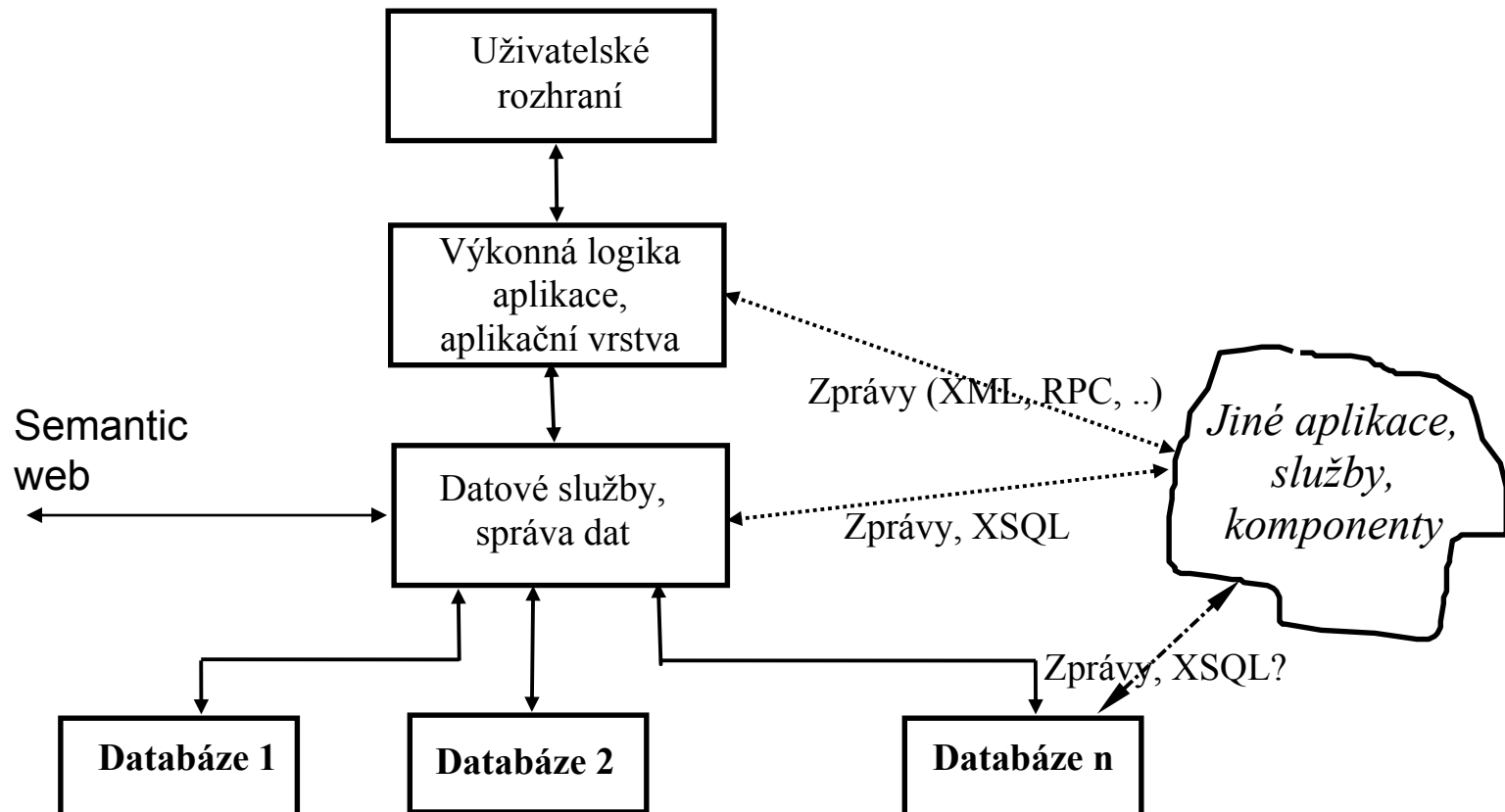


# Metody návrhu systému

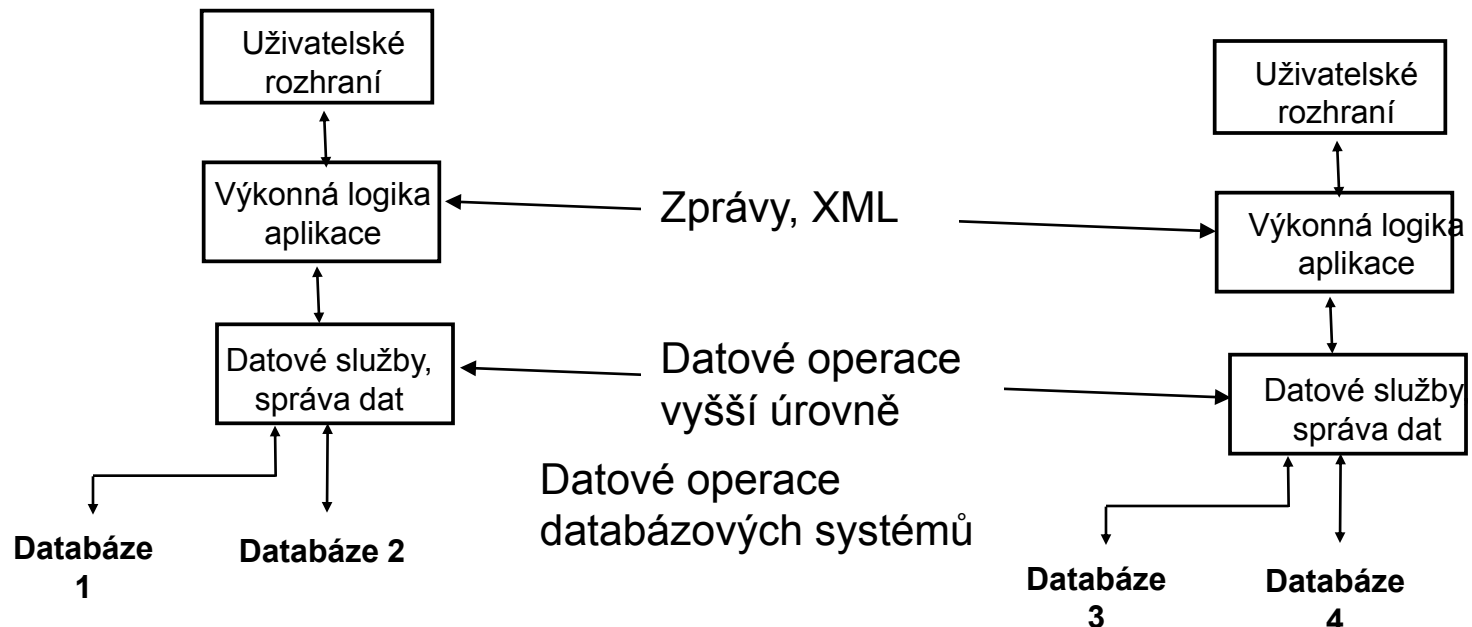
Jak pokračovat po specifikacích

# Spolupráce třívrstevných komponent



# Spolupráce třívrstevných komponent

- Spolupráce podle vrstev,
  - Logika je na aplikačním serveru, datová na datovém

