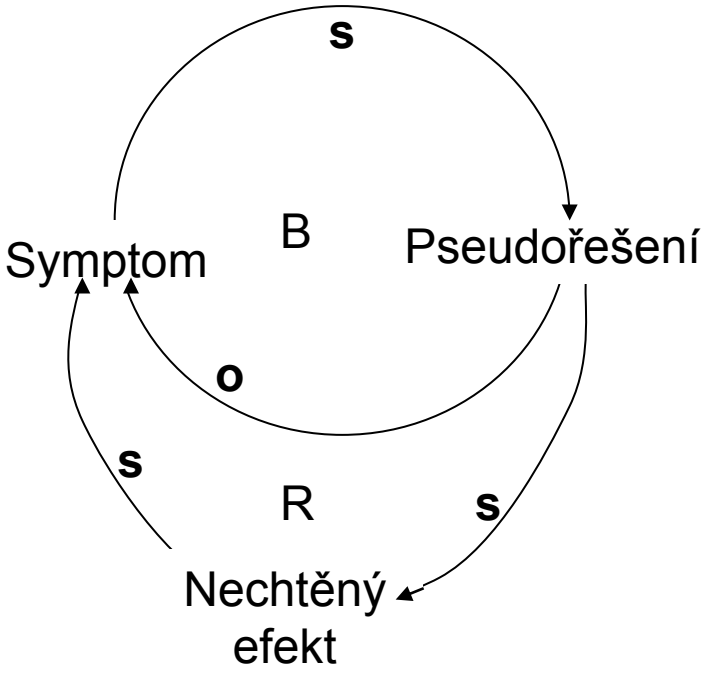
„Záplatování“

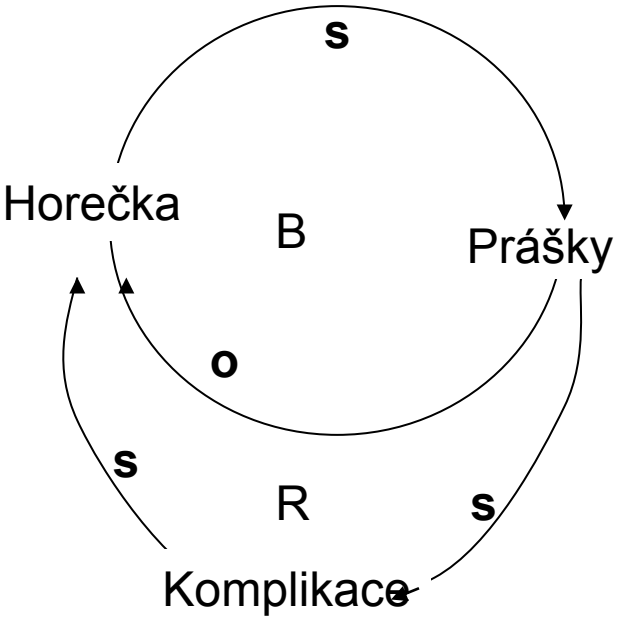


R rozvoj – nestabilita, exponenciální růst či pokles

B balancování, tendence k ustálení

„Záplatování“

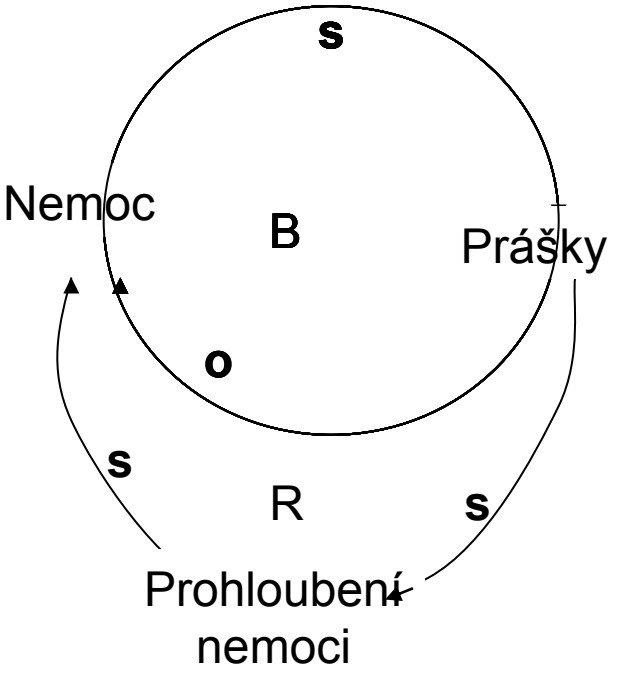




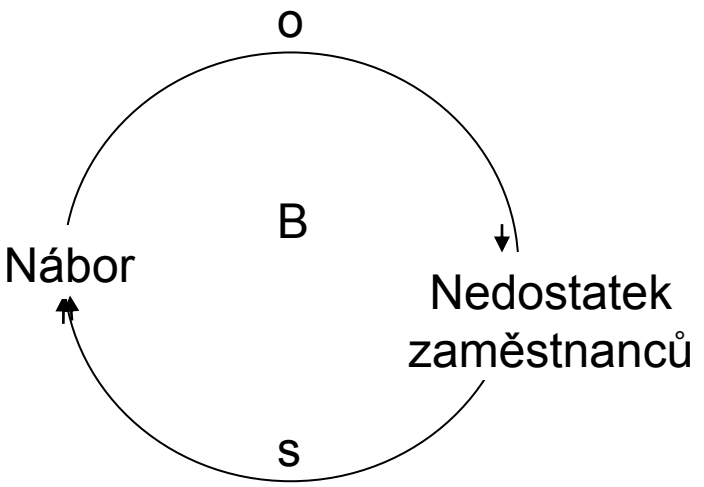
„Záplatování“

Balancující cyklus – lichý počet o (ex. Metrika, která má tendenci se ustálit)

Rozvojový cyklus – sudý počet o (ex. Metrika, která nemá tendenci se ustálit, roste nebo klesá exponenciálně)

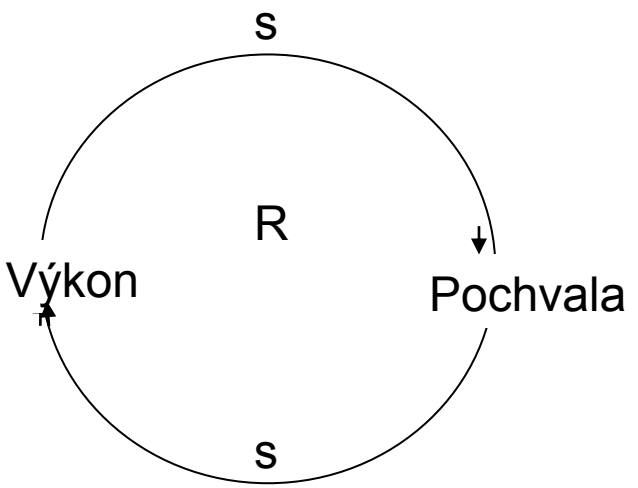


Vyvažující cyklus



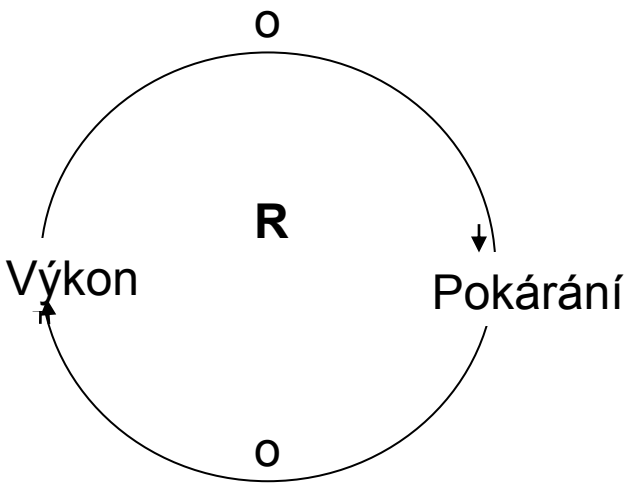
Lichý počet **o**

Rozvojový cyklus



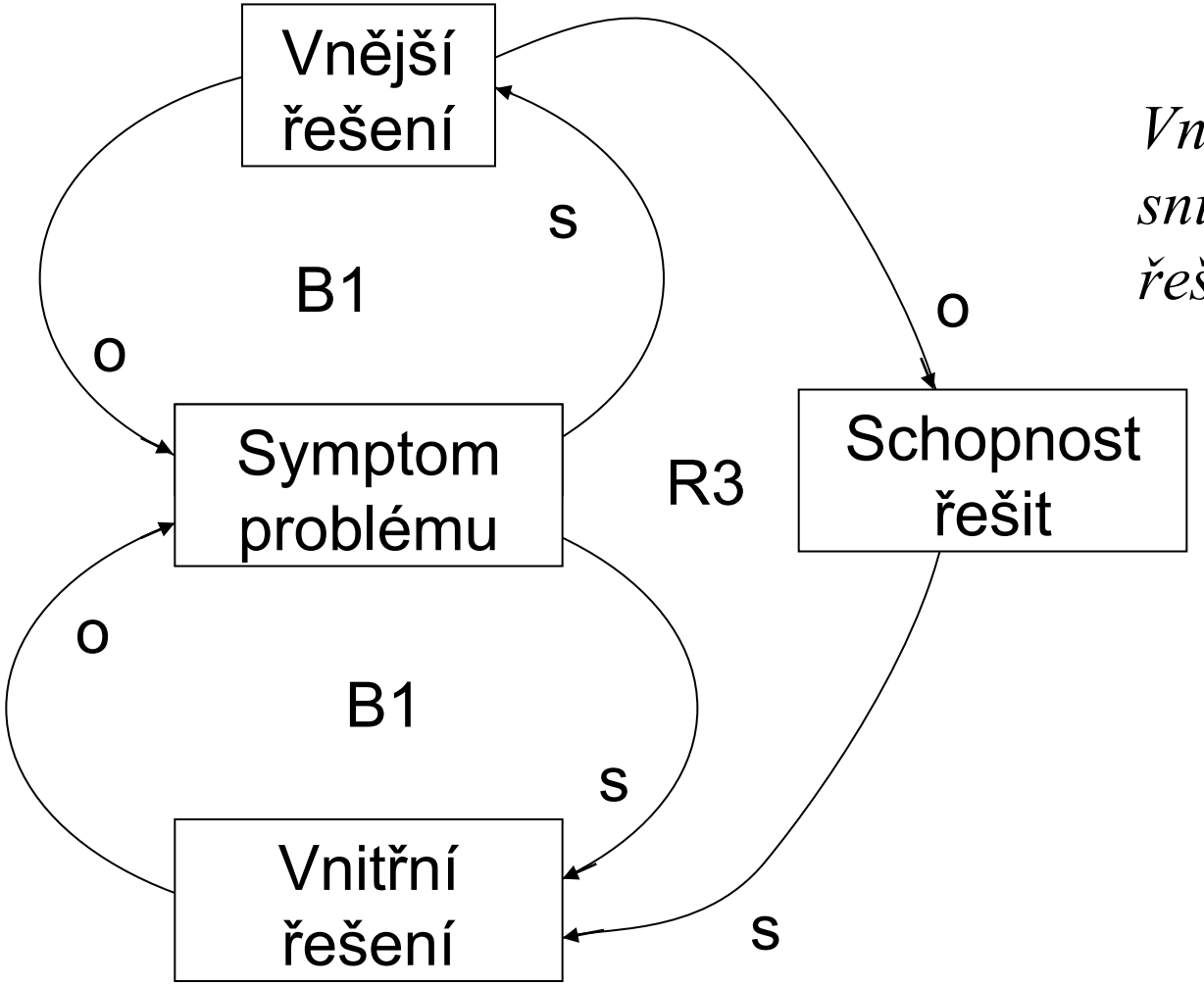
Sudý nebo nulový počet **o**

Rozvojový cyklus



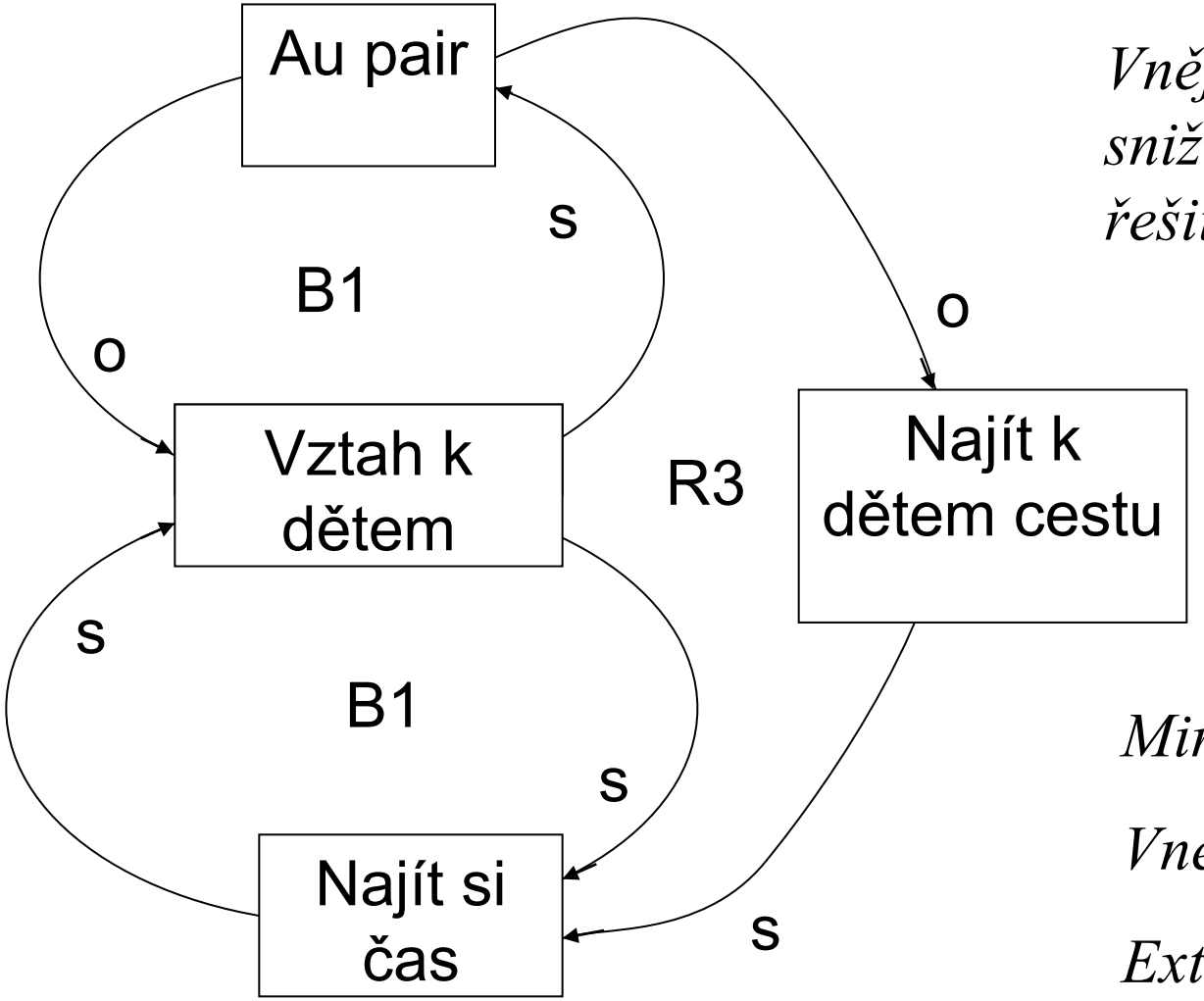
Sudý nebo nulový počet **o**

Přesun zátěže



Vnější pomoc snižuje schopnost řešit

Přesun zátěže



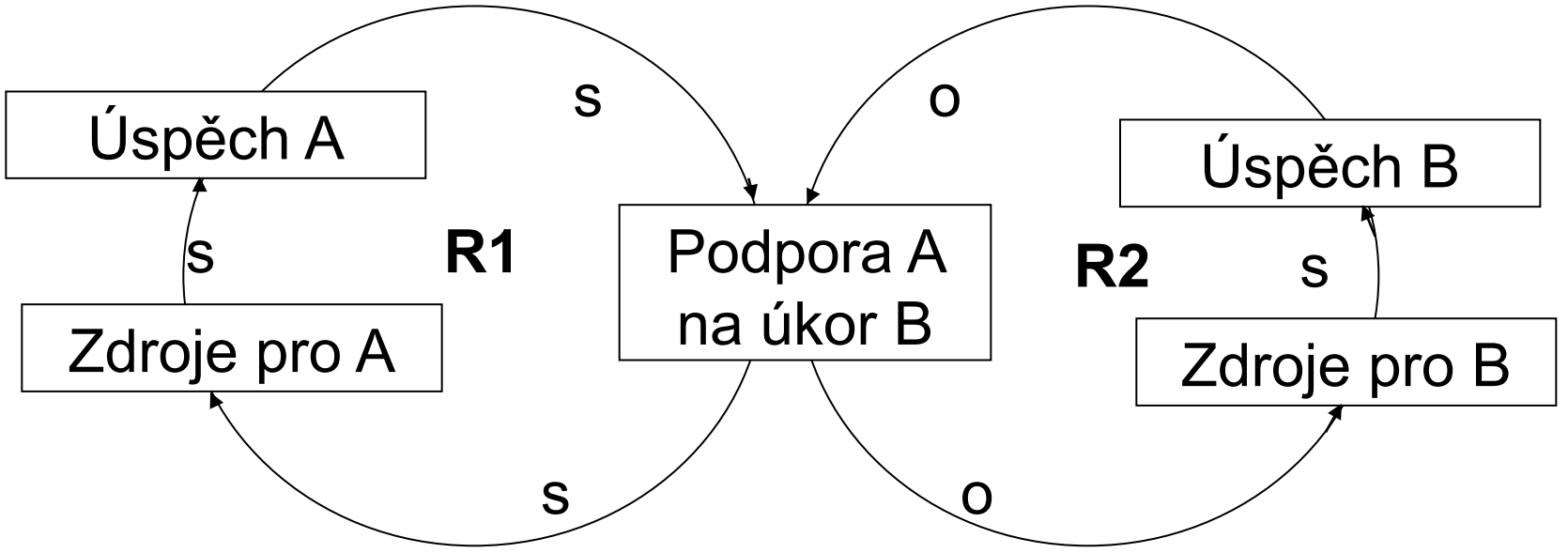
Vnější pomoc snižuje schopnost řešit

Mimořádné odměny

Vnější pomoc (ousourcing)