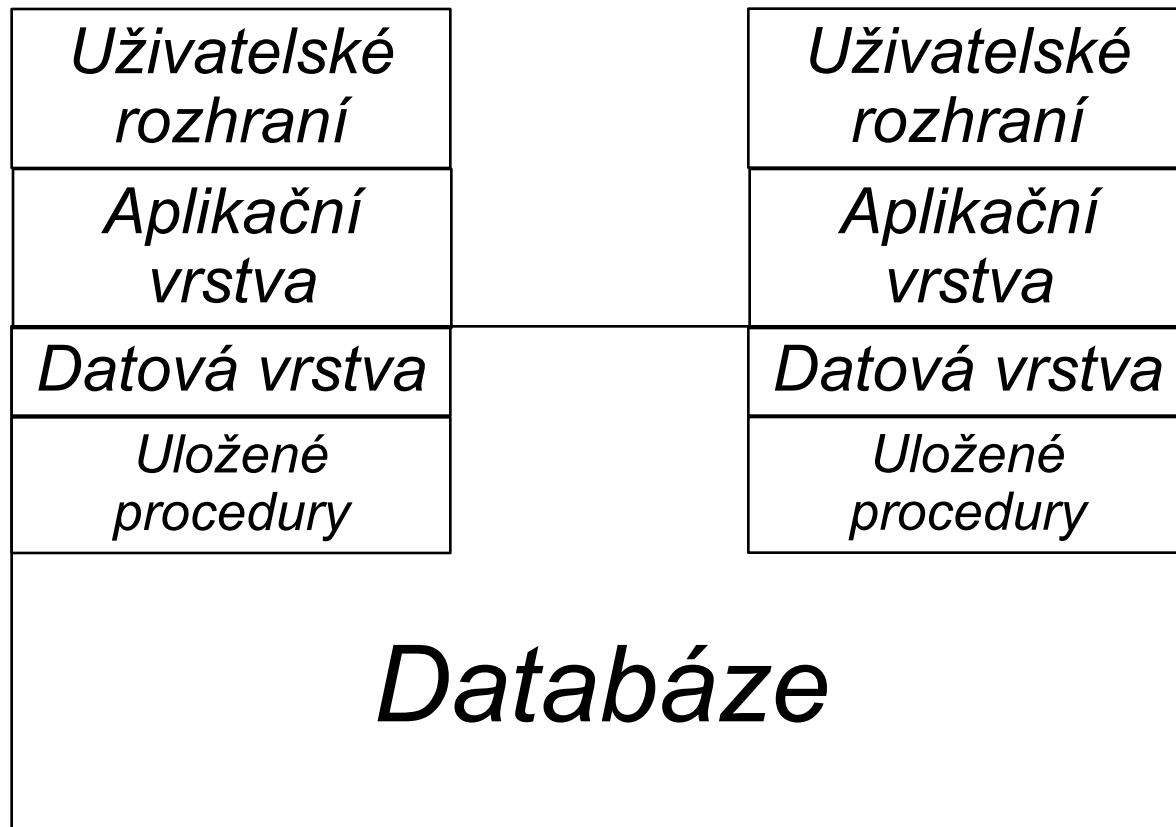


*V SOA mohou být vrstvy tvořeny podsítěmi (pod SOA)*

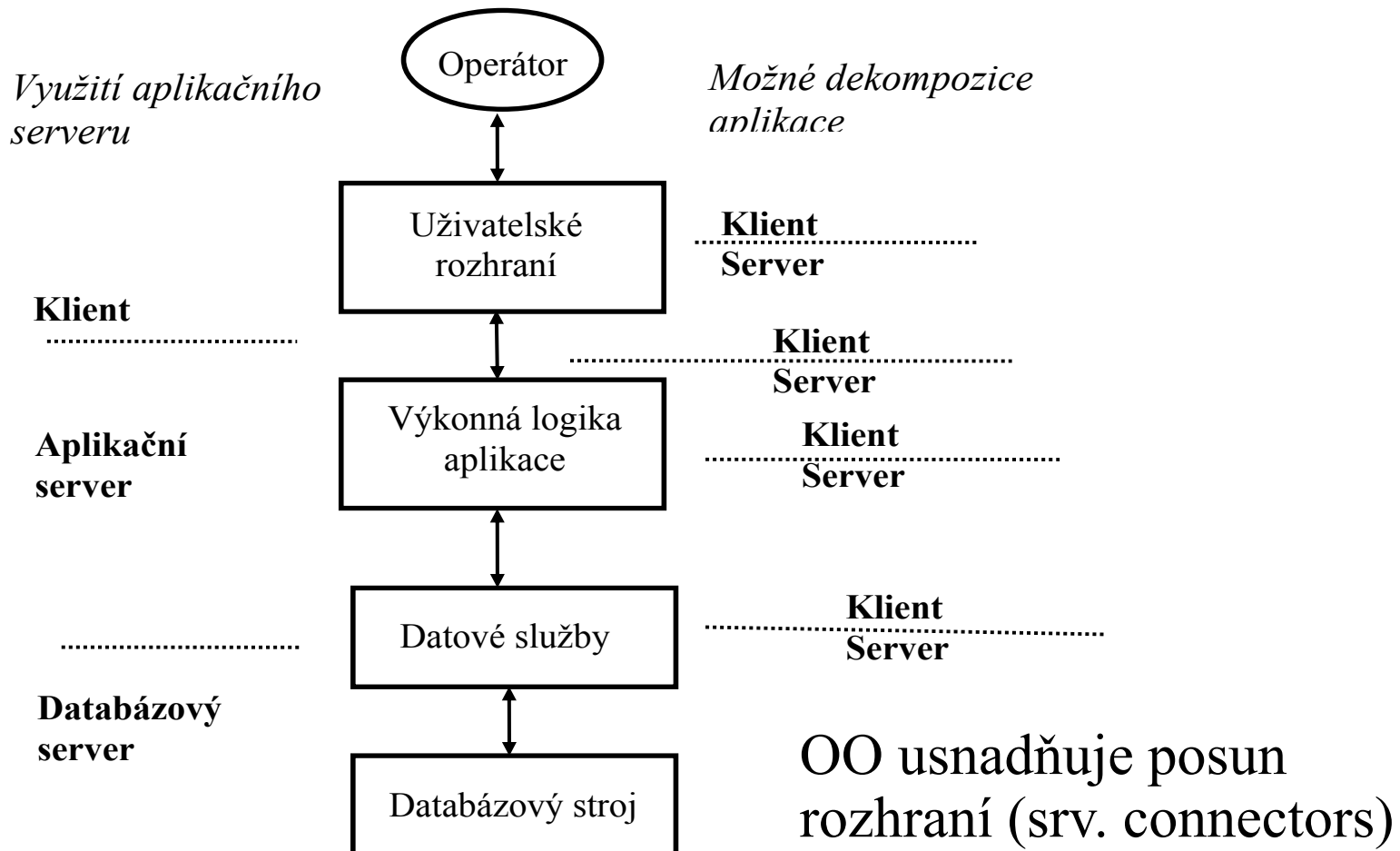
# Databázově orientované systémy

- Aplikace pracující nad stejnou DB
- Aplikační vrstva zčásti pomocí uložených procedur
- Nutná disciplína při vývoji, lze pak vytvořit systém, který se do značné míry obejde bez middlewaru (ten je nahrazen službami databáze)