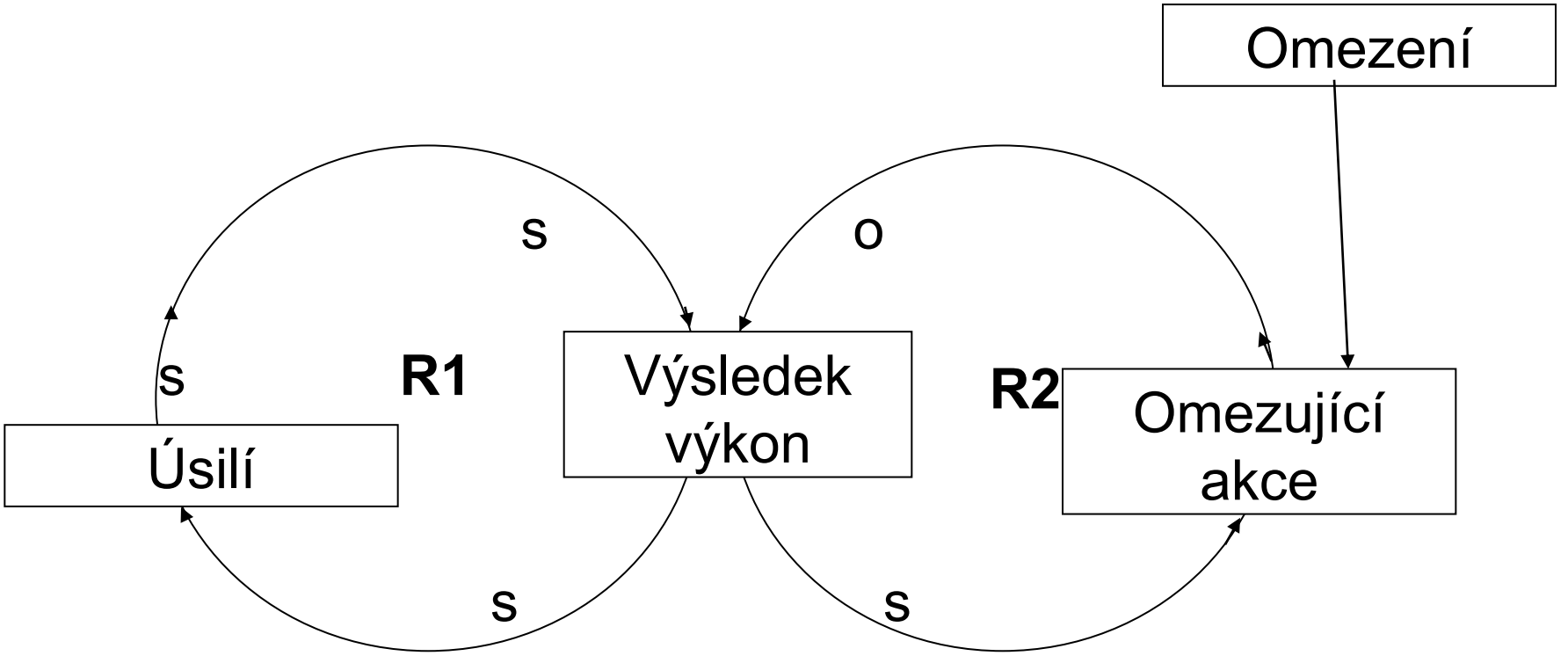
Externisté

Podpora úspěšnému



Práce kontra rodina, zavalení prací, kandidáti povýšení, prvý a druhý brankář

Hranice růstu



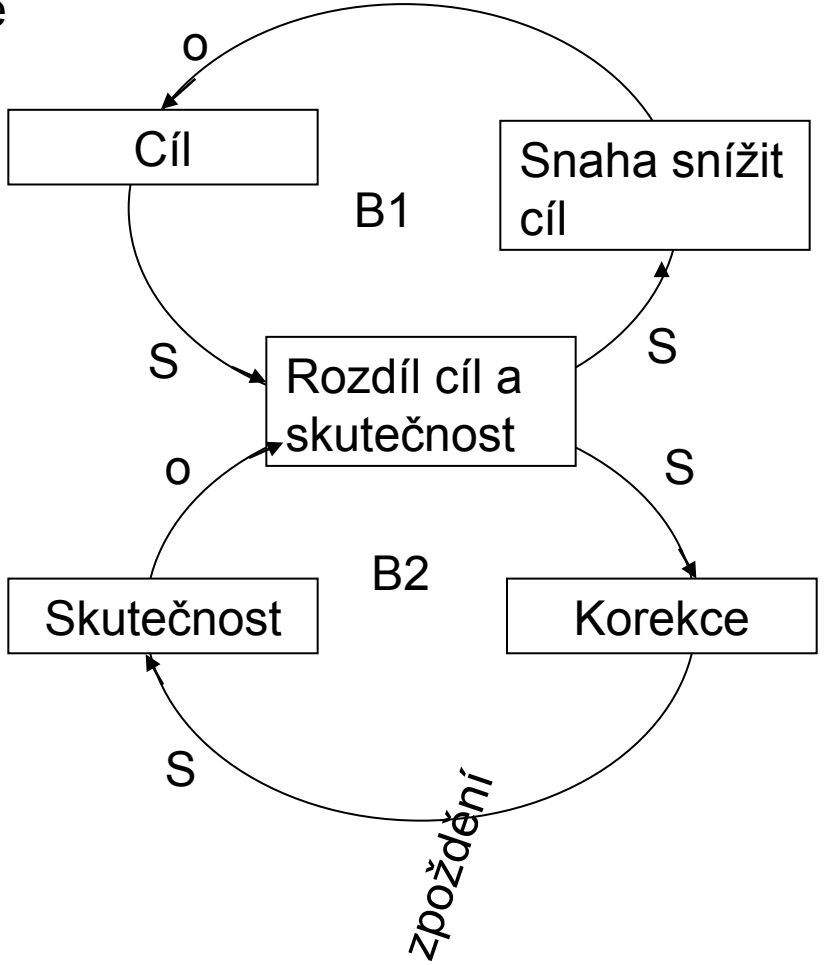
Prodej omezuje kapacita výroby, dobrý lékař získává zákazníky ale nestihne léčbu nebo sám onemocní,

počet ovcí a úživnost pastvin

Některé triky – snížení cíle

S – souhlasné

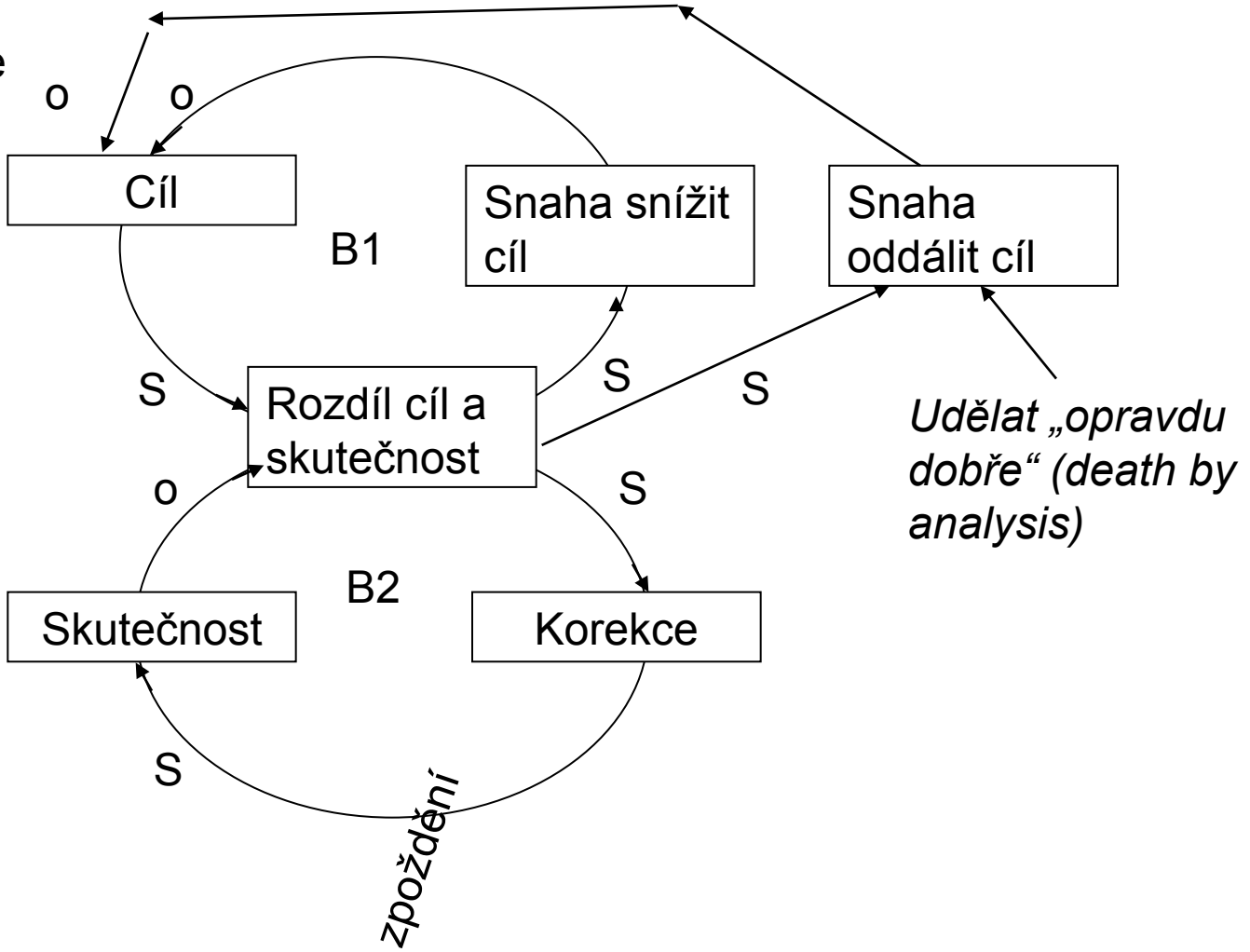
O - opačné



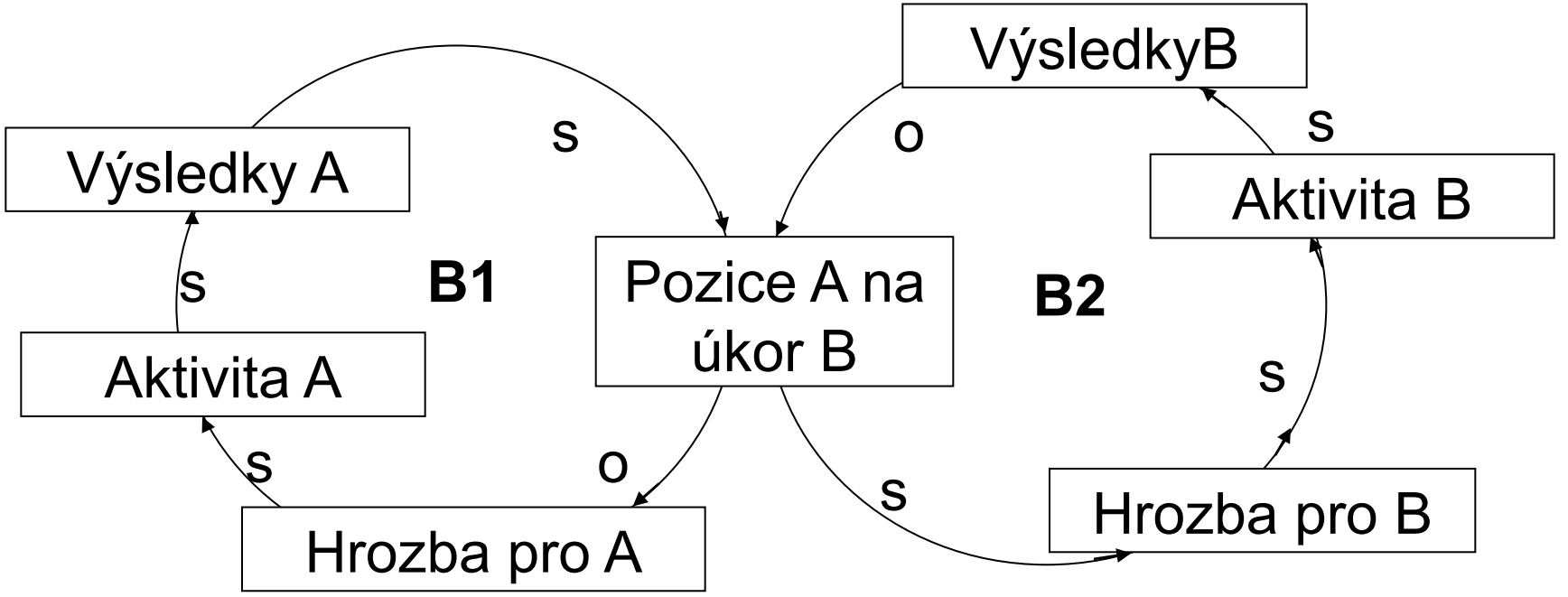
Některé triky – snížení cíle

S – souhlasné

O - opačné

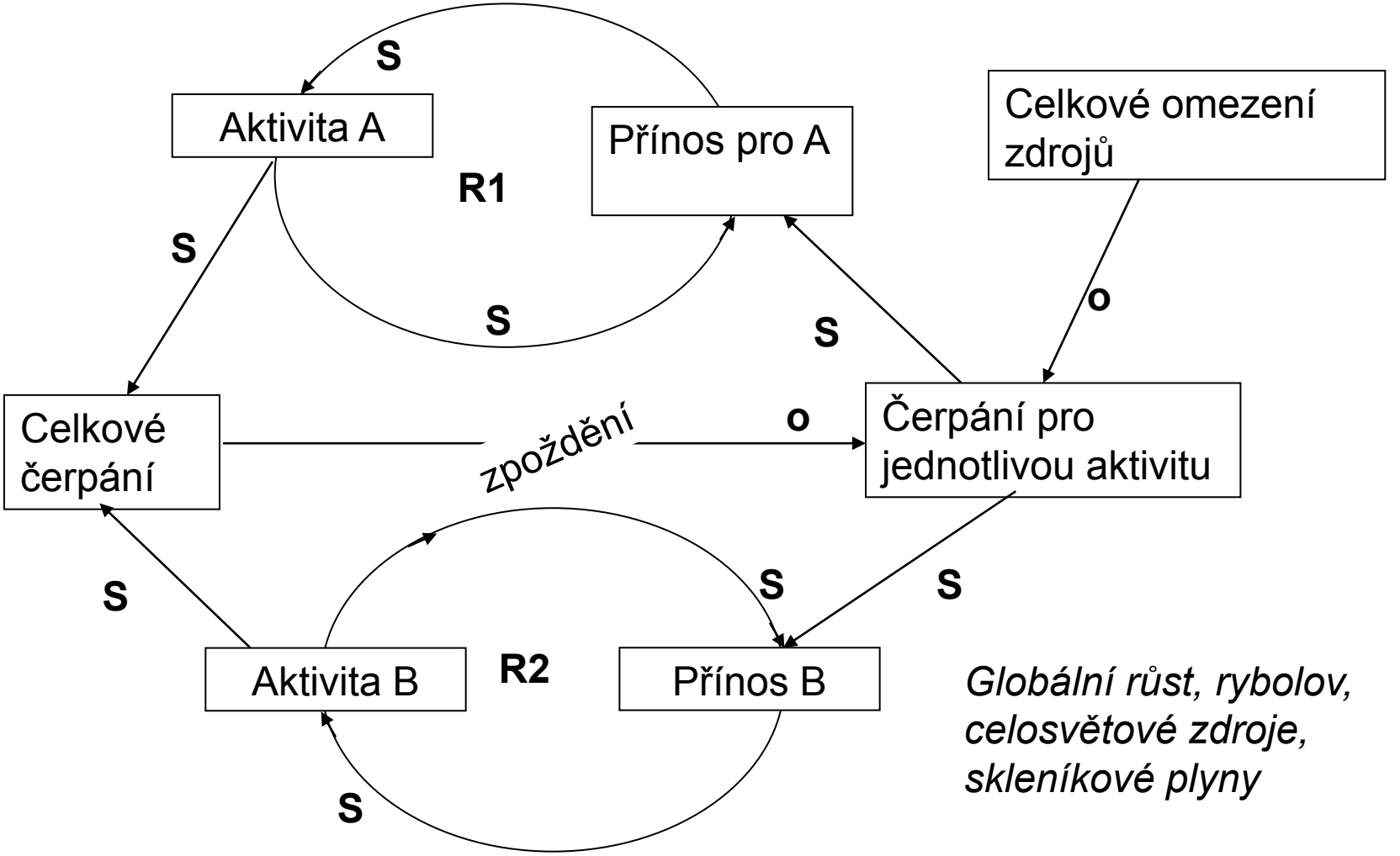


Eskalace



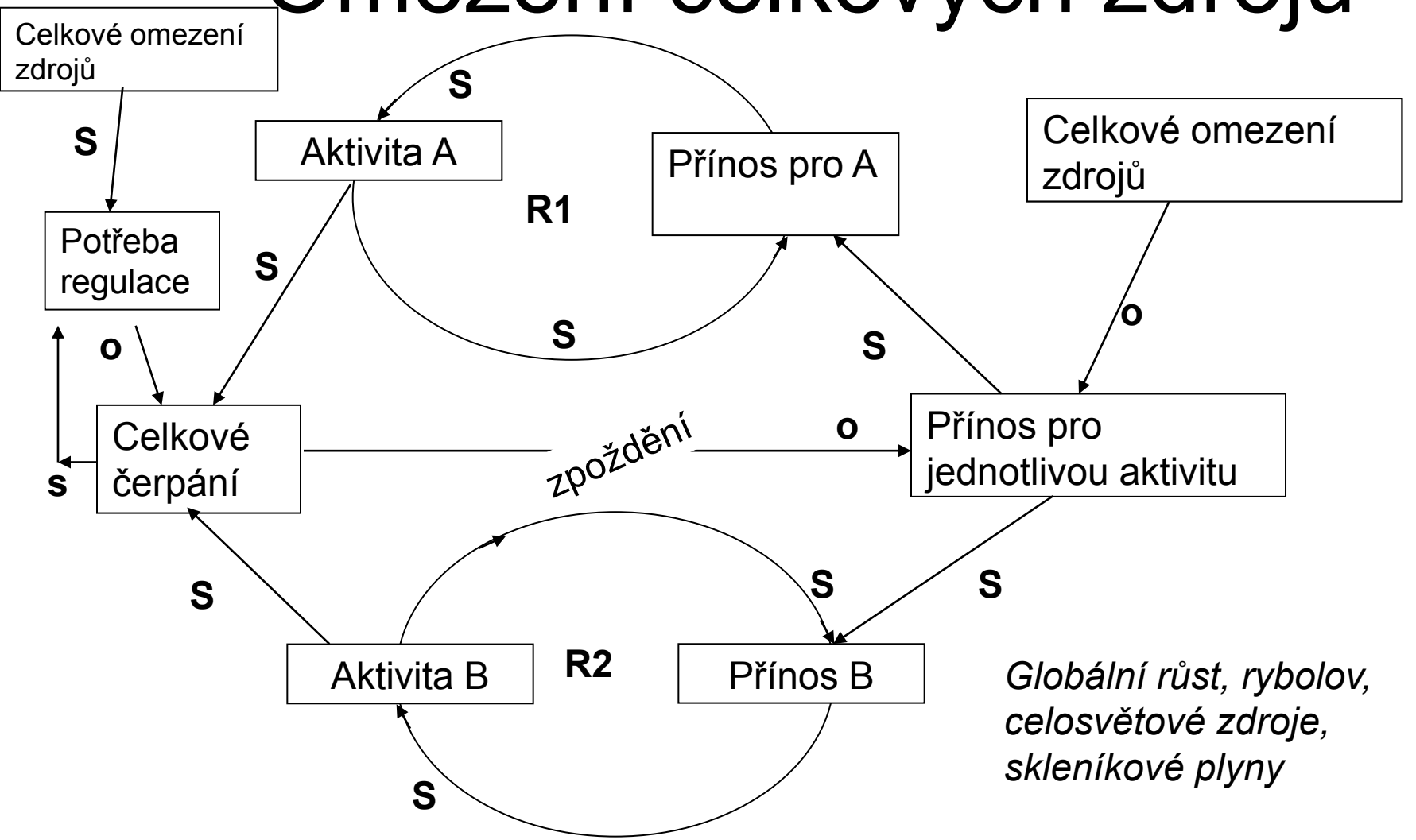
Závody ve zbrojení, cenová válka, vyhrožování mezi dětmi. *Pozor – cyklus přes všechny uzly je* ***R***