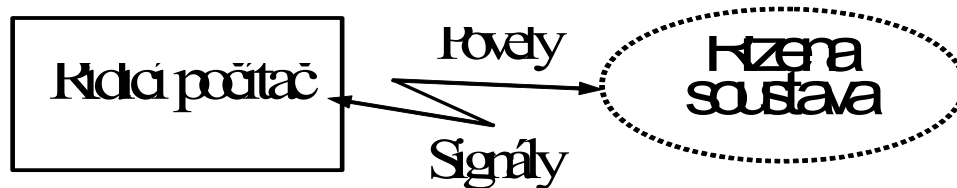
# Systemy propojené přes databázi



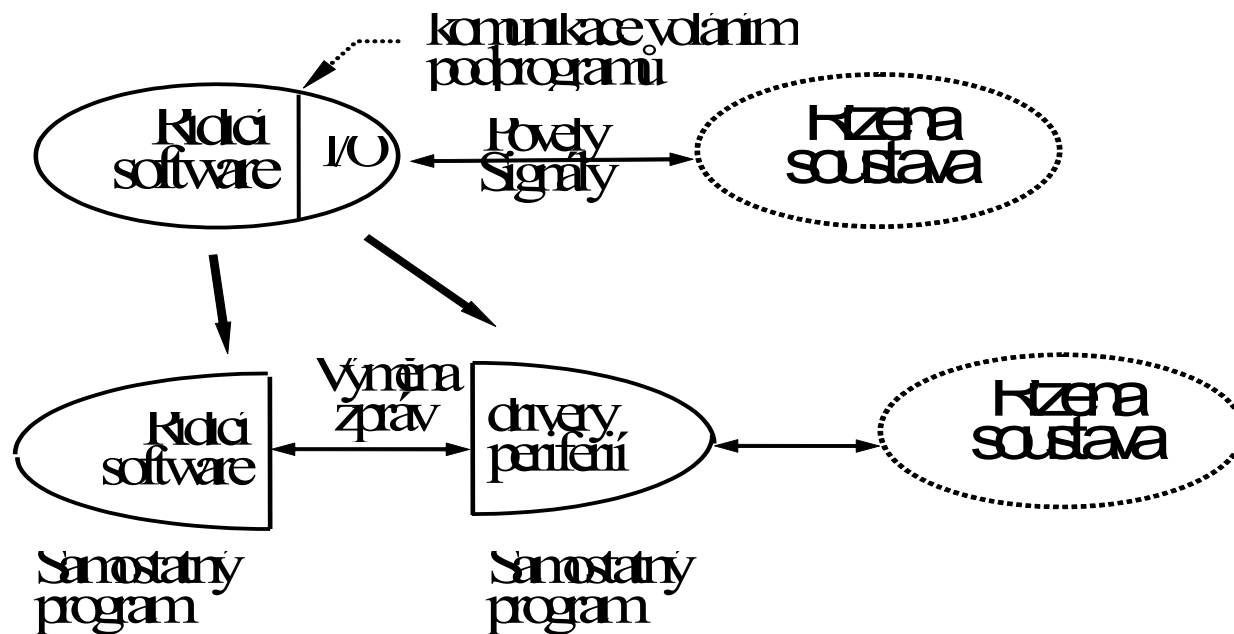
# Tři vrstvy a server



# K SOA

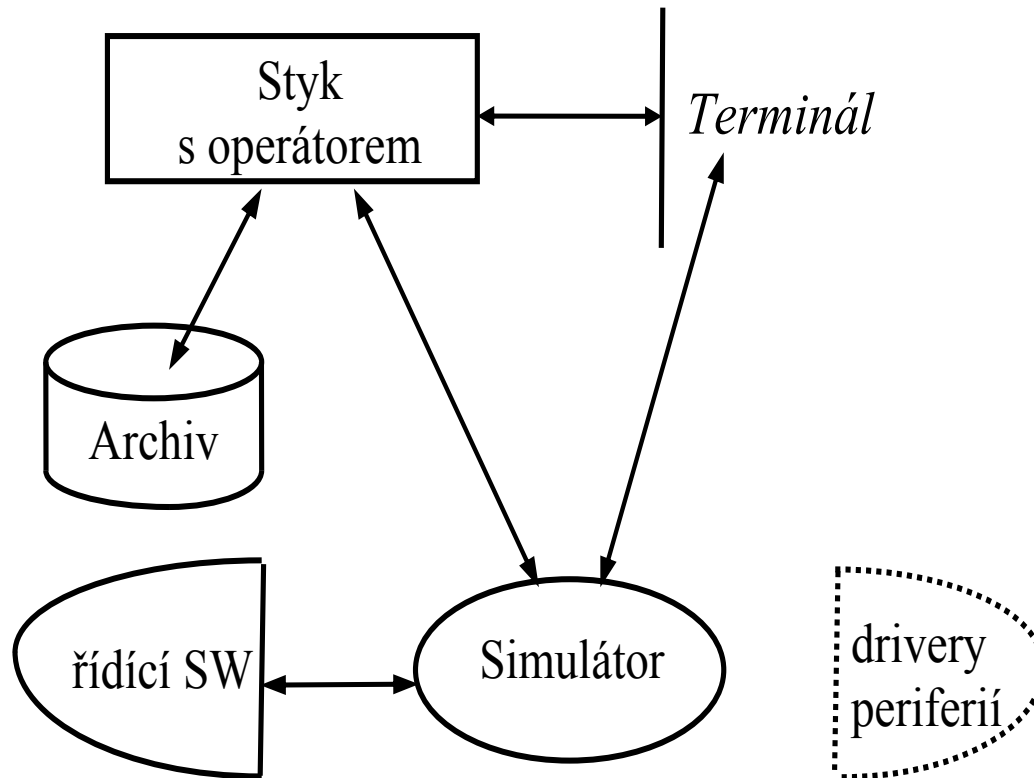


a) Soustava řízená počítačem

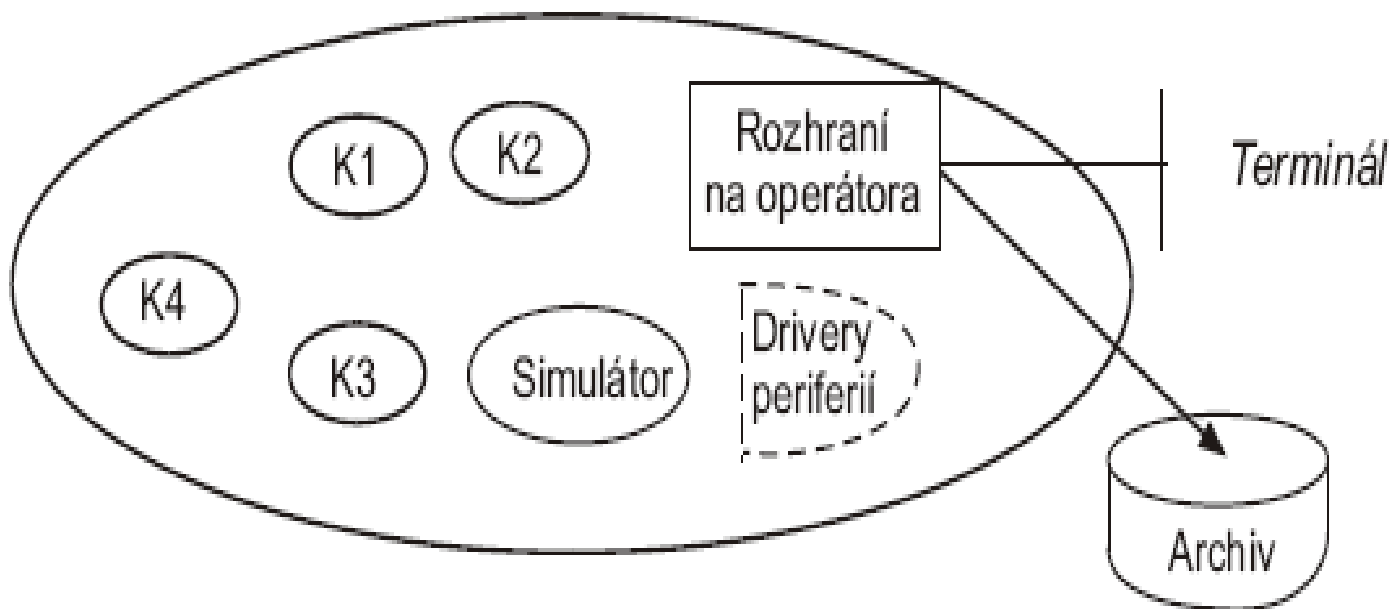


b) Oddělení ovladačů periferie realizujících styk s řízenou soustavou

# K SOA, základní sestava

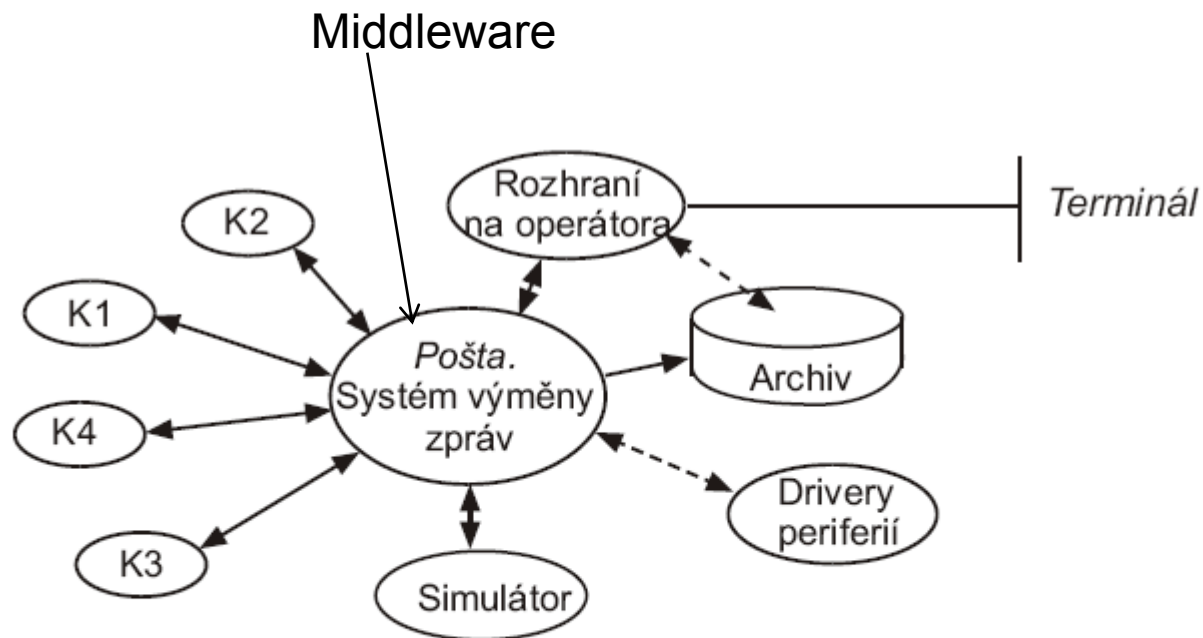






### **System po dekompozici..**

*Každá komponenta může komunikovat s libovolnou komponentou*



**Dekomponovaný systém se subsystémem výměny zpráv.**

*Jednotlivé zprávy jsou posílány prostřednictvím specializovaného subsystému zajišujícího monitorování a přesměrovávání zpráv.*