Omezení celkových zdrojů



*Známo jako **Tragédie obecních pastvin,** příklad kdy trh nefunguje a ržní chování může vést ke kolapsu zdrojů a kolapsu civilizace*

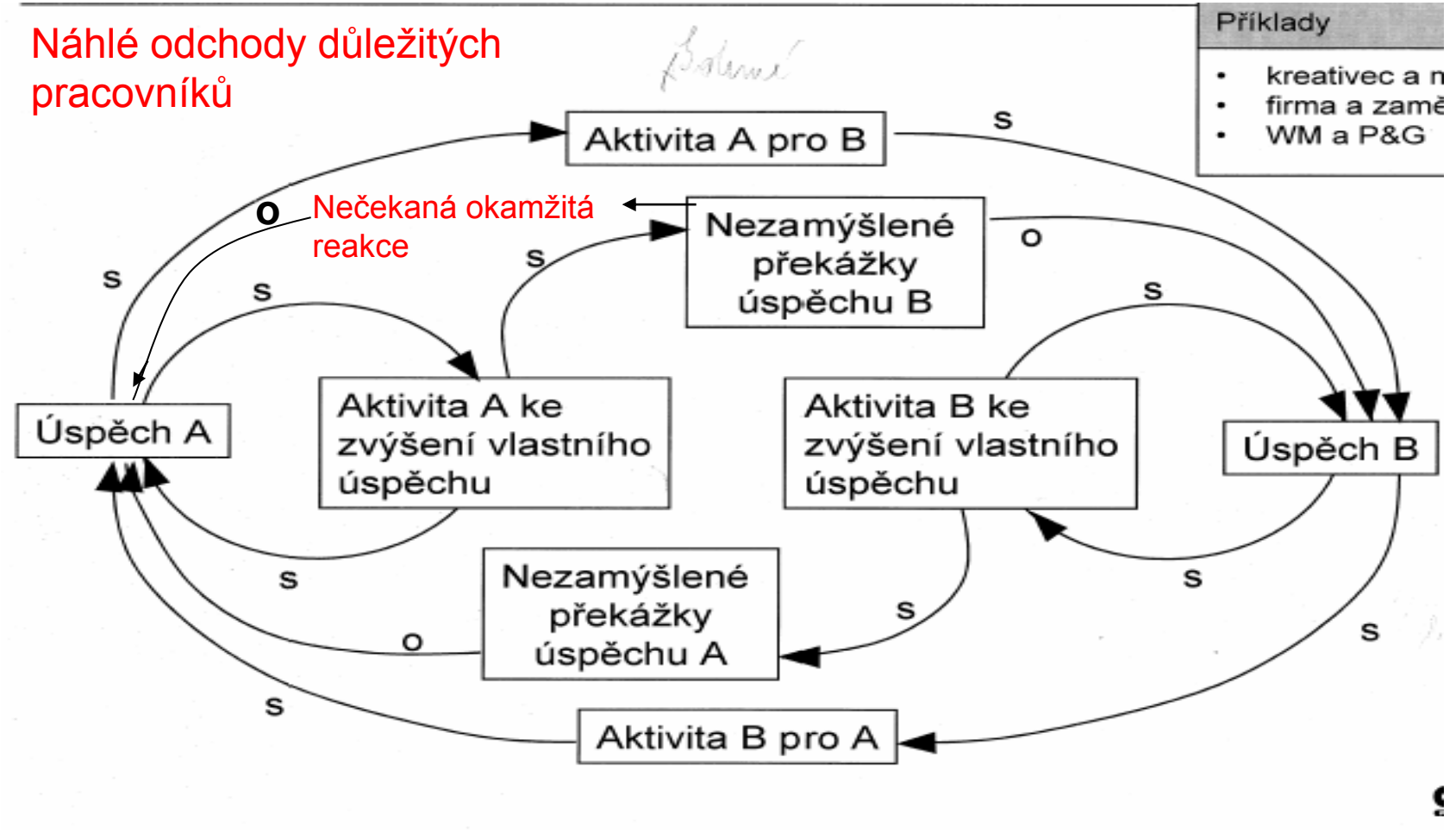
Omezení celkových zdrojů

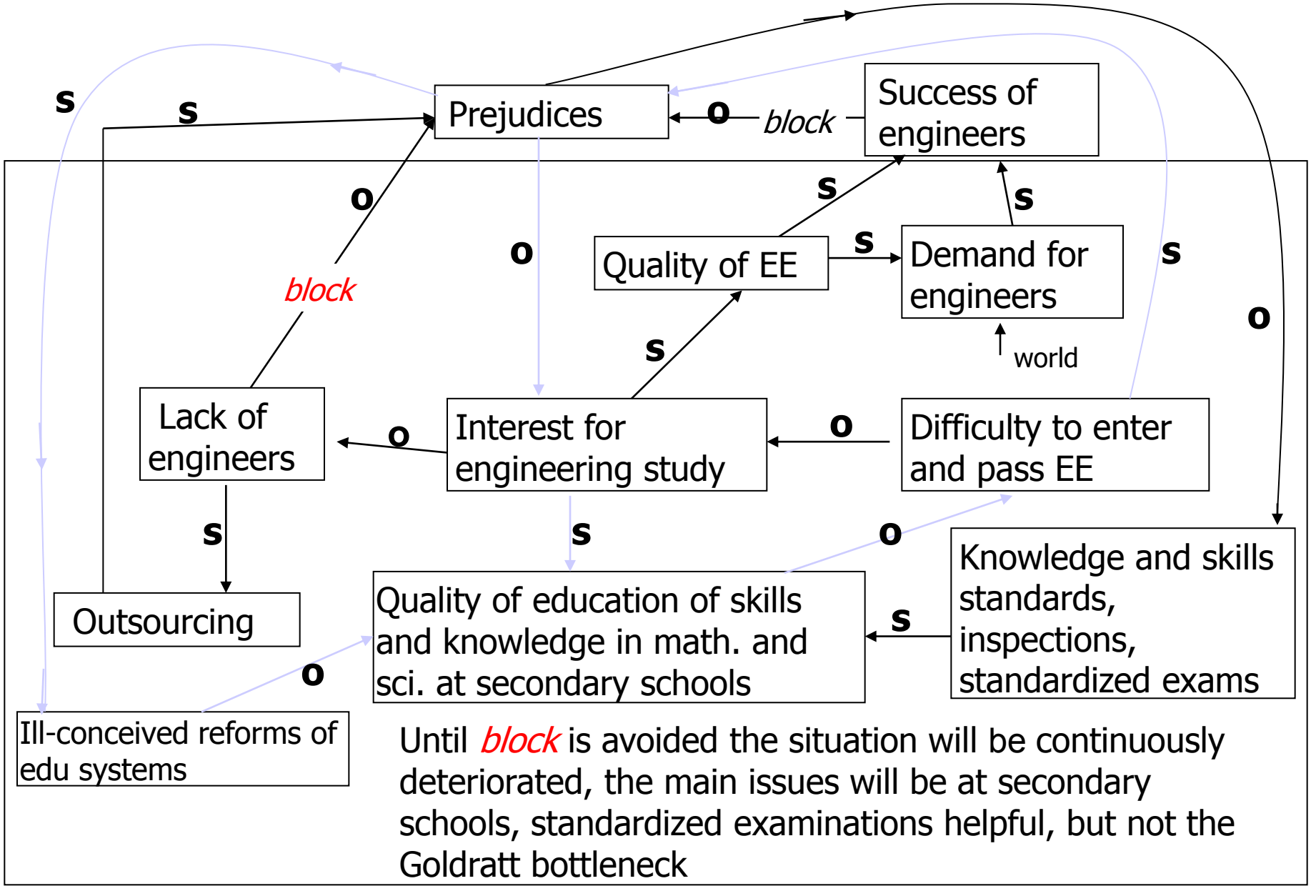


Globální růst, rybolov, celosvětové zdroje, skleníkové plyny

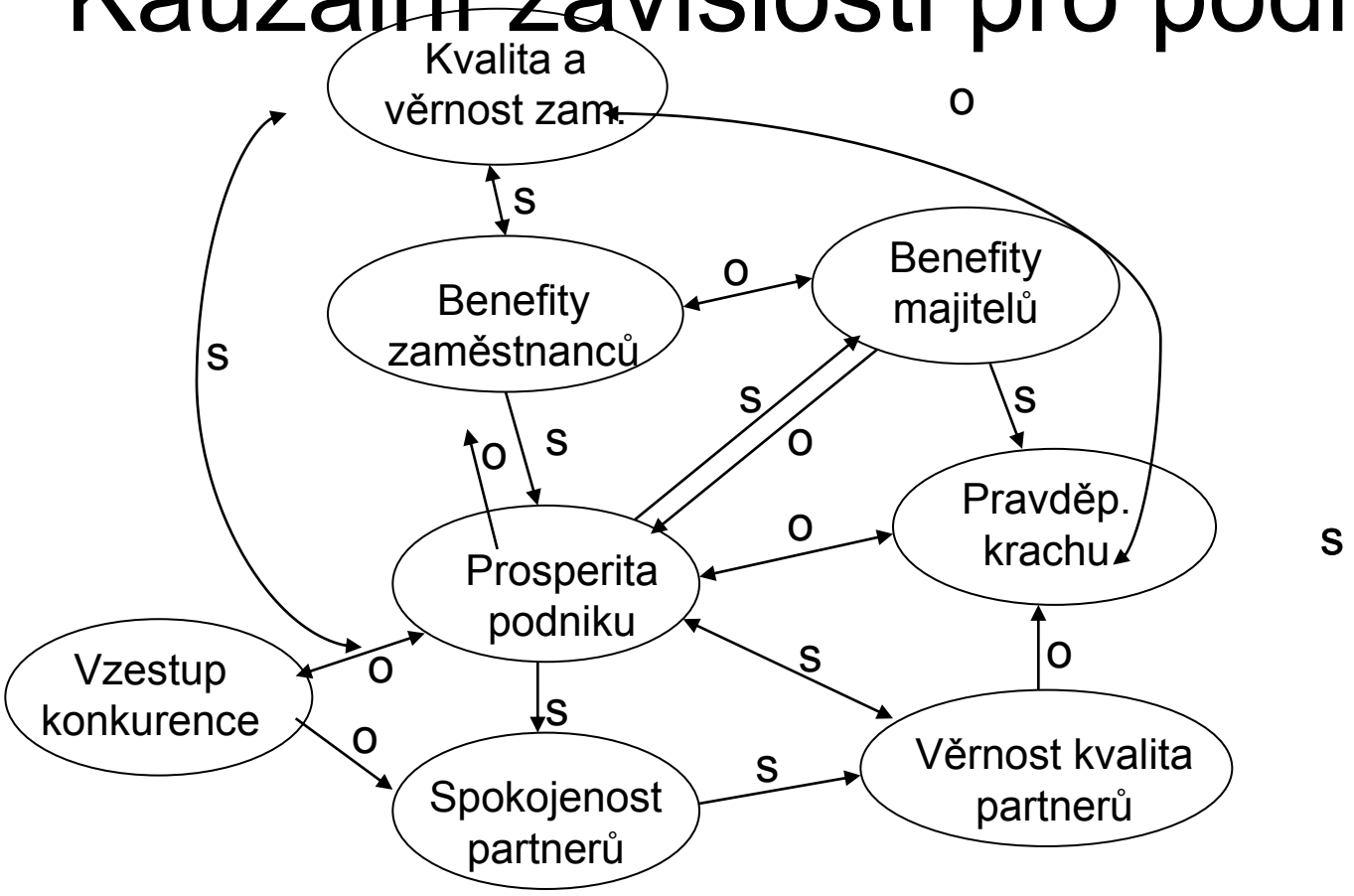
*Známo jako **Tragédie obecních pastvin,** příklad kdy trh nefunguje a ržní chování může vést ke kolapsu zdrojů a kolapsu civilizace Velikonoční ostrov*

Interakce vlastníci - zaměstnanci

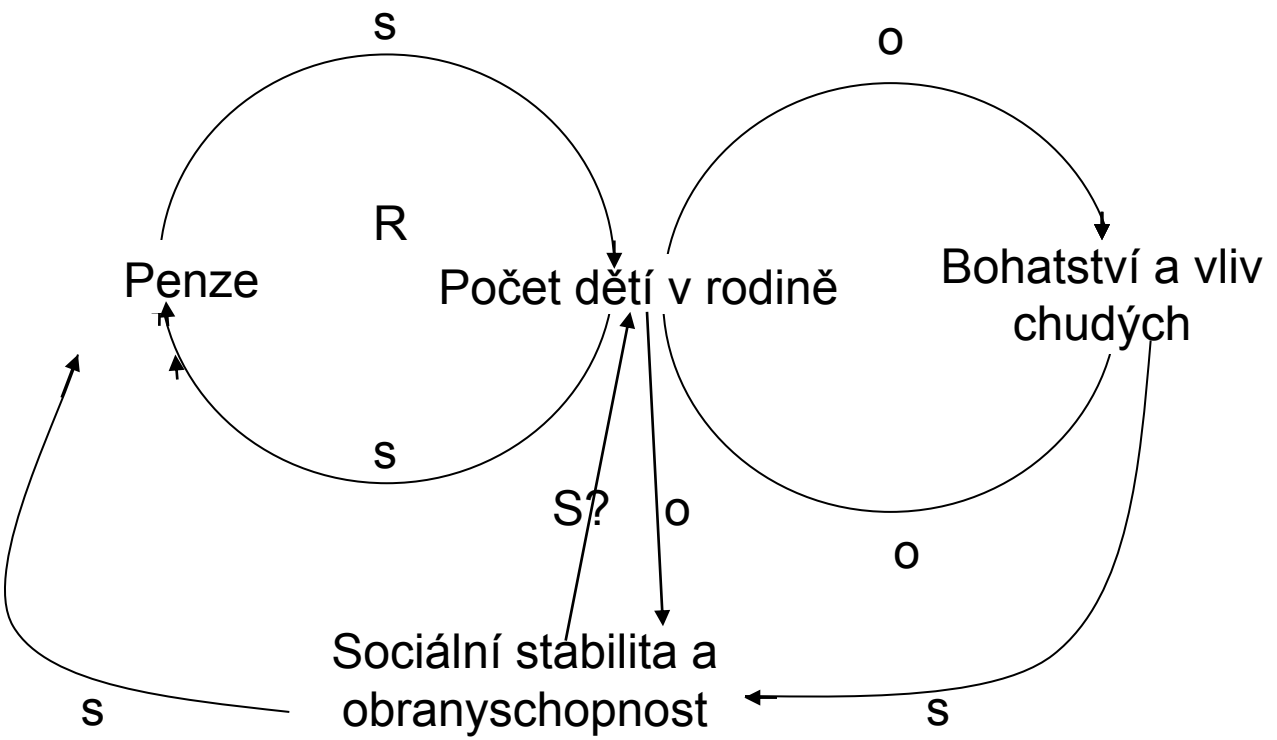



Prejudices
Success of engineers
block
Quality of EE
Demand for engineers
world
block
Lack of engineers
Interest for engineering study
Difficulty to enter and pass EE
Outsourcing
Quality of education of skills and knowledge in math. and sci. at secondary schools
Knowledge and skills standards, inspections, standardized exams
Ill-conceived reforms of edu systems
Until block is avoided the situation will be continuously deteriorated, the main issues will be at secondary schools, standardized examinations helpful, but not the Goldratt bottleneck

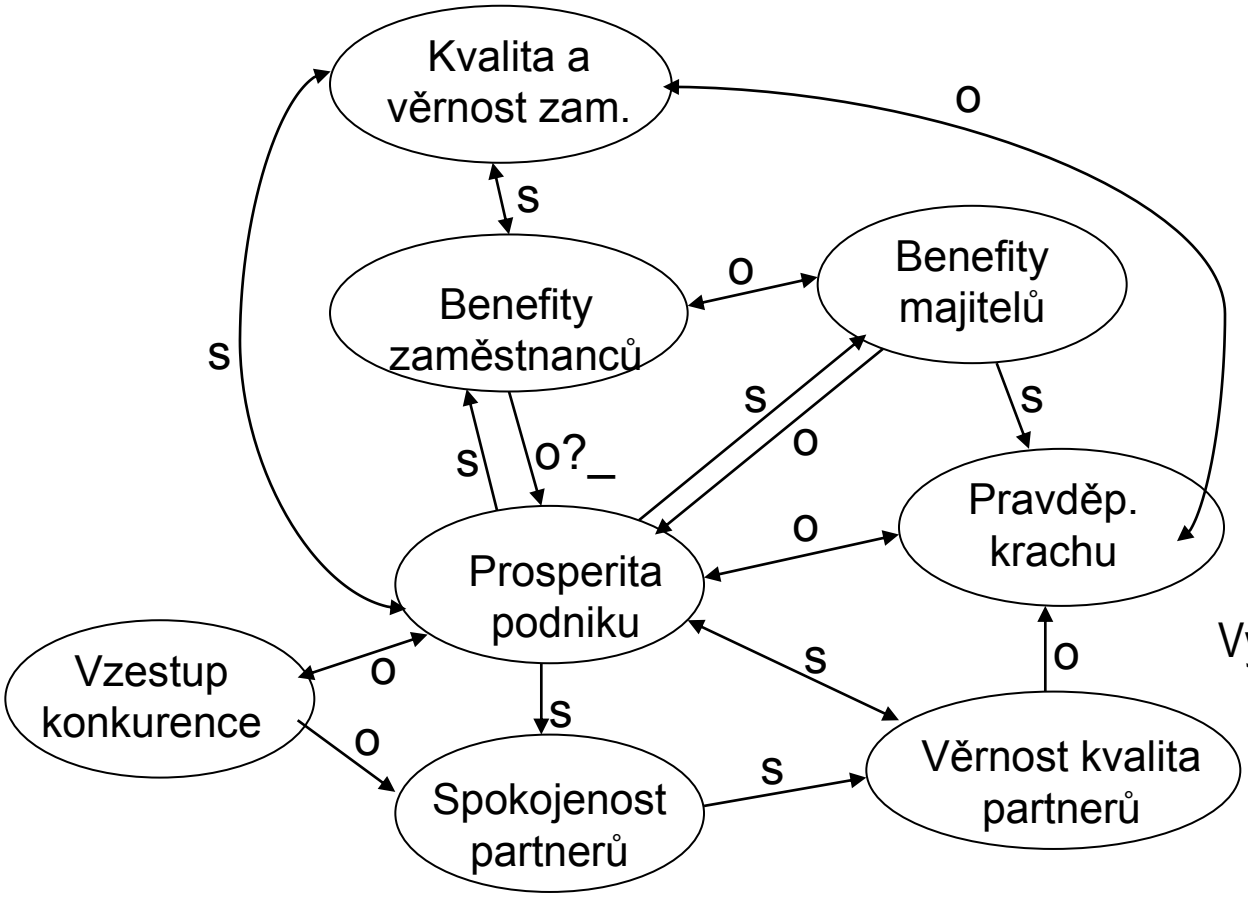
Kauzální závislosti pro podnik



Rozvojový cyklus



Kauzální závislosti pro podniky



Neinvestuje se!!!

Vývoj a dlouhodobé efekty?

Společný zájem: dlouhodobá prosperita, to by se mělo promítnout do požadavků na IS

Vybalancovat zájmy skupin v koalici, jinak podnik zkrachuje a škodní budou z dlouhodobého hlediska všichni (pokud si management moc nenakrade)

- Vedení: Udržet dobré pracovníky – dobře platit, dobré mimoplatové stimuly (benefity), ofensivní strategie na trhu, přednost v cílech má úspěch na trhu a až pak propouštění, když už to opravdu jinak nejde **Dlouhodobá prosperita** (v akciovkách obtížně dosažitelné)
- Zaměstnanci: nežádat nereálné mzdy (viz příklad Baťa za krize), zájem o profesní růst
- Snažit se vyjít vstříc obchdním partnerům případně je podpořit (Gates a Apple), platit kvalitu
- Podpora strategie firmy (to ale závisí i na SW architektuře, která musí být otevřená)

Klíčový důsledek

- Specifikace požadavků je ohrožena dvěma faktory
  - Neví se přesně co
    - Neznalost potřeb podniku, změny byznysu
    - Neznalost možností IT
    - Nejsem schopen si vzpomenout na to, co je třeba
      - Vzpomenu si až při vzniku určité situace
  - Vyvažování zájmů skupin v podniku nelze vždy předem odhadnout
    - Podpora odborného růstu, růstu mezd a zisků
    - Zlepšení postavení podniku a jeho služeb na trhu
  - Krátkozrakost některých členů koalice (fondy, především penzijní)

Koalice v profesních byrokraciích

- Profesní byrokracie – hlavní vliv mají lidé, kteří na to mají papír
  - Jsou zvoleni nebo mají profesní kvalifikaci
  - Jsou jmenováni na určitou dobu
- Kde se vyskytuje
  - Státní a místní správa
  - Zdravotní organizace
  - Školy
  - atd.

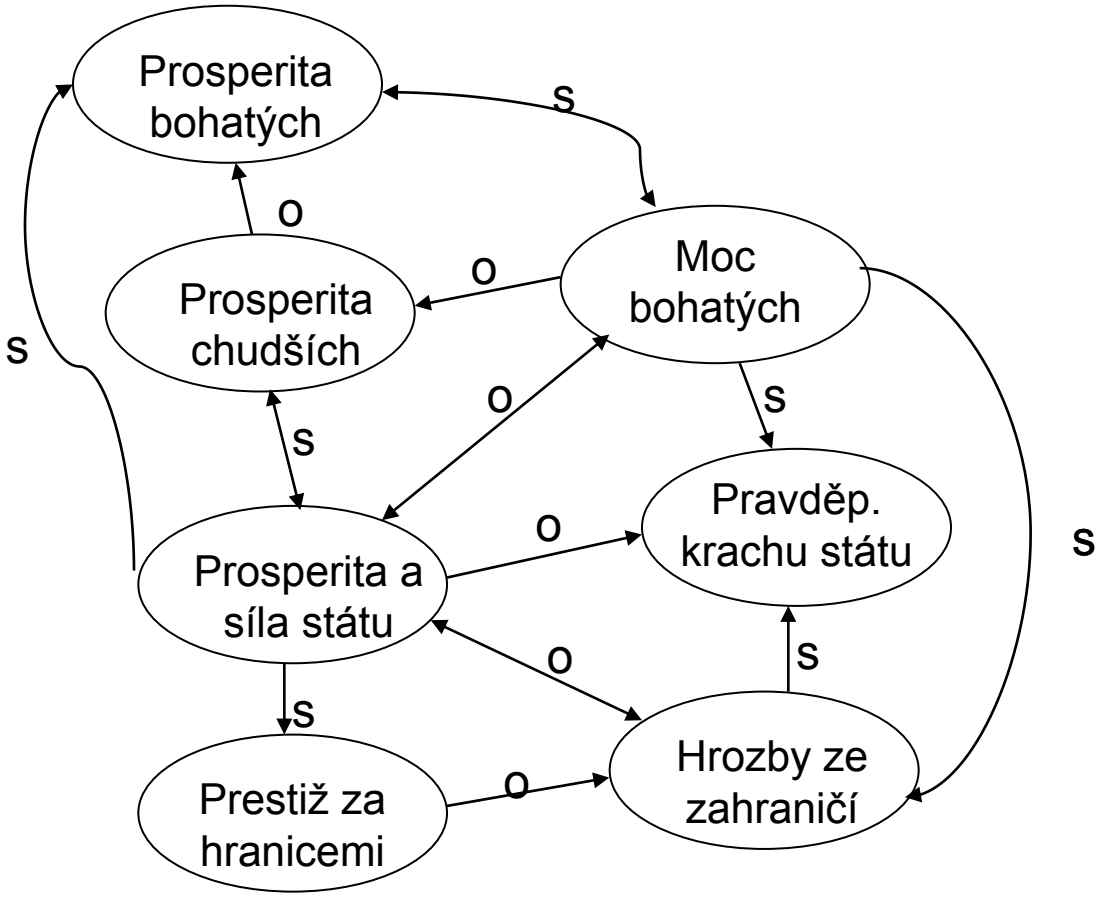
Subjekt může vystupovat v různých rolích a mít pak různé zájmy

- Daňový poplatník – co nejméně platit, co nejvíce získávat (např. bezpečnost, mír)
- Příjemce péče – co nejvíce brát (zdraví, bezpečnost, podpory)
- Úředník – co nejvíce moci a platu, co nejméně práce (ta se navíc obtížně měří)
- Politik – prosazovat zájmy svého elektorátu, udržet se u moci (i na účet fungování státu). Přímá odpovědnost politiků za svá rozhodnutí je omezená.
- *Společný zájem – prosperita státu* - fungující infrastruktura, schopnost obrany, policie, mír sociální smír, vzdělávání (Marie Terezie!, …). Společný zájem nebývá dostatečně zřejmý
  - příklady z historie, kdy nebyl o udržení centrální moci a ochrany zdrojů zájem
  - Byzanc, Mari, Řím, Mnohokrát Čína
  - Úpadek říší byl provázen nárůstem nákladů na administrativu, což úpadek zrychlovalo, administrativu ale leckdy zesložiťuje IT bez zřejmých přínosů!!!, Viz teorii od Tainter!!
  - Řešení je možné jen když existuje jistá míra solidarity všech (ochota šlechty u nás vzdát se nevolnictví pod hrozbou Pruska, snaha státu zlepšit postavení nenižších vrstev, jinak např. chybí vojáci, není dostatečně vzdělaná pracovní síla a je nízká hospodářská síla, Bismarck a Marie Terezie)

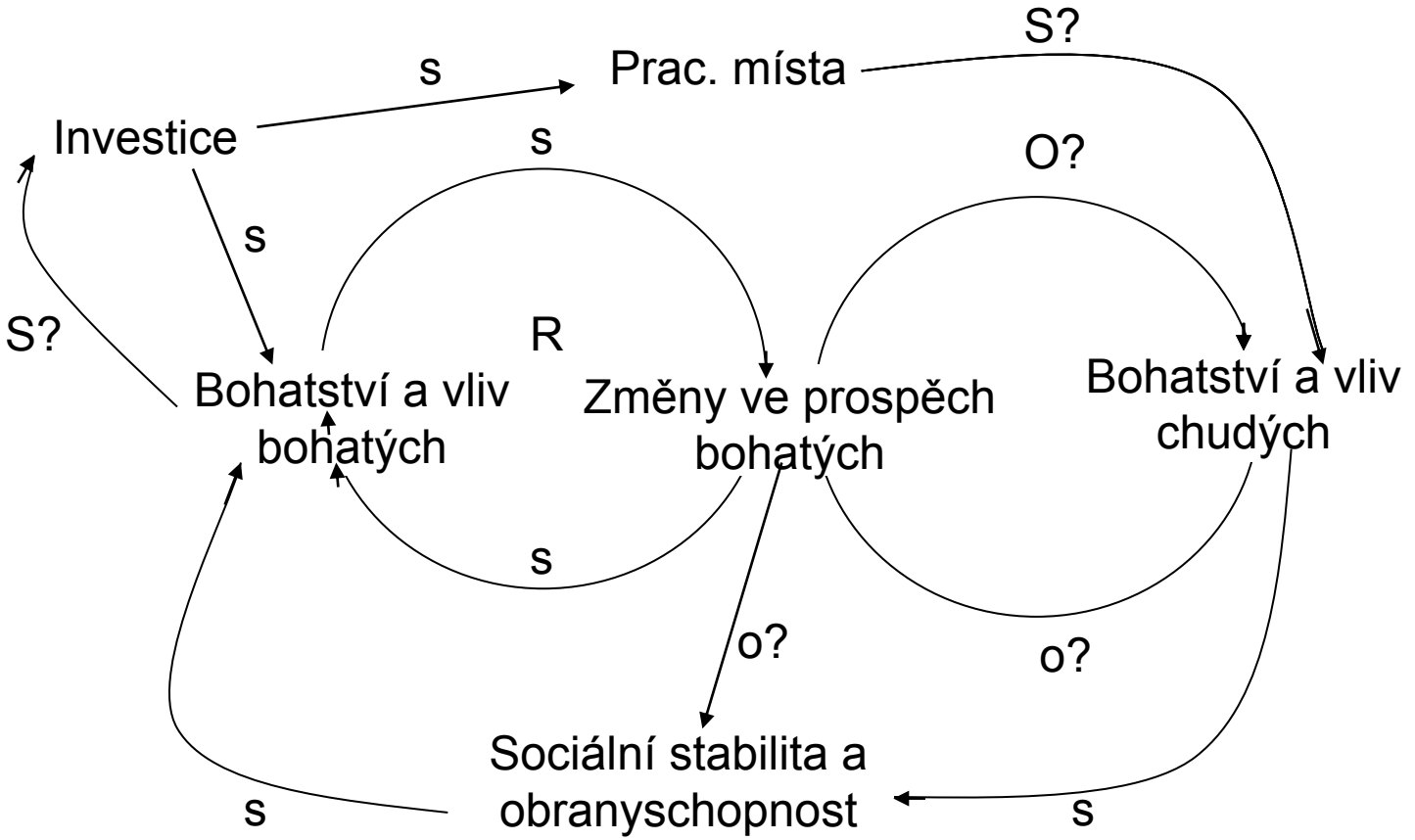
Společný zájem bývá obtížné si uvědomit a hlavně pro něj něco udělat

- Říše Mari – zotročování obyvatelstva, ztráta vojáků, zánik říše, zničil ji Chamurabi
- Čína na konci vlády dynastií (Chan, Tchang, Sun, Jüan, Ming,Čching)
  - zbídačení rolníků, růst moci a sobectví bohatých, oslabení státu a jeho vojenské síly a hospodářské síly, povstání nebo cizí vpád (od roku tisíc vládly dvě čínské a dvě cizí dynastie), ztráta bohatství bohatých, zbídačení chudých, miliony mrtvých
- Byzanc – velcí vlastníci zbavují sedláky půdy (7. století Severní Afrika, 12.století Anatolie a Řecko)
  - Tím likvidují zdroj vojenské síly a také životaschopnost říše a tím vyvolávají i ztrátu vlastního bohatství a často i života (pád Byzance)

Kauzální závislosti pro staré říše



Rozvojový cyklus jeho souvislosti



Bohatí =? majitelé a management?

Chudí= řadoví pracovníci

Kvalitní IS se v profesní byrokracii buduje obtížně

- IS státu (e-government) musí být v souladu s legislativou
  - Kvalitní řešení nebývá díky nedomyšlené nebo nevyvážené legislativě možné.
  - Může to vést až k nutnosti politických akcí (kampaně na veřejnosti, veřejné diskuse před volbami i během volebního období), různé varianty lobbingu, dlouhodobě ovlivňování stran
- IS státu by měl usnadňovat práci úředníků, usnadňovat úřední procedury pro občany a umožňovat analýzu a kontrolu politických rozhodnutí (např. ve zdravotnictví nebo při školských reformách). Zvláště slabé je to s tou analýzou.
  - Obava o pozice,
  - Důsledky nevhodných rozhodnutí jsou pro zúčastnělé nepřímé.
    - Nebezpečí nekalých praktik je vyšší než v soukromé sféře
    - Menší nasazení a menší opatrnost při formulaci požadavků
  - Obava z kontroly
  - Nízká zainteresovanost na kvalitě řešení, někdy opačný zájem (další peníze na úpravy, možnost dalších úplatků)
  - Pomalé zavádění datových schránek v ČR
- Nějakým způsobem by měl podporovat odpovědnost za práci a spolupráci lidí v různých rolích.
- Podobné úkoly jsou i v jiných profesních byrokraciích mimo státní správu (školy)

Pozorování

- Nedomyšlený systém ochrany osobních dat je s velkou pravděpodobností úzkým místem (bottleneck) využívání IT v řadě oblastí, především ve státní správě a zároveň je zásadní dlouhodobou hrozbou
- To znamená, že v těchto oblastech situaci nezlepší libovolně velké investice pokud se pravidla ochrany dat nezmění.
  - Investice pouze umožní další zkomplikování administrativních procesů bez skutečných přínosů, to se podobá situaci ve starých říších na konci jejich existence
  - To ohrožuje informatiku a vlastně celou společnost

Špatně nastavená pravidla ochrany osobních dat - úzké místo IS, především veřejných informačních systémů

Brutální metody ochrany (osobních) dat mají chránit základní lidská práva

Dosahují ale opaku.
Ohrožují budoucnost IT a nejen jich

Ničení dat jako ochrana před Velkým bratrem

Prý nutné pro splnění zásad Deklarace základních lidských práv a svobod, především práva na soukromí

- Data se de facto smí bez explicitního souhlasu dotčených osob používat a shromažďovat pouze k účelům, pro které byla pořízena a to jen pověřenými institucemi
- Každá data nevyhovující této podmínce musí být zničena
- To nazveme **brutální proces ochrany dat (BPOD)**

Výchozí mlčky činěné chybné předpoklady, vlastně předsudky

1. BPOD jsou v plném souhlasu s Deklarací základních lidských práv a jsou jejím důsledkem
2. BPOD umožňují efektivně chránit osobní data,
   - podstatně omezí počet případů, kdy mohou moje osobní data uniknout
3. BPOD nemají zásadní negativní sociální, celospolečenské a ekonomické efekty a nemají ani podstané negativní dopady na informatiku
   – Předpokládá se tedy, že škody, ke kterým by došlo kompromitováním osobních dat pokud by se BPOD nepoužívala, jsou podstatně závažnější než důsledky nedostupnosti *zveřejnitelných* informací vypočitatelných z osobních dat

Žádný z těchto předpokladů neplatí!!

Brutální procesy ochrany dat nezlepšují podstatně ochranu dat

- Pro každého je důležité, aby jeho osobní ata nepřišla (neunikala) do nežádoucích rukou
  - jako osobě je mi jedno jakým způsobem a za jakým účelem.
- *Existuje ale mnoho kanálů úniku osobních dat a to BPOD nezmění!!*
- Některé existují ze zákona!!!!

Kanály úniků dat, některé je obtížné jiné nemožné uzavřít

- Mnohé údaje jsou veřejné ze zákona (obchodní rejstříky, registry nemovitostí, ..) a mnohé se z nich dá zjistit, jiné nejsou dostatečně zabezpečeny
- Některá data pacienta jsou např. pro léčbu natolik potřebná, že lékař považuje za správné je i přes zákazy využívat (jinak poruší Hippokratovu přísahu, de facto i zákon)
  - To oslabuje celý systém ochrany dat (legislativní disciplinu)
  - Ukazuje to, že není vše v pořádku

Kanály úniků dat, některé je obtížné jiné nemožné uzavřít

- Registry a rejstříky
  - (katastrální, obchodní, občanů,spolků, …)
- Mobilní telefony
- Webové služby
- Sociální software a sítě
- Serverové stanice, cloudy (DATA JSOU LECKDE)
- Finanční instituce
- Zdravotní instituce
  - (nesmí porušit Hippokratovu přísahu)
- Obchodování na webu
  - (často partneři nejsou dostatečně profesionální a opatrní, někdy ani nemohou být)
- Atd.

BPOD ohrožuje základní lidská práva, např. právo na informace, život a dobrou zdravotní péči

- Příklad zákazu SOA systému na online monitorování výdeje léků jako prevence výroby pervitinu
  - Blokoval se nadměrný výdej léků s pseudoefedrinem jedné osobě za krátkou dobu jako prevence výroby Pervitinu
  - Výroba Pervitinu skutečně významně klesla
  - Systém byl zakázán ÚOOÚ, neboť používal zdravotní data jednotlivých osob (léky, které používají)
- Ponecháváme stranou podezření, že někteří zúčastnění s takovým výsledkem předem počítali

BPOD ohrožuje základní lidská práva, např. právo na život a na dobrou zdravotní péči

Důsledky:

- Výroba Pervitinu se po uplatnění BPOD (skartace a zákaz sběru dat o výdejích léků) zase rozjela
  - Tragédie narkomanů a jejich rodin
  - Posílení podsvětí
  - Znemožnění optimalizace spotřeby léků, kontroly kvality zdravotní péče a podpory zdravotního výzkumu

BPOD ohrožuje základní lidská práva, např. právo na život a na dobrou zdravotní péči

Důsledky 2:

Ztráta budoucích příležitostí:

- Nelze pomýšlet na on-line prevenci chybných medikací (ohrožení životů a zdraví),
  - To způsobuje ztráty životů na úrovni ztrát životů v dopravě (více než tisíc ročně),
    - v USA jsou kvalifikované odhady na úrovni cca 50000 ročně, takže u nás nějaký ten tisíc ročně, jistě existují kvalitnější odhady, základní zjištění platí a dá se použít i ve veřejných debatách.
  - Prevence chybných medikací by to mohla podstatně omezit počet vážných poškození zdraví.
    - V USA se odhaduje na cca 1,2 mil. ročně, takže u nás tak asi 50000 ročně. Počet postižených jde do statisíců

BPOD ohrožuje základní lidská práva, např. právo na život a na dobrou zdravotní péči

Důsledky 3:

Ztráta budoucích příležitostí

- Zhoršení podmínek zdravotnického výzkumu a kvality reakce na epidemie,
- Blokování optimalizace systému zdravotních pojišťoven,
- Kontrola účinků léků, optimalizace léčby.
  - Pár miliard by to hodilo.
- Objev cest šíření cholery analýzou osobních dat provedený londýnským lékařem kolem r. 1850 by dnes byl nezákonný

BPOD ohrožuje základní lidská práva, např. právo na život a na dobrou zdravotní péči

- Zákaz platí i pro využívání dat zdravotních pojišťoven akreditovanými pracovišti
  - To už je naprostá zhovadilost
- Pro státní správu má tedy de facto přednost ochrana dat před ochranou životů a zdraví
  - Pověsti,že některé instituce se k tomu oficiálně hlásí
  - Mělo by být veřejnosti známo, že hlavní efekt často je nemožnost veřejné kontroly!!!

Omezování práva na vzdělání

- Chybí nezávislý systém evaluace kvality škol a vzdělávání podle kriterií hodnotitele, např. rodiče
- Proto je obtížné vynucovat kvalitu výuky a správně volit směr studia a školu, není dohled nad efekty didaktických modernizací,
  - Stížnosti u nás i v USA (nedávno Obama)
  - Je dost indikací, že se kvalita vzdělání snižuje (STEM), ale je obtížné vyvolat změnu

Omezování práva na vzdělání

- Kriterium kvality - profesní úspěšnost a úspěšnost na vyšších stupních škol
  - Možné jen s využitím osobních dat
  - Je třeba veřejná kontrola
- Právo na vzdělání je omezeno, promrhávají se talenty – náš hlavní obnovitelný přírodní zdroj,
  - vážnější hrozba než státní dluh
- Jde o podmínku nutnou, ne postačující. Předsudek, že práce je fuj a dřina a ve škole trápení dětí (neboli dril), je krátkodobě silnější než mluva dat.
  - Dlouhodobě snad nikoliv

Jak na úředníky a zákony

Tlačit na změnu postojů veřejnosti a zákonů pomocí výše uvedených skutečností

Metody

Sami si to vyjasnit, udělat z toho politické téma,Media

Lobbing, kontakty na politiky, na strany, poslance a právníky

Budeme se zabývat především strojovou byrokracií.