*Architekturní služby fungují jako rozšíření middleware, architekturu jako službu, agilní vývoj a agilní byznys procesy*

# Hlavní výhody SOA

- Znovupoužití existujících a cizích aplikací
- Autonomní vývoj částí
- **Inkrementální vývoj**
- Modifikovatelnost a udržitelnost
- Umožnění principů agilního vývoje ve velkých systémech
- *Cesta k softwaru jako high tech*

# Některé nevýhody SOA

- Sekvenční zpracování je nutné zajišťovat
  - Řešení: Odpovím určené službě
  - Mohu čekat na odpověď
  - Identifikátor zprávy na kterou se odpovídá a nebo vratný parametr (identifikuje odkud pokračovat)
- Nejasné jak efektivně spolupracovat se SOAP a obecně jak optimálně aplikovat to nejlepší z objektové orientace
- Obtížné přijetí filosofie SOA

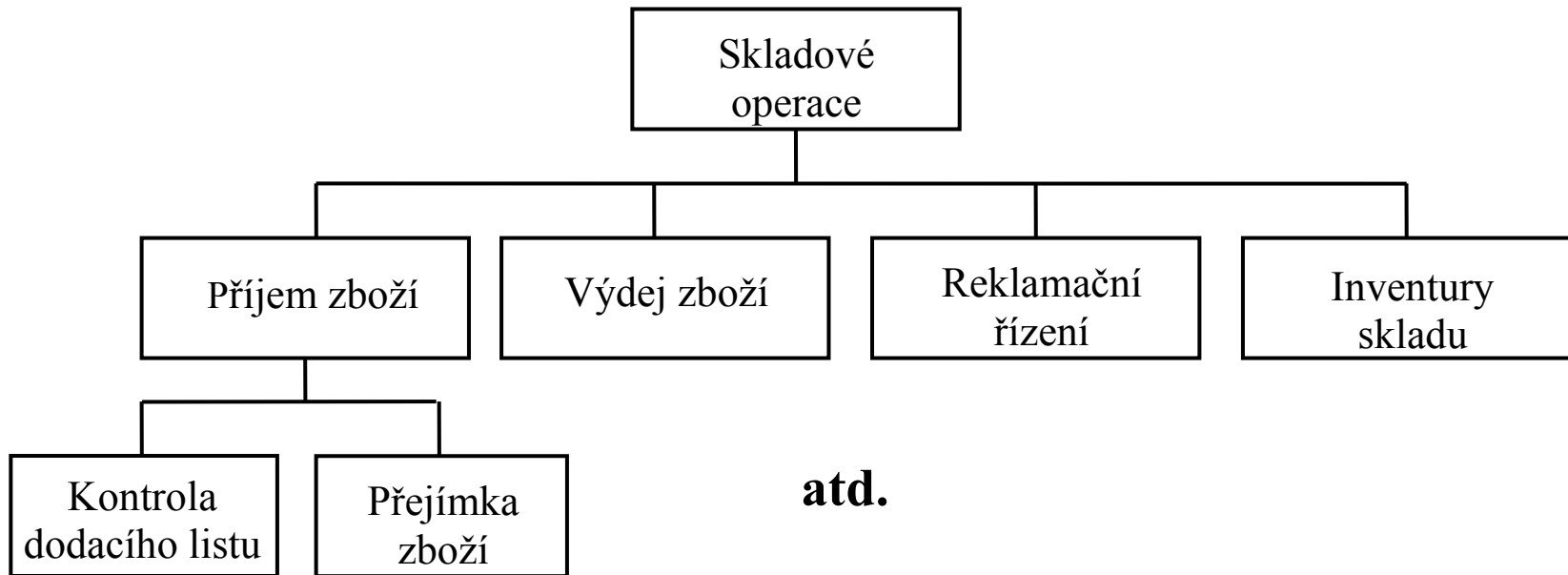
# Jaksonova metoda

- Je vhodná pro sekvenční dávkové zpracování uspořádaných souborů
- Základní princip:
  - Logiku mnohých programů lze odvodit z toho, jak se zpracovává soubor
    - Akce na začátku nebo ří změně klíče (pohyby na daném účtu)
    - Varianty zpracování vět
    - Akce na konci seznamu pro daný klíč
  - Lze opakovat při změně hodnoty klíče