V profesních byrokraciích zatím dosti tápeme.

Málo zřejmé riziko

Změna zákazníka doprovázená změnou typu jeho byrokracie je velmi významným rizikem

Sedm S, požadavky

Informační systém ve strojové byrokracii by měl zlepšovat klíčové vlastnosti organizace charakterizované termínem 7S

Platí to do značné míry i pro profesní byrokracii.

IS je v mnoha systém podporující nebo dokonce podmiňující chod jiných systémů (metasystém)

Sedm S

Informační systém by měl zlepšovat klíčové vlastnosti organizace charakterizované termínem 7S

1. Společné cíle
2. Strategie
3. Struktura a úlohy (centralizace versus decentralizace, procesy)
4. Styl managementu a podniková kultura (podpora spolupráce, identifikace s podnikem, podchycení iniciativ, využití znalostí, poskytování informací, manažerské dovednosti)
5. Systémy (kvalita, podpora spolupráce), sem patří IS, musí ale podporovat i jiné systémy
6. Spolupracovníci (kvalita, struktura pracovních týmů, spokojenost, kvalifikační růst)
7. Schopnosti a dovednosti, klíčové know-how, vývoj a výzkum, školení

Strategie versus operativní řízení

Operativa Převažuje řízení ze dne na den

- Výjimečně analýza dat, většinou jednoduché příkazy
- Velká opakovatelnost akcí, ty pracují s malými soubory dat
- Rychlá odezva, relativně rychlé provedení
- Akce mohou být i kritické (okamžité škody)

Taktika Řízení s výhledem měsíců až let

Mix metod operativy a podpory strategie, (běžné obchodní akce a procesy, roční plány)

Doposud v IS převažovala podpora operativy, dnes roste význam podpory managementu (taktika, strategie) a k tomu vhodných IS (manažerských IS), využití BI (business inteligence)

Strategie versus operativní řízení

Strategie: Klíčová rozhodnutí na dlouhou dobu

Převažují koncepce, dlouhodobé problémy

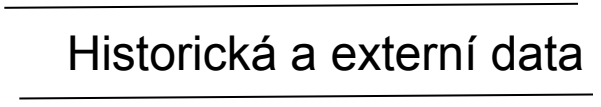
- Velký význam zkušenosti a intuice, využívání BI
- Převažuje mezi úkoly vyššího managementu
- Značné využívání externích a historických informací
- Malá opakovatelnost akcí
- Silné zastoupení analýzy dat, práce s velkými soubory dat. Data mnohdy nemusí být té nejvyšší kvality
- Obvykle pomalá odezva na provedená opatření

- *Systém by měl podporovat operativu i strategii*

Pozorování

- *IS pro operativu se liší od IS na podporu managementu i po stránce technické, v MIS (manažerský IS) je*
  - větší role datově orientovaných často statistických metod při rozhodování - trendy, efekty rozhodnutí, větší role statistiky
  - Větší otevřenost systému - externí data, analýza trhu, spolupráce s obchodními partnery
  - Je nutno pracovat s rozsáhlými daty, které mohou být různé kvality (nepřesná data operativy, např. účetnictví, nepřipouští)
- Systémy pro operativu musí spolu s vnějším světem poskytovat data managementu pro rozhodování
- Je nutné řešit propojení různých systémů. Je to otázka architektury softwaru, dnes řešeno servisní orientací, zvláště webovými službami

Toky dat nad cloudem

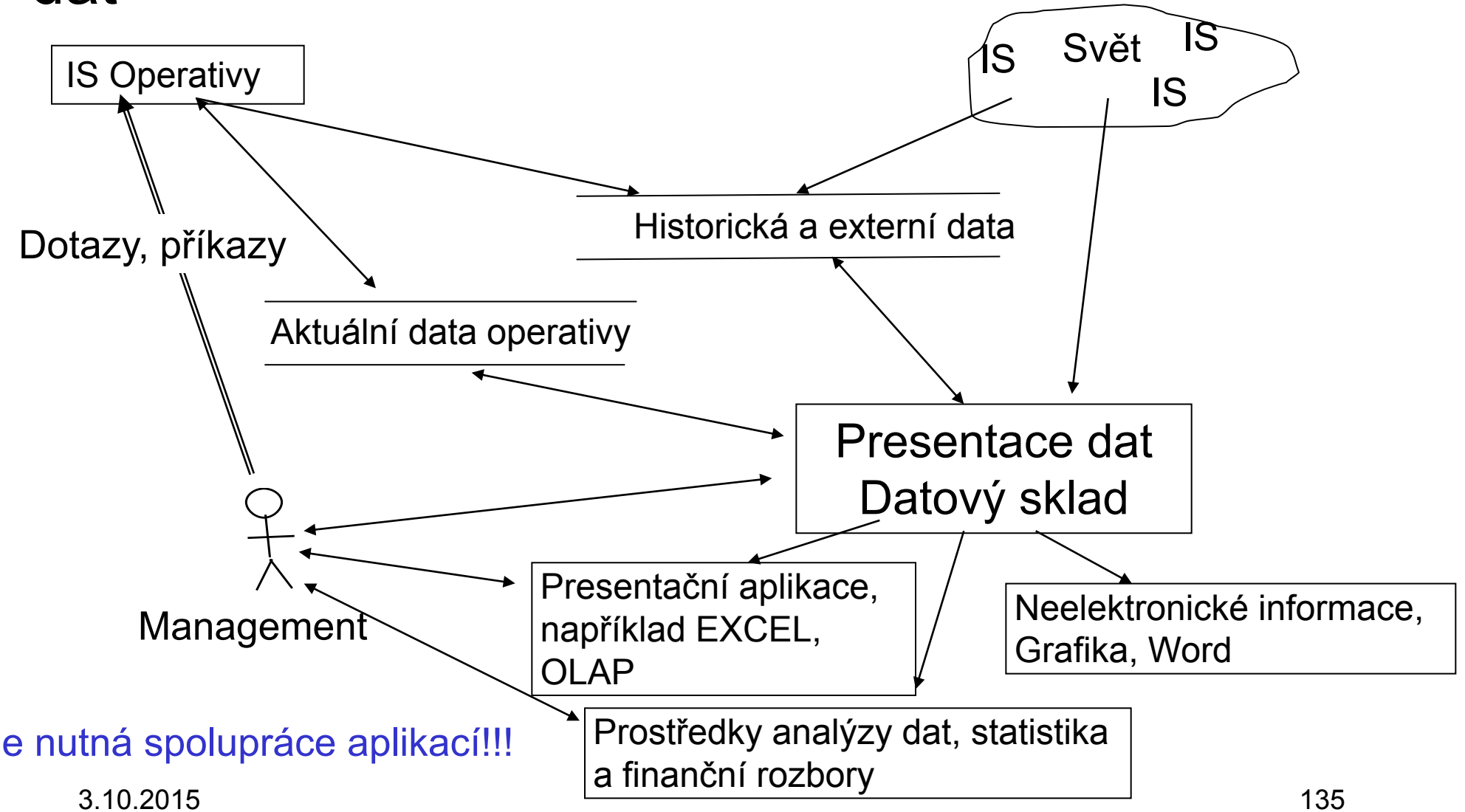


Datové úložiště

Tok dat, bulk transfer

Proces, podsystém

Manažerský IS, modifikovaný diagram toku dat



Je nutná spolupráce aplikací!!!

Pozorování

Manažerský informační systém (MIS) integruje nebo využívá funkce mnoha aplikací/systémů.

Je žádoucí, aby MIS spolupracoval s obdobnými externími systémy.

Manažeři vyžadují podporu své intuice.

Nelze od nich žádat hlubší znalosti IT (mají svých starostí dost).

- To neplatí pro funkce IS, IS hlubší a širší znalosti vývojářů potřebuje

- MIS musí obvykle používat data horší kvality
  - Nepřesná, neúplná, ne zcela důvěryhodná, nedostupná, zastaralá

Spolupráce aplikací a systémů je v praxi běžná

- Je to typické pro manažerské hry (co se stane když)
- Spolupráci aplikací usnadňují volně použitelné metaaplikace, např. mashups, služby podporující architekturu, např. adaptéry či úložiště zpráv

Podpora strategie firmy

Teorie omezení (TO) říká, že obvykle existuje jediné úzké místo. Úzké místo je nějaká charakteristika systému. Úzké místo podle TO má tu vlastnost, že pokud je nevyřešíme, nedojde i při libovolně velkých investicích ke zlepšení chování systému

(př. tunel Mrázovka neřeší úzké místo pražské dopravy, jeho otevřením se dopravní situace podstatně nezlepšila, totéž platí o pisáreckém tunelu v Brně)

Úzké místo může být stejně dobře kvalita lidí, jako kvalita výrobků, kapacita výroby, vývoj nových výrobků nebo marketing.

Určení úzkého místa je obvykle problém rozhodování s neúplnou informací, je to tedy úloha vyžadující zkušenost a intuici.

Úzké místo může být jiné pro různé úkoly pro různé doby výhledu

V české ekonomice jsou kandidáti na úzké místo pro výhled roků sociální systém, právní systém a daně, ve výhledu desetiletí demografie, především porodnost a kvalita vzdělávání

IS by měl usnadňovat hledání úzkých míst (např. kritické cesty) a pomáhat je řešit

Budeme se málo zabývat vývojem hromadně používaných systémů, jako jsou operační systémy, editory atd.

Vývoj od začátku versus nákup (customizace)

- **Vývoj od začátku „na míru“**
  - Převažují nově vyvinuté moduly
  - Často velký třesk (vše naráz)
  - Požadavky nemusí být omezovány tím, jaký SW máme k dispozici
- **Customizace - Nákup softwaru a jeho přizpůsobení konkrétním podmínkám (přesnost dat, formáty, účetní schéma) -** U velkých firem dnes customizace převažuje, trend se ale možná obrací
- SOA může kombinovat obojí

Životní cyklus softwaru, vývoj od začátku pro monolitická řešení

1. *Marketing a specifikace cílů (formulace problému)*, hledání odpovědí na otázky *proč* a rámcově *co*. Výstupem je ***vize či specifikace cílů***
2. *Specifikace požadavků*, formulace přesné odpovědi na otázku *co*, zčásti na otázku *jak*. Výstupy
   - Model systému, diagramy a (formalizované) požadavky,
     - závisí na architektuře SW modelovacích prostředcích a použitém paradigmatu (např objektovém), musí být srozumitelné oběma stranám
   - Termíny řešení, náklady, zdroje

   Oponentura: *feasibility study* (studium uskutečnitelnosti)
3. *Návrh systému.* Řešení technických otázek dekompozice, návrhy rozhraní, struktury dat a algoritmů a volba softwarových vývojových prostředků a systémového softwaru. Dostatečně podrobná dokumentace (u agilního vývoje nemusí být rozsáhlá a obvykle to platí i pro SOA). Návrh celku může dělat specialista

   Oponentura návrhu

   Případné předvedení prototypů

4. *Kódování (programování) částí*.

   Čtení kódu

5. *Testování*: částí – unit tests (testy částí, to dělají kódéři), integrační, funkcí, systému a předávací.
6. *Oživení a předání*: instalace hardwaru a základního softwaru, instalace systému zaškolení uživatelů, předávací testy, zkušební provoz. Někdy *zkušební provoz.*
7. *Provoz a údržba.*
   - odstraňování chyb zjištěných za provozu,
   - přizpůsobování novému hardwaru (HW) a přizpůsobování změnám v použitém základním (systémovém) softwaru (ZSW), jako jsou databázové a operační systémy, a konečně
   - vylepšování funkcí.
8. *Stažení z provozu.*

Životní cyklus softwaru, customizace

1. *Marketing a specifikace cílů (formulace problému)*, hledání odpovědí na otázky *proč* a rámcově *co. Výběr dodavatele*.
2. *Specifikace požadavků*, formulace přesné odpovědi na otázku *co*, zčásti na otázku *jak* od daného dodavatele.
   - Model systému, diagramy, závisí pravidlech dodavatele modelovacích prostředcích a použitém softwaru,
   - Výběr modulů a funkcí
   - Termíny řešení, náklady, zdroje

   Oponentura *feasibility study*
3. Customizace *systému (generace systému).* Zadávání funkcí, jejich vazeb a parametrů (např. přesnost, formáty dat). Tato etapa je poměrně pracná a náročná na práci expertů

4. *Kódování (programování) nezbytných doplňků*. Obvykle provádí dodavatel, kódování není mnohdy nutné
5. *Testování*: testy systému a předávací.
6. *Oživení a předání*: instalace hardwaru a základního softwaru, instalace systému, předávací testy, zkušební provoz.
7. *Provoz a údržba.* odstraňování chyb zjištěných za provozu, přizpůsobování novému hardwaru (HW) a změnám v použitém základním (systémovém) softwaru (ZSW), jako jsou databázové a operační systémy, a konečně úpravy systému. Většinu provádí dodavatel. Často spojeno i se zajišťováním provozu
8. *Stažení z provozu.*

Vodopád

- Právě uvedené etapy jsou v nějaké formě přítomny při vývoji každého technického výrobku
- Pokud se ale postupuje tak, že výstupy jednotlivých etap považuji v podstatě za definitivní, předávám je týmu následující etapy a dále se o ně nestaráme mluvíme o „metodě“ Vodopádu, *jak začnu padat už nemám šanci se vrátit,* skrytý častý důsledek – *vyvíjím vše naráz* (velký třesk)
- Až na speciální případy je *Vodopád* při vývoji větších SW systémů, zvláště IS velmi riskantní,
  - uvidíme později proč a jak to lze napravit (OO, SOA, Cloud, oponentury, prototypy..)

Výhody customizace

- Ověřený dodavatel, zná obor, ověřené techniky specifikací a oživování, know-how z mnoha instalací
- Dostupnost referencí,
- Alibi pro management (jinde to přece fungovalo)
- Menší nebezpečí selhání projektu a toho, že dodavatel opustí trh
- Úspory-cena (má to ale háček, viz níže), hlavní úspora je u údržby
- Velká nabídka funkcí (ale nebezpečí, že se koupí i zbytečnosti a že cena proto bude zbytečně vysoká a údržba také)
- Rychlejší realizace (ne závratně)

Osvědčuje díky malé pravděpodobnosti totálního selhání projektu, plný úspěch také nebývá častý

Nevýhody customizace

- Kupuje se vlastně něco jako konfekce
  - Může znamenat zbytečné organizační změny a tím značné zvýšení nákladů,
  - Může blokovat žádoucí organizační změny
  - Nemusí vyhovovat daným podmínkám, což může vyvolat další ztráty
  - Často se používají zastaralá řešení s dopady na funkce
  - Často se nakupují zbytečnosti – větší náklady při nákupu a při provozu, zbytečnosti mohou při provozu i překážet
  - Nedostatečná lokalizace (dnes spíše jiné než jazykové problémy, např. nedostatečná implementace legislativy a nedostatečné zohlednění místní kultury)
  - Často vhodné spíše pro velké podniky neb to vyrábí obr pro obry

Nevýhody customizace

- Ztráta vlastních znalostí a kvalifikace a nezdravá závislost na dodavateli, často nemožnost používat i svoje řešení a řešení třetích stran (řešení – servisní orientace)
- Odstraňuje spíše konkurenční nevýhodu než přináší výhodu. Vhodné spíše pro operativu.
- Zákazník je příliš závislý na dodavateli
  - To je známo jako „vendor lock-in“ antipattern

Jiné varianty vývoje

- Iterativní (postupné nabalování, např. pomocí API), typické pro metodiku Scrum
- Inkrementální (volná spolupráce postupně přidávaných autonomních komponent – služeb, obvykle asynchronní výměnou zpráv nebo přes (distribuovanou) databázi nebo cloud, cloud může mít nástroje i na ukládání procesů a SW komponent

Porovnání vlastního vývoje od počátku jednoho systému (u jednoho zákazníka) a customizace v procentech, data pro vývoj jsou 100, Údržba u customizace se provádí pro mnoho zákazníků současně!

	Vývoj		Customizace	
	Pracnost	Doba	Pracnost	Doba
Vize	5	8-10	5	8-10
Specifikace	15-25	25-40	10-15	20
Návrh/generace	15-20	Cca 20	15-20	10-15
Kódování	15-20	10	Cca 5	<5
Testování	35-40	25-30	Cca 20	Cca 10
Celkem	100	100	45-65	Cca 50
Údržba	200		cca30	
Celkem s údržbou	300		70-110	

Pozor

- Customizace ušetří při vývoji zpravidla jen max. 50% času a 50-70% nákladů (specifikace zůstává pracná). Navíc se často uplatňuje antivzor „ještě by se mohlo doplnit“ **To se často zapomíná!!!**
- Hlavní úspora je ve sdílení údržby
- Významné je menší nebezpečí krachu projektu
- Je to cosi jako konfekce, ale dá se nosit a je k dispozici brzy.
- **Odstraňuje konkurenční nevýhodu, nezajišťuje výhodu, nutí často k restrukturalizaci i když je zpravidla riskantní**
- Dlouhodobé zkušenosti dodavatele
- Výhoda nového programovacího jazyka není ve zrychlení programování, přínos jsou nástroje a nové metody a znovupoužívání a především vyhovění potřebám uživatelů. Hlavní přínos Fortranu byla přenostitelnost

Základní typy vývoje softwaru

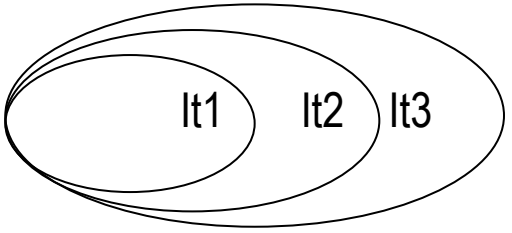
- *Software pro hromadný prodej*, bez předběžné konzultace s uživateli (operační systém, univerzální aplikace, např. MSWord)
  - Alfa testing (u vývojáře)
  - Beta testing (u vybraných uživatelů)
- *Vývoj na míru*
- *Customizace*
  - Customizovaný systém obvykle vzniká z úspěšných systémů vyvíjených na míru jejich zobecněním a doplněním nástrojů pro customizaci

Ani pouze od začátku, ani pouze customizace!

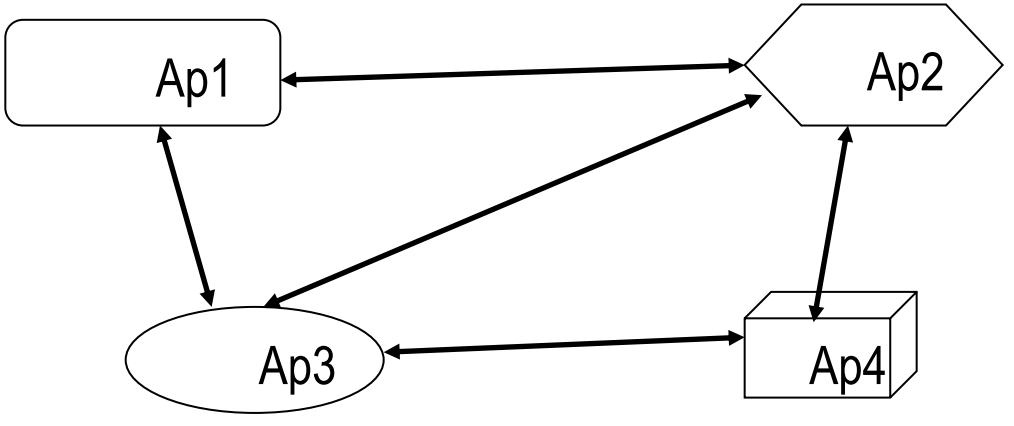
Umožnit pomocí vhodné architektury (SOA) informačního systému využití

- vlastních byznys znalostí a vlastních lidí
- existujících vlastních aplikací či celých systémů,
- nakupovaného a otevřeného SW a
- nově vyvíjených věcí

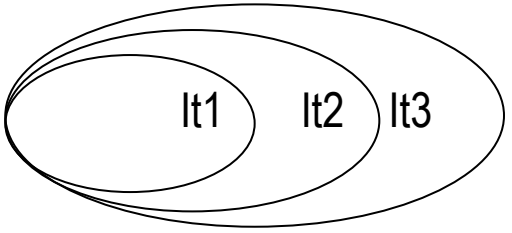
Výrobci SW z toho nemají radost,
Vymysleli SOA se spoustou služeb a tím docílili vendor lock-in a ztížili cestu k plnému využití IT u menších firem



Iterativní vývoj –jedna rostoucí aplikace (může být distribuovaná), pro vývoj iterace potřebuji mít již vyvinuté jádro a **to potřebuji při vývoji iterace používat. Typické pro agilní vývoj a scrum**

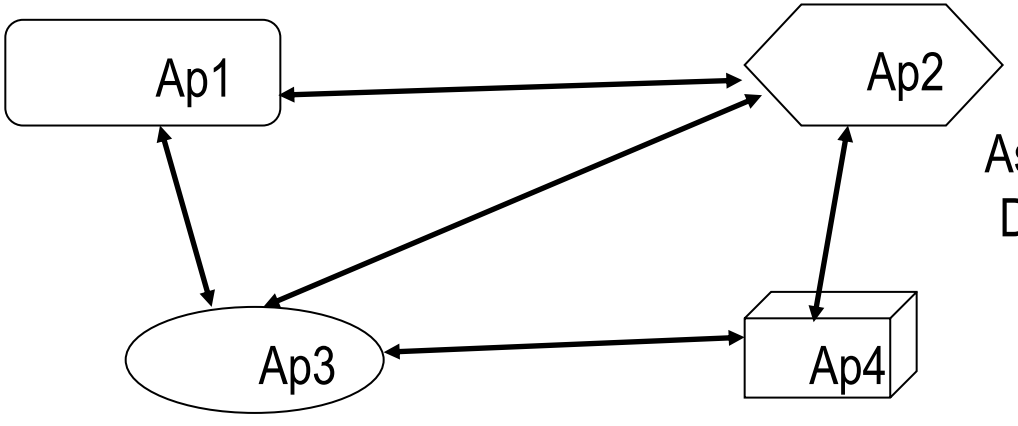


Inkrementální vývoj, propojují se (různorodé) aplikace Ap1, Ap2, .. většinou pomocí výměny zpráv nebo dat přes middleware, nebo přístupu ke společným datům (datovým úložištím). Při vývoji přírůstku **nepotřebuji mít k dispozici zbytek systému!!!, Agilita potřebuje doladit**



Obvyklé rozhraní: RPC, API

Iterativní vývoj –jedna rostoucí aplikace (může být distribuovaná), pro vývoj iterace potřebuji mít již vyvinuté jádro a **to potřebuji při vývoji iterace používat.**



Asynchronní komunikace společná DB (menší systémy) architekturní služby, kombinace s cloudy, autonomnost komponent

Inkrementální vývoj, propojují se (různorodé) aplikace Ap1, Ap2, .. většinou pomocí výměny zpráv nebo dat přes middleware, nebo přístupu ke společným datům (datovým úložištím). Při vývoji přírůstku **nepotřebuji zpravidlamít k dispozici zbytek systému!!!, Agilní vývoj potřebuje specifické nástroje, integrace spíše než přidávání**

Závislost procesu vývoje na typu řešení

- Skoro vodopád:
  - Kritická řízení technologií, kdy nelze jen připustit nedokonalé specifikace. I u technologií tendence k autonomii technologických součástek
  - Specifikace musí být téměř OK, to je potřeba při přijímání norem
- Modifikovaný vodopád:
  - Vývoj SW pro hromadné použití
    - Hry, OS, základní aplikace
      - Velké firmy nebo open source

Modifikovaný vodopád pro hromadný SW

1. Pečlivé specifikace +oponentury uživatelů
2. Návrh
3. Kódování
4. Alfa testování ve vývojovém týmu,
5. Beta testování vybranými uživateli, testuje se celková vlastnosti,
   - Vlastně pokročilé prototypování
6. Distribuce (předání),
7. Permanentní úpravy, údržba pro mnoho uživatelů současně
8. Lze realizovat i inkrementální vývoj v bodech 1. až 4., používá se ale zřídka

Kdo systém vyvíjí, případy

- Programátoři uživatele od počátku
- Programátoři dodavatele od počátku, spolupráce s pracovníky uživatele v kompetenci vedoucího projektu
- Customizátoři u customizovaných systémů
- U servisní orientace diskutované níže často programátoři (programují infrastrukturní služby) za silné spolupráce s koncovými uživateli. Totéž platí u agilních variant vývoje
- U pokročilých systémů vyvíjejí či modifikují systém i koncoví uživatelé (konkrétní tvar uživatelského rozhraní, definování procesů) – end user development , modifikace Mashup programming, extrémní-agilní programování

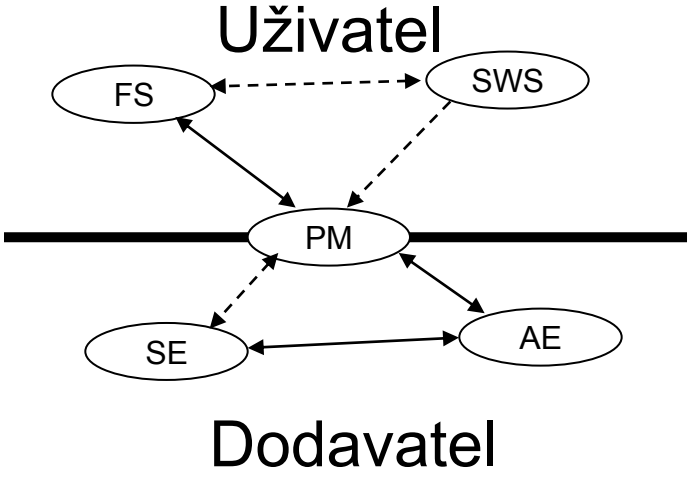
Skupiny rolí ve velkých projektech, iterativní vývoj

- *Software sponsor* (SWS), právě jeden
  - Rozhoduje o penězích, stanovuje vize
- *Function sponsor* (FS), alespoň jeden
  - Specifikuje funkce
- *Project manager* (PM), měl by mít zástupce schopného ho kdykoliv zastoupit
- *Application expert* (AE), alespoň jeden,
  - implemetuje funkce, rozumí problému
- *System expert* (SE), alespoň jeden,
  - Odpovídá za SW nástroje a vazby na systém

PM je úzké místo.

Vhodné pro velké a kritické systémy.

Hlavní výhodou je, že se všechny části systému vyvíjejí koordinovaně a pod kontrolou



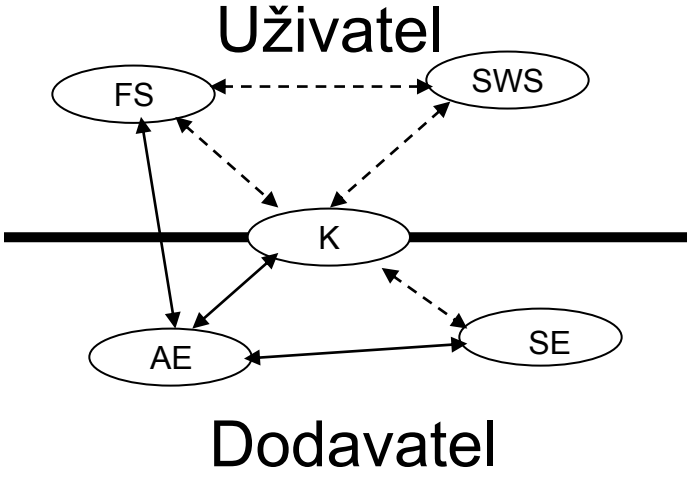
Role v nekritických projektech (Agilní vývoj, širší spolupráce!)

- *Software sponsor* (SWS), právě jeden
  - Rozhoduje o penězích, stanovuje vize
- *Function sponsor* (FS), alespoň jeden
  - Specifikuje fukce, alespoň jeden FS je stálým členem vývojového týmu
- *Kouč* (K), neformální vedoucí, napomáhá spolupráci lidí v týmu, trochu manažer
- *Programátor*(AE), alespoň dva
  - Spolu s FS vymýšlejí jak systém vylepšovat, ve dvojicích nebo malých skuppinách programátorů se do systému doplňují funkce

Úzké místo je udržení integrity a spolehlivost.

Vhodné pro spíše menší systémy a pro systémy, které nejsou kritické

Hlavní výhodou je, že se části systému vyvíjejí v úzké spolupráci se zadavateli, rychlá zpětná vazba a vysoká produktivita vývoje , potřeba vhodné architektury



Další důvody proč nedělat vodopád

Zlepšit specifikace, inkrementální vývoj, hotové zlepší specifikací toho, co se má dodělat

Nedá se stihnout včas (nedosažitelná oblast)

Snížit pracnost dělením na autonomní části

- Rozdělením se sníží pracnost a doba vývoje
- Využít hotové
- Nakoupit cizí

Zlepšit údržbu (kontinuální vývoj)

Kde je úzké místo při vývoji SW

Víme, že problém jsou specifikace,
ukažme si příklady

Procenta výskytu a cena odstranění chyb daného typu

	Četnost	Náklady
Chyby ve specifikacích	56%	82%
Chyby v návrhu	27%	13%
Chyby v kódování	7%	1%
Jiné chyby	10%	4%

Source: “An Information Systems Manifesto”, James Martin

Vlastní zkušenost přednášejícího

Hlavním zdrojem problémů jsou *nedokumentované a hlavně opomenuté předpoklady*, *nejasné požadavky* a požadavky u nichž se neví, jak se k nim došlo (nelze je vystopovat). Ty vedou i k neúplným a neadekvátním specifikacím a jsou odpovědné i za nekonzistence a ztrátu znalostí o tom, proč se věci řeší tak a ne jinak (traceablity). Tj. podobně jako v předchozím slajdu odpovídaly za 80% problémů ve specifikacích.

Shrnutí a doplnění Příklady zamlčených předpokladů

- Tunel Mrázovka ulehčí dopravě, protože auta rychleji projedou centrem a nebudou překážet (u Pisáreckého tunelu v Brně je to podobné).
  - Zamlčený předpoklad – v okolí tunelu je průjezd volný a to neplatí.
- Obchvat Praze pomůže, protože odvede transitní dopravu
  - Zamlčený předpoklad – transit sice tvoří značnou část pražské dopravy, ale lidé jezdí do centra, protože jen tak mohou rozumně dojet na pracoviště, nebo ve středu pracují a jezdí tam z pohodlnosti nebo z prestiže. Došlo ke zlepšení, nikoliv však postatnému.
- Skrytý základní předpoklad – Zvýším-li kapacitu parkovišť, věc se vyřeší.
  - To neplatí. Zahltí se ulice. Rozšířené ulice zhorší kvalitu života a opět se zahltí
  - Řešení: nepřipustit nebo omezit příjezd do centra:
    - Kodaň: Omezení parkovacích míst v centru
    - Londýn: Placené propustky

Příklad zamlčených předpokladů

Čidlo přítomnosti vlaku na dané koleji je tak jednoduché, že se nemůže porouchat

- Ale může, nebo se může změnit něco dalšího. Proto došlo k železničním nehodám.
  - Porucha sypače písku způsobila neplatnost předpokladu, že kola vagónu jsou vždy v elektrickém kontaktu s kolejí, což byla podmínka práce čidla
  - Doplnit nezávislé kontroly možných návazností událostí (vlak nemůže bez dalších signálů z koleje zmizet)

Růst ceny opravy

- Cena opravy se ztrojnásobí až zpětinásobí na každou etapu, kterou daný defekt projde aniž je detekován.
- 1,3,9,27,81
- 1,4,16,64,256
- Řešení“ provádět oponentury po každé etapě

Doporučované pro testování

1. Testy připravuje a provádí programátor, testy částí
   - Po napsání programů (kromě testů částí se to nedoporučuje)
   - Před napsáním programů (agilní vývoj - návrh testů, oponentury specifikací, formulace testovatelných výstupů)
2. Testy připravují a provádí testéři (s možnou výjimkou testů částí) a systém testují jako černou skříňku
   - Nutné a velkých a u kritických systémů
   - Pracné ale účinné, v agilních metodách vývoje realizováno jinak
   - Je žádoucí opakování testů automatizovat. Úplná automatizace testování je podle teorie složitosti (např. test na mrtvý kód, Churchova téze) nereálná
3. I varianta s testéry má dvě možnosti
   - Testování černých skříněk (pracné, účinnější), testér zná jen rozhraní, nezná kód, nemá znát ani kódéra
   - Testování bílých skříněk

Hlavní problém testů: dokazovat sám sobě nebo svému spolupracovníkovi, že jsme chybovali – proto při testování černých skříněk musí testovat testéři, kteří kód nepsali a dokonce nemají ani znát autora kódu.

Být dobrým testérem vyžaduje specifický talent.

Je důležité už ve specifikacích a především v návrhu systému zohledňovat možnosti testování (viz agilní programování)

Testování černých skříněk je důkladnější, ale pracnější

- Je zaměřeno jednoznačně na rozhraní
- Nevadí osobní vazby mezi kodéry a testéry, není tendence krýt kamarády
- Testéři by neměli navrhovat opravy, poněvadž to podle zkušeností zhoršuje jejich kapacitu detekovat chyby
  - důsledek: při oponenturách detekuji případy selhání, ale nemám nic opravovat
  - Podobně by to mělo být i uvýstupů testů a zaznamenávaní stížností uživatelů
    - Detekuji problém ale **nesnažím se ho ihned řešit**

Účinnost testování

Částí (unit tests)	24%
Systému	36%
Funkcí	25%
Integrační	24%

Provádí-li se testování částí, systému a integrační testování, je celková účinnost 1-0.76*0,75*0,64= 0,64.Je to asi příliš pesimistické, poněvadž se nejvíce frekventované chyby se odhalí dříve, i tak hlavní závěr o malé účinnosti platí

Provádí-li se testování částí, systému a integrační testování, je celková účinnost 1-0.76*0,75*0,64= 0,64.Je to asi příliš pesimistické, poněvadž se nejvíce frekventované chyby se odhalí dříve, i tak hlavní závěr o malé účinnosti platí

Výpočet pravděpodobnosti

- $P_{odhalí} = 1\text{-}P_{neodhalí}$
- $P_{neodhalí} = \prod_i (1\text{-}P_{odhalí\text{-}i})$
- $P_{odhalí} = 1 - \prod_i (1\text{-}P_{odhalí\text{-}i})$

Trochu terminologie

- **Validace:** Ověření správnosti systému nebo jeho funkčního modelu pokusem
  - Standardní test (věc vývojářů)
  - Předvádění uživatelům (tak to chápe ISO 9000 ISO 25000, viz též ISO 20000)

V obou případech obdobné techniky.

  - Předvedení bývá spojeno s testy systému a samozřejmě s předáváním.

- **Verifikace:** Ověření správnosti nějakého dokumentu důkazem či oponenturou.
  - Často výstupního dokumentu etapy proti zadávajícímu dokumentu etapy (vize/cíle * specifikace požadavků, specifikace požadavků * návrh, …)

Další způsoby léčby vodopádu

Zkušenost ukazuje, že ani verifikace s validací nestačí, protože nedostatečně chrání proti chybám ve specifikacích a v managementu projektu. Skutečně radikální léčba musí proto modifikovat metodu vodopádu – vložit častější zpětné vazby. Používaná řešení (lze je kombinovat):

- Validovat specifikace předvedením SW prototypů (částečně funkčních modelů systému). To probudí blokované znalosti – lidé si uvědomí, co je třeba
- Validovat i prototypy (spirálová metoda)
- Postupně systém jako monolit rozšiřovat (iterativní metoda)
- Systém inkrementálně rozšiřovat integrací autonomních aplikací integrovaných jako služby (inkrementální vývoj).

Pro velké systémy a v řadě jiných situací je nutný inkrementální vývoj. Ten ale předpokládá vhodnou architekturu softwaru.

Zlepšování vizí u veřejných IS (státních)

- Často narazíme na to, že zákony značně omezují positivní efekty IS (viz ochrana osobních dat). Je tedy žádoucí zákony změnit
  - Lobování (působení na poslance přes známé)
  - Diskuse v médiích
  - Odborné články a jiné metody opřesvědčování veřejnosti

Další způsoby léčby

Je důležité nepřecházet do následující etapy předčasně, dříve než nejsme schopni v dané etapě něco rozumného za rozumnou cenu a v rozumném čase dělat.

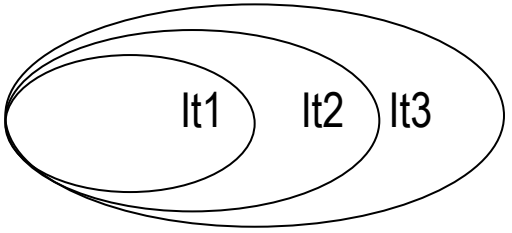
Začni programovat co nejpozději!

Pozor: Formální specifikace mají společné aspekty s programováním

Vyvíjet agilně (řada iterací za účasti uživatelů, vhodné pro menší projekty, v SOA i pro velké systémy, tam se ještě zkoumá)

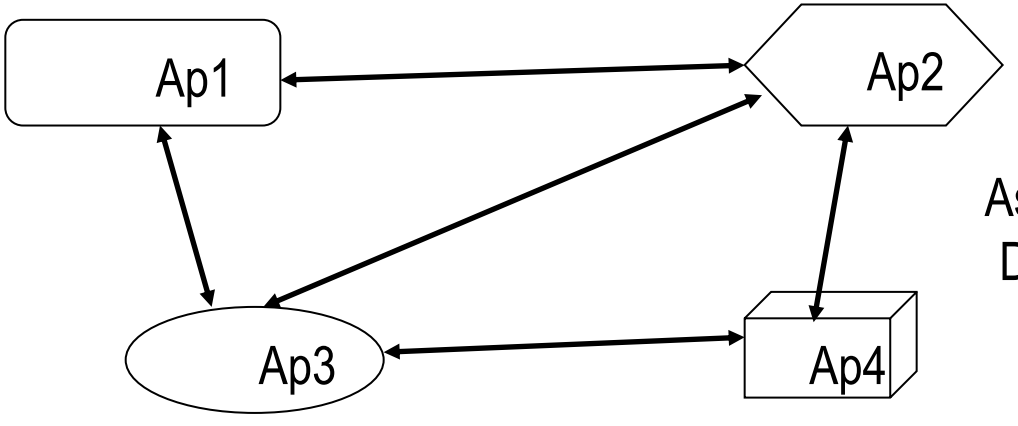
Jak oslabit nevýhody vodopádu

1. Zlepšit kvalitu specifikací
   1. Myšlenkové pokusy (what if, a co když)
   2. Ověření pomocí prototypů případně formalizace, ale až když je intuitivně jasno
   3. Oponentura specifikací (feasibility study)
2. Rozdrobení vodopádu na řadu menších vodopádů (kaskáda)
   1. Agilní vývoj
   2. Iterativní vývoj (postupné zvětšování programu), nejen agilní metodikou
   3. Inkrementální vývoj (integrace programů, využití stávajících programů, většinou servisně orientované architektury)
   4. Na rozhraní posledních dvou bodů je využívání zásuvných modulů



Obvyklé rozhraní: RPC, API

Iterativní vývoj –jedna rostoucí aplikace (může být distribuovaná), pro vývoj iterace potřebuji mít již vyvinuté jádro a **to potřebuji při vývoji iterace používat. Typické pro agilní vývoj**



Asynchronní komunikace společná DB (menší systémy) architekturní služby

Inkrementální vývoj, propojují se (různorodé) aplikace Ap1, Ap2, .. většinou pomocí výměny zpráv nebo dat přes middleware, nebo přístupu ke společným datům (datovým úložištím). Při vývoji přírůstku **nepotřebuji mít k dispozici zbytek systému!!!, Agilní vývoj potřebuje specifické nástroje**

Agile development prefers:

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

Není dobré pro kritické a komplexní systémy
Dokumentace, smlouva atd. v nejmenším rozsahu potřebném pro spolupráci, ne však menším (základní údaje ve smlouvě, dokumentace pro uživatele by měla bý, jakýsi plán by měl být, nástroje se mnohdy hodí).

Zásady agilního vývoje

- Pro malé nekritické aplikace nebo SOA
- Princip mnoha malých kroků se zjednodušeným životním cyklem, spíše iterativní než inkrementální vývoj
  - **Extrémní programování**
  - **Scrum**
  - Dynamic systém development method
  - Adaptive software development
  - Feature software development
  - Lean development
  - Agile modeling
- Chybí návaznost na rigorózní metodiky a na SOA
  - Částečné řešení agilní SOA pomocí architekturních služeb
  - Problém je stálý kontakt s uživateli (mají svých starostí dost)

Principy agilního vývoje 1

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.

Principy agilního vývoje 2

5. Build projects around motivated individuals.
   Give them the environment and support they need,
   and trust them to get the job done.
6. The most efficient and effective method of
   conveying information to and within a development
   team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.
   The sponsors, developers, and users should be able
   to maintain a constant pace indefinitely.

Principy agilního vývoje 3

8. Continuous attention to technical excellence and good design enhances agility.
9. Simplicity--the art of maximizing the amount of work not done--is essential.
10. The best architectures, requirements, and designs emerge from self-organizing teams.
11. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile development prefers:

A. Working software over comprehensive documentation

B. Individuals and interactions over processes and tools

C. Customer collaboration over contract negotiation

D. Responding to change over following a plan

To ale neznamená, že se např. dokumentace vůbec nedělá

Problém agility

- Zajistit spoluprácí se zákazníky
- Jde v podstatě o postup pokus-omyl a ten není vhodný pro
  - Zajištění dlouhodobých vizí, strategii
  - Vývoj kritických aplikací
  - Vývoje velmi velkých systémů

Architektura, shrnutí

- Struktura systému ve velkém.
  - Komponenty, jejich vlastnosti a spolupráce a rozhraní navenek
- Jednotící idea/filosofie a framework
- Základní vlastnosti rozhraní systému a jeho částí
- Systém může mít různé architektury z hlediska HW, logiky a fyzické dekompozice softwaru
- Architektura ovlivňuje uživatelské vlastnosti systému, dostupné operace, techniky specifikací a metodologii vývoje, použité technologie

Divný výsledek agilitity

- Scrum jako manufaktura
- Dělá se i s větším zapojením lidí do analýzy

Účel architektury

- Celková koncepce a návrh, jednotící idea, porozumění celku
- Usnadňuje zobrazení celku (dílenský výkres)
- Většinou prostředek dekompozice, porozumění, analýzy (např. konzistentnost a kvalita návrhu)
- Důležité pro
  - Zvládnutí vývoje, zlevnění vývoje, SW inženýrské výhody jako škálovatelnost a flexibilita
  - Znovupoužitelnost částí, využívání produktů třetích stran
  - Údržbu
  - Dostupnost některých manažerských operací, např. outsourcingu
  - Podporu decentralizace
  - …..
- Usnadňuje řízení (procesy, malé etapy, inkrementální vývoj, minimální rozsah, rozšiřování)

Architektura klient – server

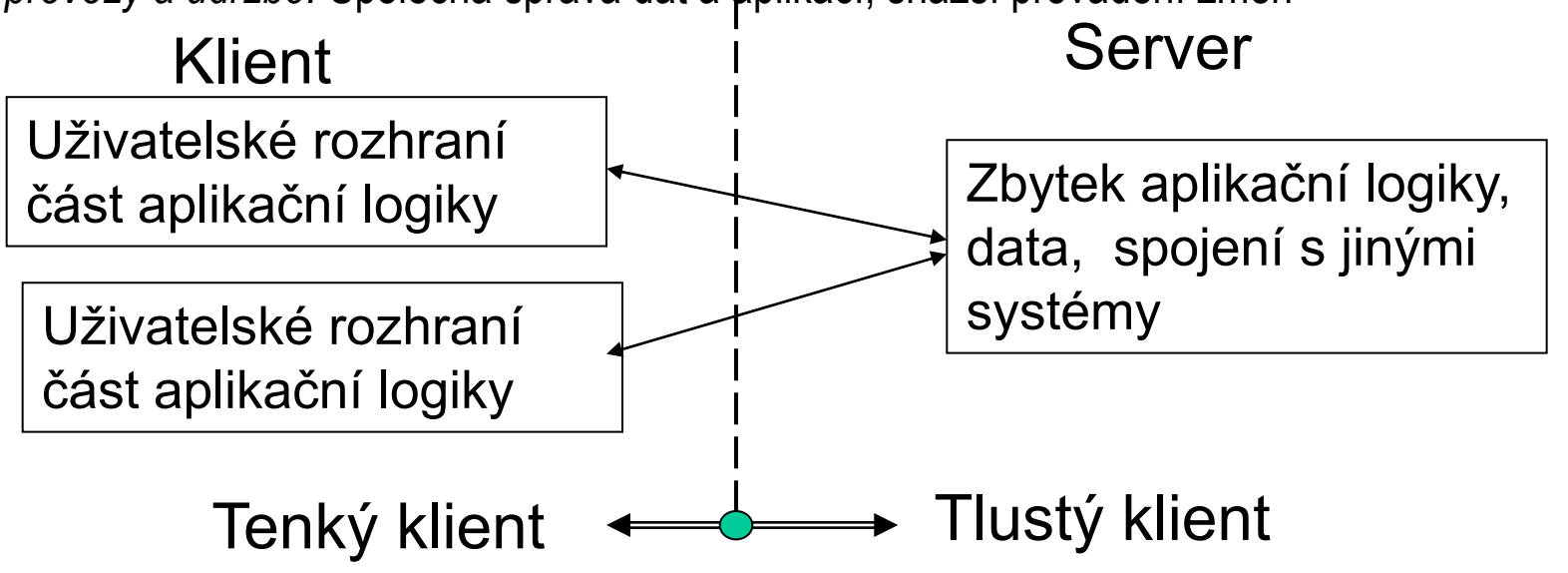
Důvody používání

,

Úspora paměti a nákladů koordinace (kód a společná data pouze jednou nebo málokrát, opravy na jediném místě)

Úspora výkonu – vysvětlení – server jako služba, výkon součet nezávislých n.v. s velmi malou střední hodnotou a velkým rozptylem

Úspory při provozy a údržbě: Společná správa dat a aplikací, snazší provádění změn



Tenký a tlustý klient
Výhody tenkého klienta

- Úspory nákladů na hardware klientů, je-li mnoho klientů
- Snazší správa
  - Bezpečnost
  - Méně záplat prováděných u klientů
  - Méně příležitostí k flákání hraním her
  - Lepší kontrola práce (nesmí se ale přehánět)
  - Snazší upgrade

Výhody tenkého klienta

- Snazší využívaní centrálních zdrojů
- Snazší udržování centrálních zdrojů
- Úspora nákladů na koupi a provoz HW

Úspora výkonu

- Zátěž klienta se chová jako náhodná veličina s velmi malou střední hodnotou *M* a poměrně velkým rozptylem *D*. *Pokud klient nevyužívá server pak* obvykle stačí výkon klienta zajišťující, že se jeho kapacita překročí jen v 1% případů. Výkon klienta by tedy neměl být menší než $M+3\sqrt{D}$ *(hranice konfidenčního intervalu). Pro velké D je to dosti vysoký požadavek*

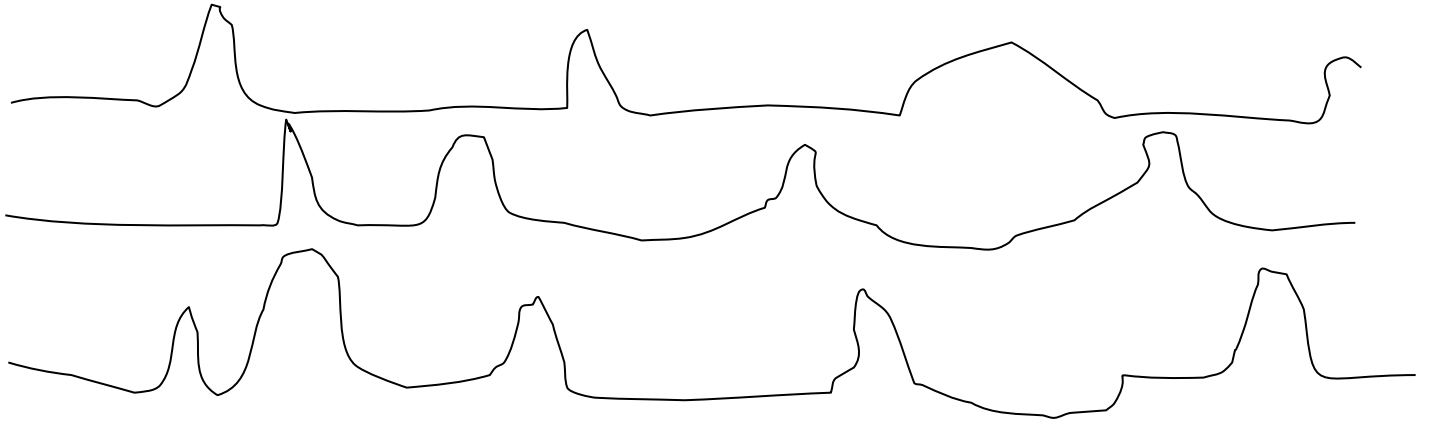
Úspora výkonu

- Přesunutím části zátěže (špiček) z klienta na server lze často dosáhnout toho, že se zátěž na klientovi chová jako n.v. se střední hodnotou M_1 a rozptylem D_1, $M_1 < M, D_1 << D$.
- Na server se přesune zátěž s parametry M_2 , D_2 , kde $M_2 \sim 0$ a $D2 \sim D_1$. Poněvadž jsou zátěže od různých klientů nezávislé stačí na serveru (za předpokladu, že se všichni klienti chovají stejně a je jich n) výkon

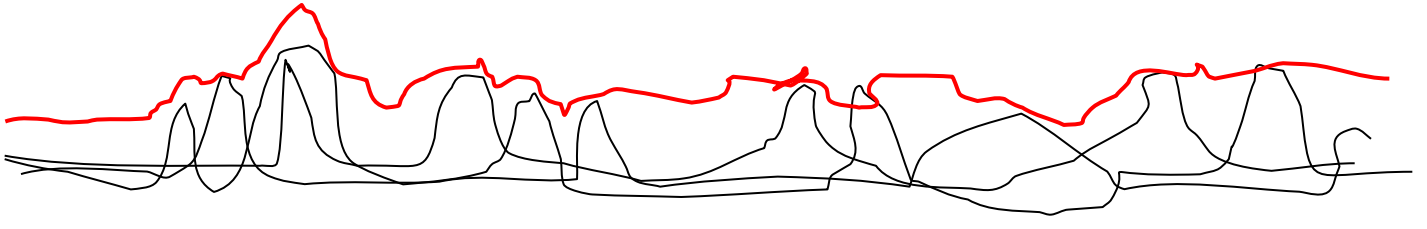
$$nM_2 + 3\sqrt{n}\sqrt{D_2} << nM + 3n\sqrt{D}$$

Na klientu stačí výkon $M_1 + 3\sqrt{D_1} << M + 3\sqrt{D}$

Průběh zátěže klientů



Průběh zátěže serveru



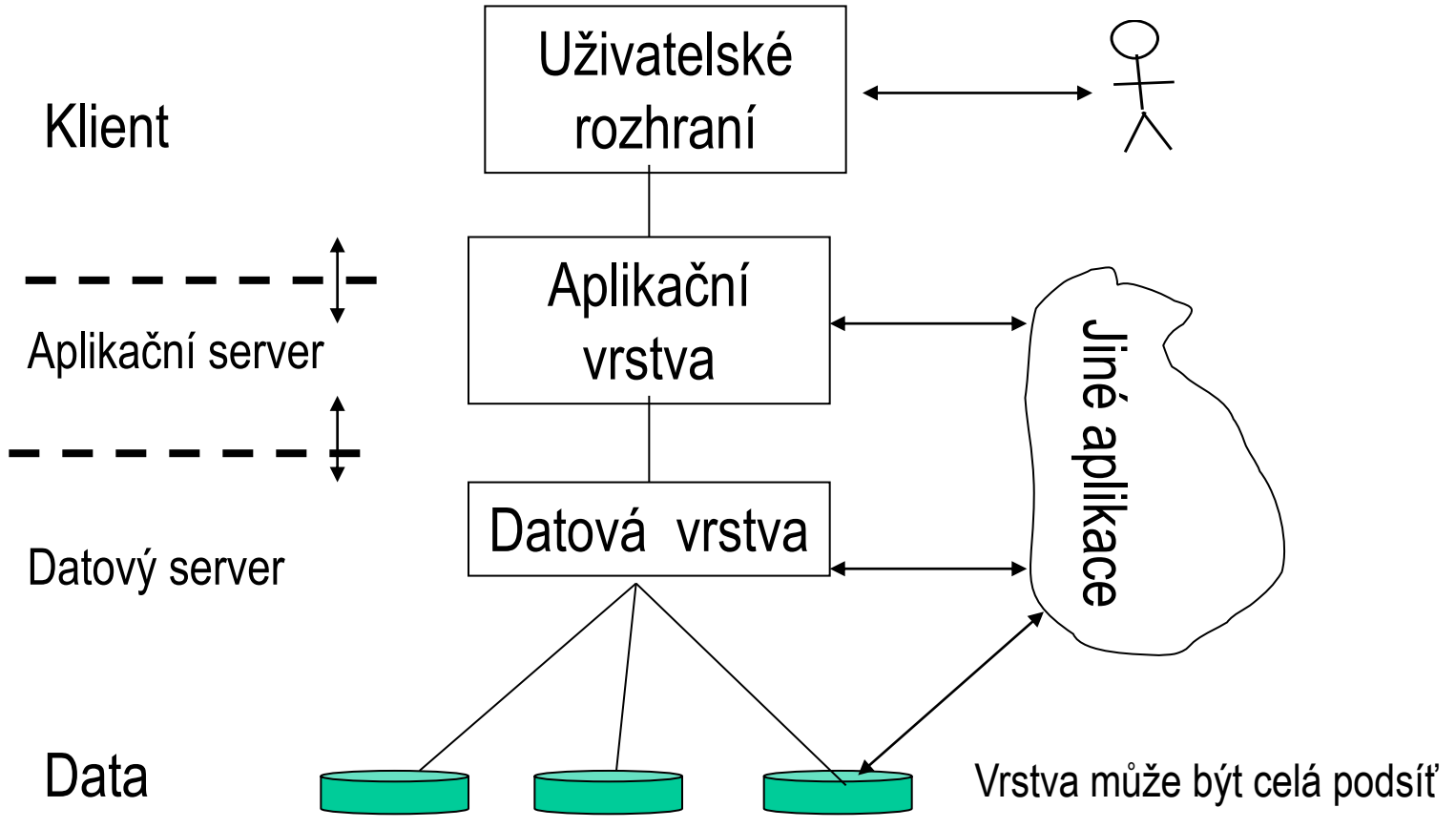
Úspora výkonu

Podobně se řeší i otázka, kolik komunikačních kanálů s malou kapacitou lze nahradit kanálem s velkou kapacitou při statistickém multiplexingu.

Nevýhody tenkého klienta

- Výpadek centrálních služeb znamená totální výpadek systému
- Některé služby mohou být lépe proveditelné lokálně.
- Neochota zvládat nové a cena přechodu na novou architekturu
- Ztráta lokální prestiže (odpovědnost je na centru)
- Obtížnější outsourcing (viz ale servisní orientaci)
- U systémů s několika málo klienty neefektivní
- **Přetěžování sítě u distribuovaných řešení**

Třívrstvá architektura,



Vliv objektové orientace

Objektová orientace usnadňuje přesuny částí programů mezi jednotlivými počítači a mezi sebou, srv. technologii CORBA. To usnadňuje přeměnu tlustého klienta na klienta tenkého a naopak.

Pozor: Nevýhodou je tendence k jemnozrnnému rozhraní použitých SW komponent a k rozhraní, kterému uživatel nerozumí (volání procedur). V SOA to značně ztěžuje používání některých technologií a zhoršuje uživatelské vlastnosti (dynamiky změn, agilitu používání, sourcing a dokoce řešení soudních sporů)

Spoje s jinými systémy, diskuse variant

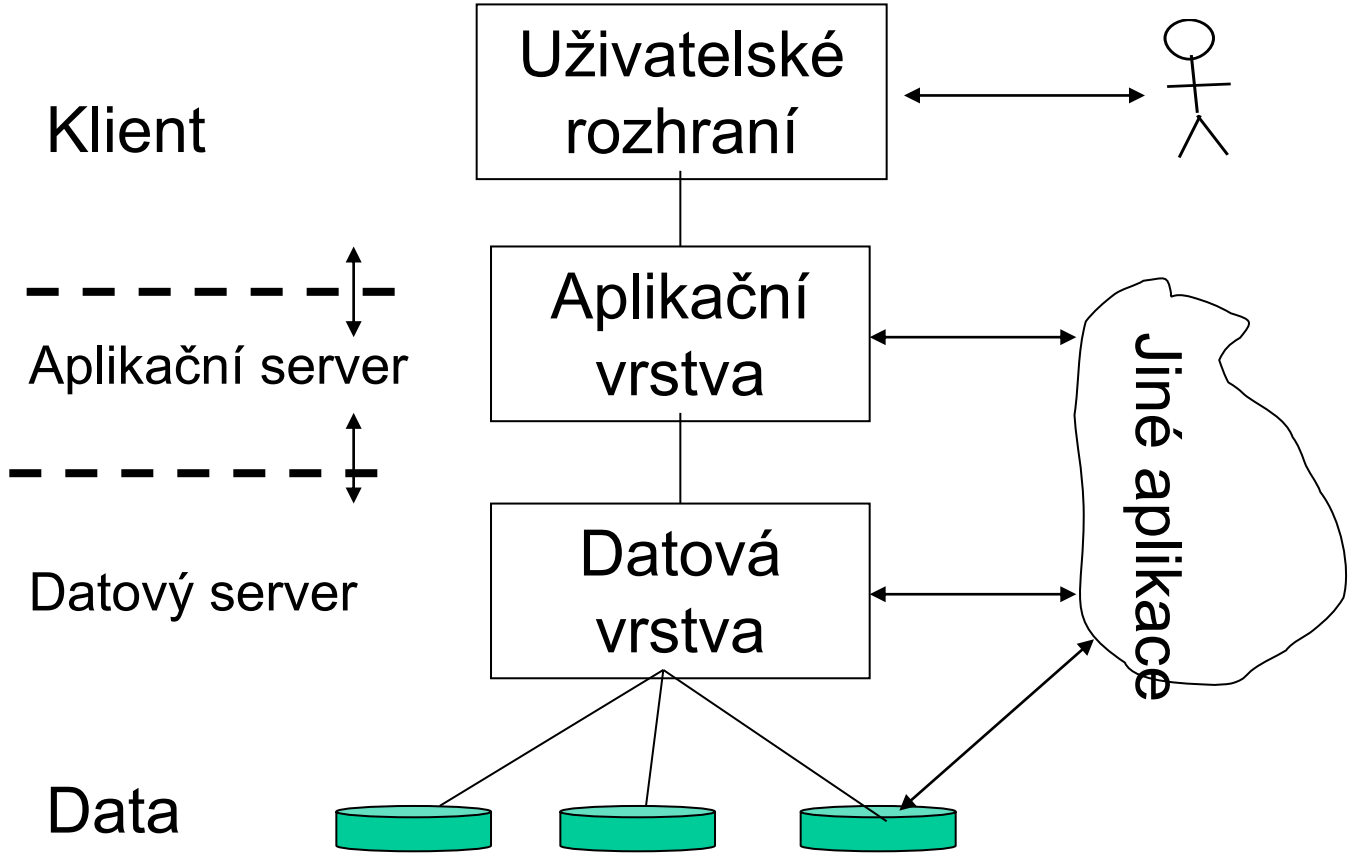
- Spoje na aplikační vrstvu jsou nejbezpečnější, poskytuje přístup k funkcím partnera, avšak jsou málo flexibilní neboť umožňují jen to, s čím tvůrce systémů počítal
- Spoj na datovou vrstvu je flexibilnější, neposkytuje ale funkce partnera a je riskantnější, i když ne příliš.
- Spoj na DB závisí na konkrétní implementaci, mohou být proto problémy s přenositelností, a je velmi riskantní (pokud není omezen na čtení, možnost čtení se málo využívá)

Přístup k datům customizovaných systémů

- Velké systémy, jako je R3 od SAP nebo Oracle Financial mají v podstatě třívrstvou architekturu, byť jsou jednotlivé vrstvy tvořeny sítěmi aplikací/služeb. Data jsou uložena ve standardních databázích.
- V těchto systémech probíhá přechod na servisně orientovanou architekturu (net weaver)
- Je výhodné si u customizovaných systémů sjednat přístup k datům pro čtení pro řešení ad-hoc potřeb managementu. To co se potřebuje lze často zobecnit a dát implementovat dodavateli
  - Ten takový postup z obchodních důvodů nemiluje

Integraci systémů je nejlépe dělat přes vhodný middleware jako virtuální p2p síť (SOA)

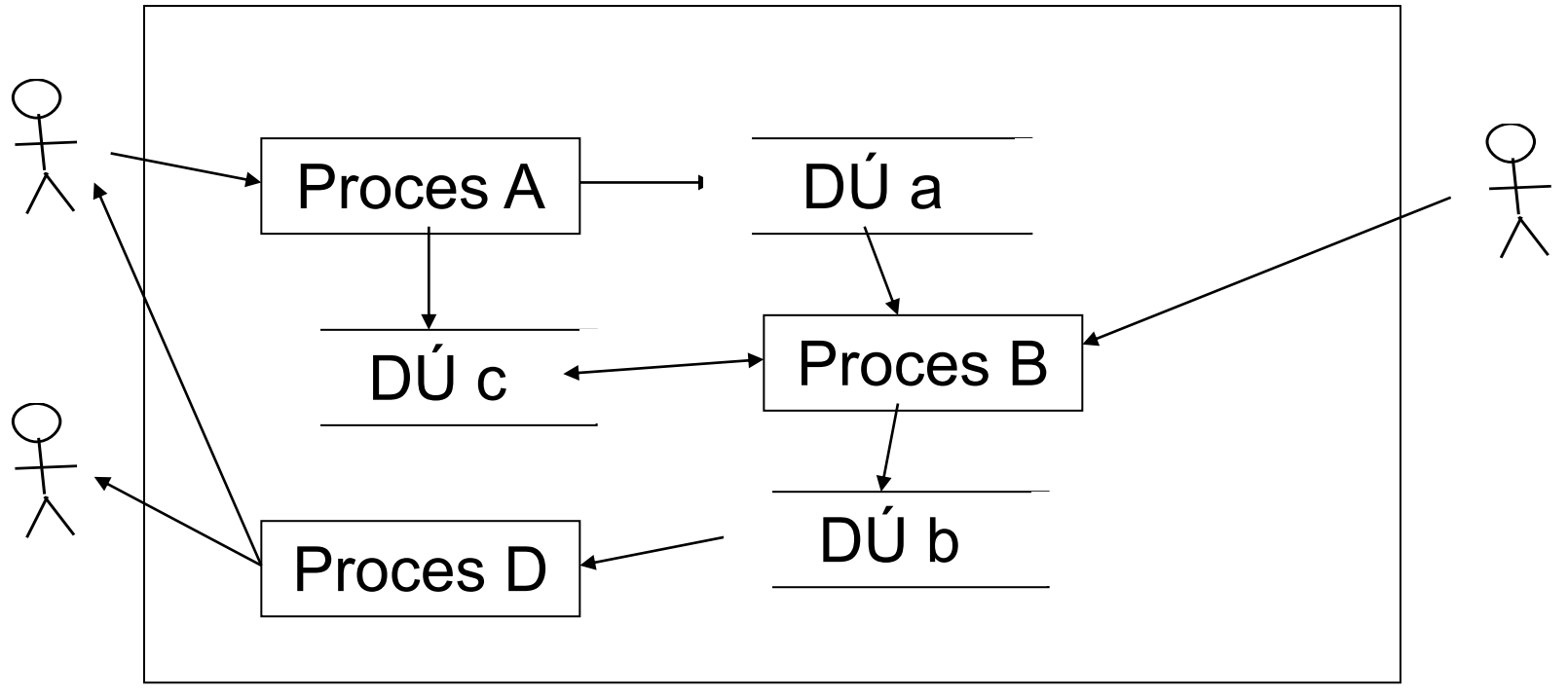
Třívrstvá architektura virtuálně existuje i u sítí programů (SOA)



Pozorování

- Uživatelské rozhraní závisí na místním jazyku a kultuře, jeho vývoj se nedá outsourcovat do Indie. Jsou ale tendence k tomu, aby si je koncový uživatel vyvinul sám (end-user development).
- Specifikace požadavků se nedá plně outsourcovat – je ji nutné dělat ve spolupráci s koncovými uživateli a závisí na místní kultuře.
- Návrh systému je možné dělat zčásti i mimo náš kulturní prostor
- Kódování, testování částí a integrační testování je možné obvykle outsourcovat. Testování systému jen zčásti.
- Univerzálně použitelné programy lze vyvíjet kdekoliv ve světě, uspěji jen vysokou kvalitou práce (výkon 1:20), nejde vždy

Offline komunikace: data-flow diagram



Autonomie procesů Zavrhováno v OO

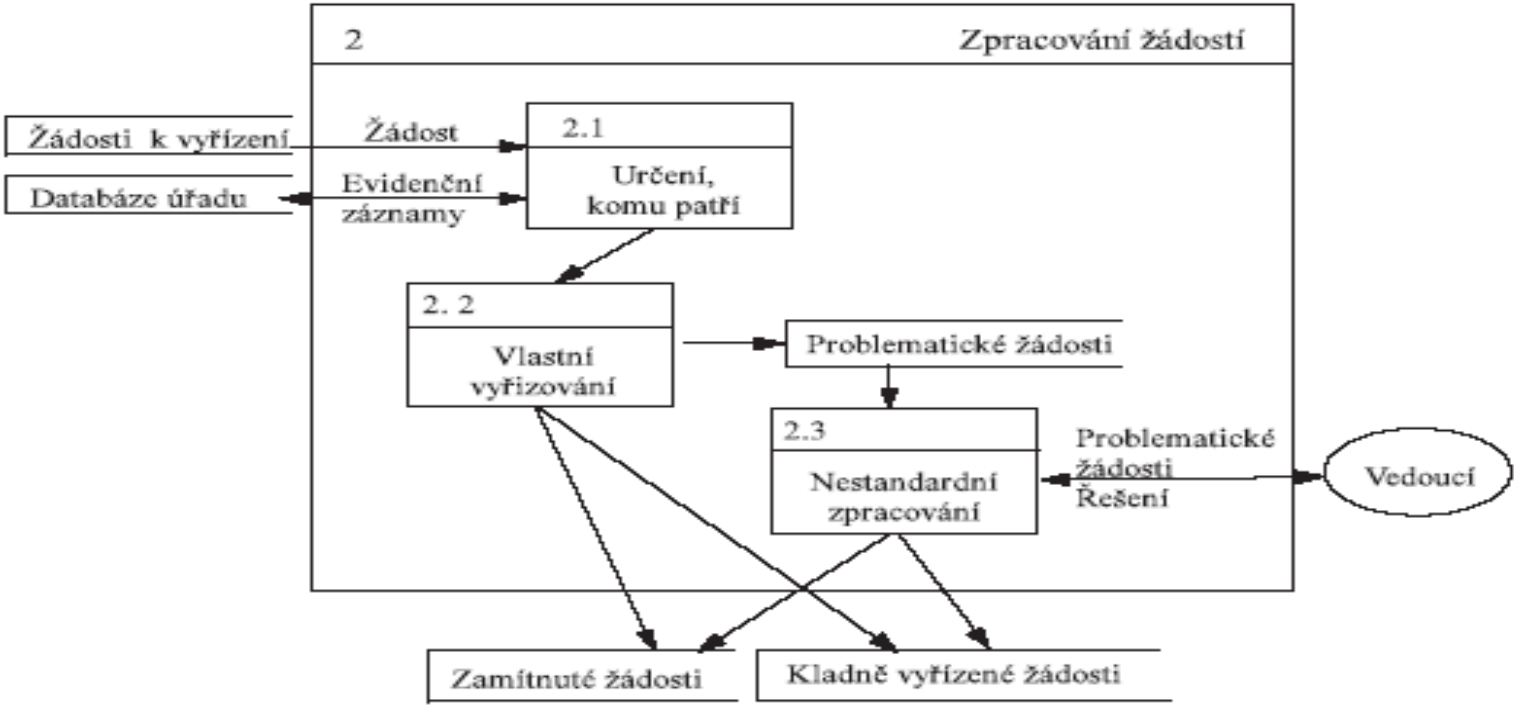
Pohled REA, pohled zdola

- Resource-event- actor,
- Separace procesního uzlu a jeho zobecnění.
- Vstupní zdroje a proces potvrzení, že jsou připraveny (commitment)
- Výstupní zdroje a commitment, že jsou vytvořeny,
- Aktor – ten,kdo za to odpovídá-potřebuje to a je schopen inicializovat a využít

Dataflow digramy (DFD) lze doplnit o toky příkazů

- Lepší řešení je jednotlivé procesy (ve smyslu DFD) zapouzdřit tak, aby se z hlediska řízení chovaly jako služby
- Tím lze zkombinovat výhody dávkových systémů s výhodami SOA
- Pro některé systémy je to nutné, např. při regulaci vyžadující i rozvrhovací výpočty

Dataflow

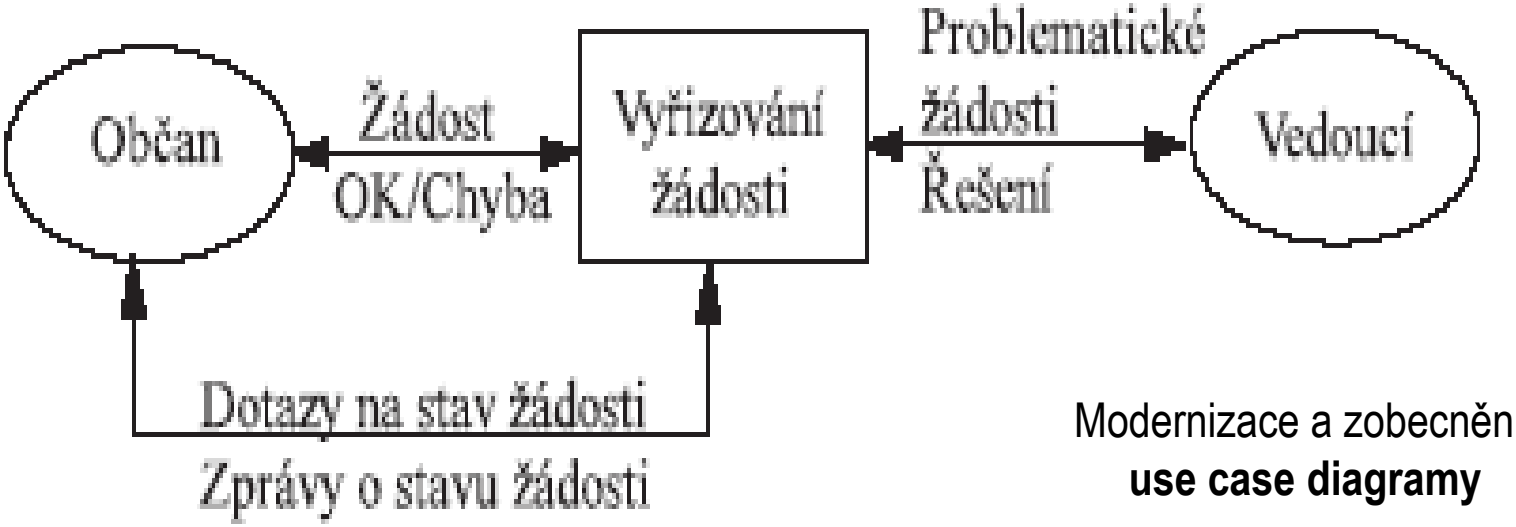


Obr. 12.10: Dekompozice procesu Zpracování žádostí. Kontext diagramu musí odpovídat kontextu procesu Zpracování žádostí v diagramu Vyřizování žádostí.

Odvozená hierarchická dekompozice

Obr. 12.11: Hierarchie vytvořená postupnou dekompozicí systému Zpracování žádostí.

Diagram kontextu.
Dají se použít Use Case



Modernizace a zobecnění:
use case diagramy

Obr. 12.12: Diagram kontextu systému Vyřizování žádostí.

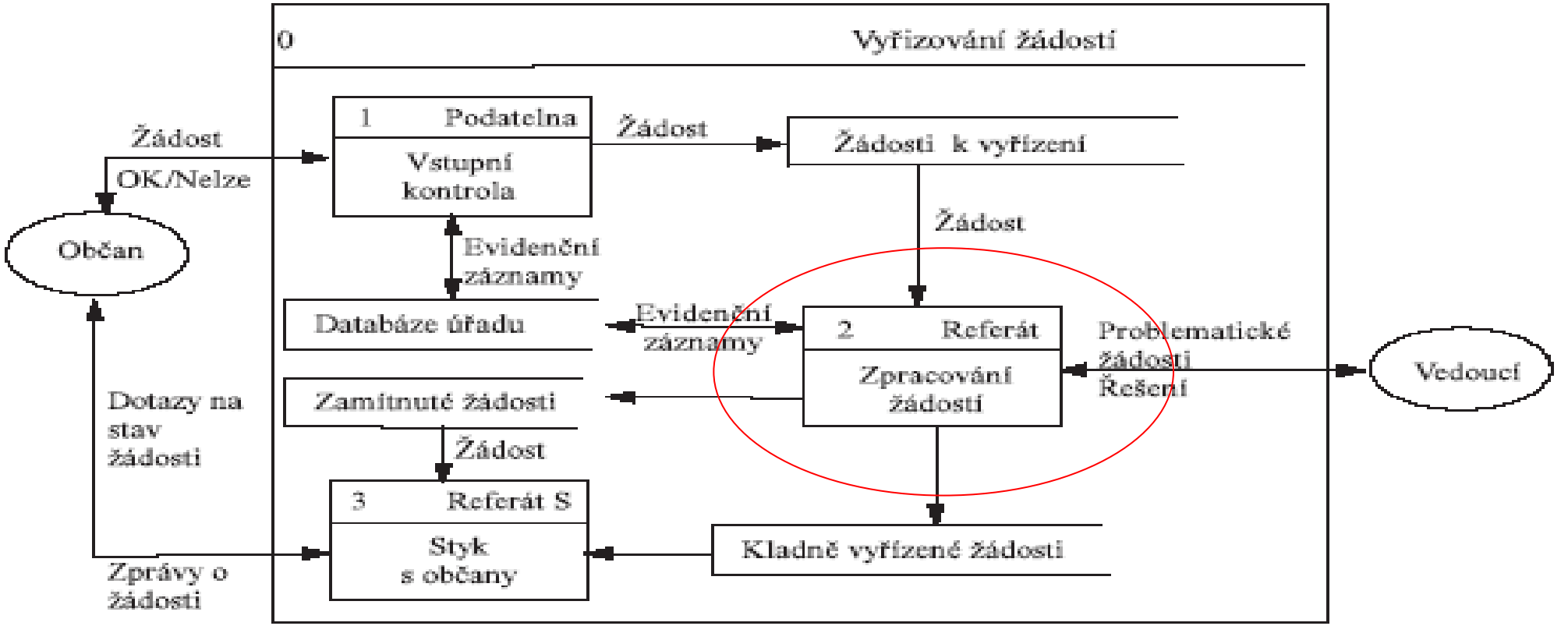
Varianty vývoje

- Případ Y2K ukázal v roce 2000 velkou stabilitu výše uvedených struktur (dadaflow architektur)
  - Systémy byly v té době bez údržby po mnoho let, takže nebyli k dispozici žádní programátoři schopní programy upravit
- Typické pro dávkové zpracování
  - V poslední době se modifikace DFD používá i v objektově orientovaných systémech
- Snadný autonomní vývoj dávkových aplikací
- Aplikace v DFD se v podstatě chovají jako peers ve virtuální síti se speciálním typem komunikace (bulk transfer)

Malé je hezké

- Omezuji automatizaci na ty činnost, kde je lepší, než neautomatizovaná varianta
- Začínám od co nejmenšího rozumně užitečného jádra (iterativní nebo inkrementální vývoj) a i tam umožňuji agilní používání
- Pokud to jde používám dávkové zpracování
- Kombinuji hotové, customizované, nově vyvíjené
- Outsourcuji, ale tak, abych mohl insourcovat

Dataflow a resource event agent, REA



Obr. 12.9: Diagram toků dat systému Vyřizování žádostí.

REA

- Zdroje: vstupy, výstupy a jejich párování
  - Vstupy musí před provedením události výstupy po provedení události splňovat „commitments“
  - Speciální vztah mezi některými vstupy a nekterými výstupy
- Akce je ekonomická událost (proces)
- Agent je osoba inicializující a kontrolující události, koncept agenta lze zobecnit

Vzory událostí REA

- Několik desítek vzorů, např. jak má probíhat kontrakt
- Vázby událostí
- Slabá hierarchizace
- *Hruby, Pavel/ Kiehn, Jesper (CON)/ Scheller, ChrisModel-Driven Design Using Business Patterns:***Model-Driven Design Using Business Patterns,** Springer 2006, *ISBN*: 9783540301547