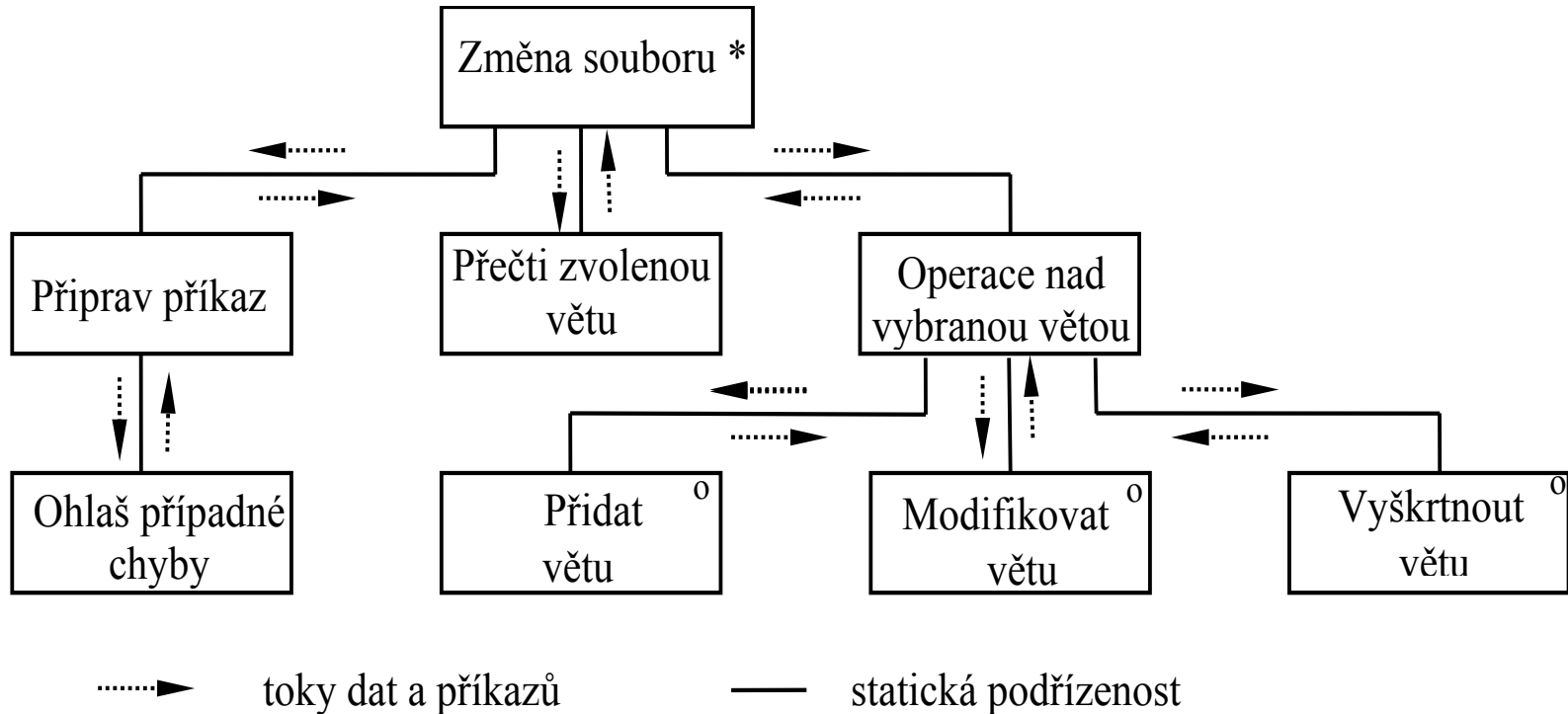
# Jaksonova metoda

- Je vhodná pro vývoj jednotlivých procesů v diagramu toků dat
- V jistém smyslu obdoba objektové orientace pro dávkové systémy

# Jaksonova metoda



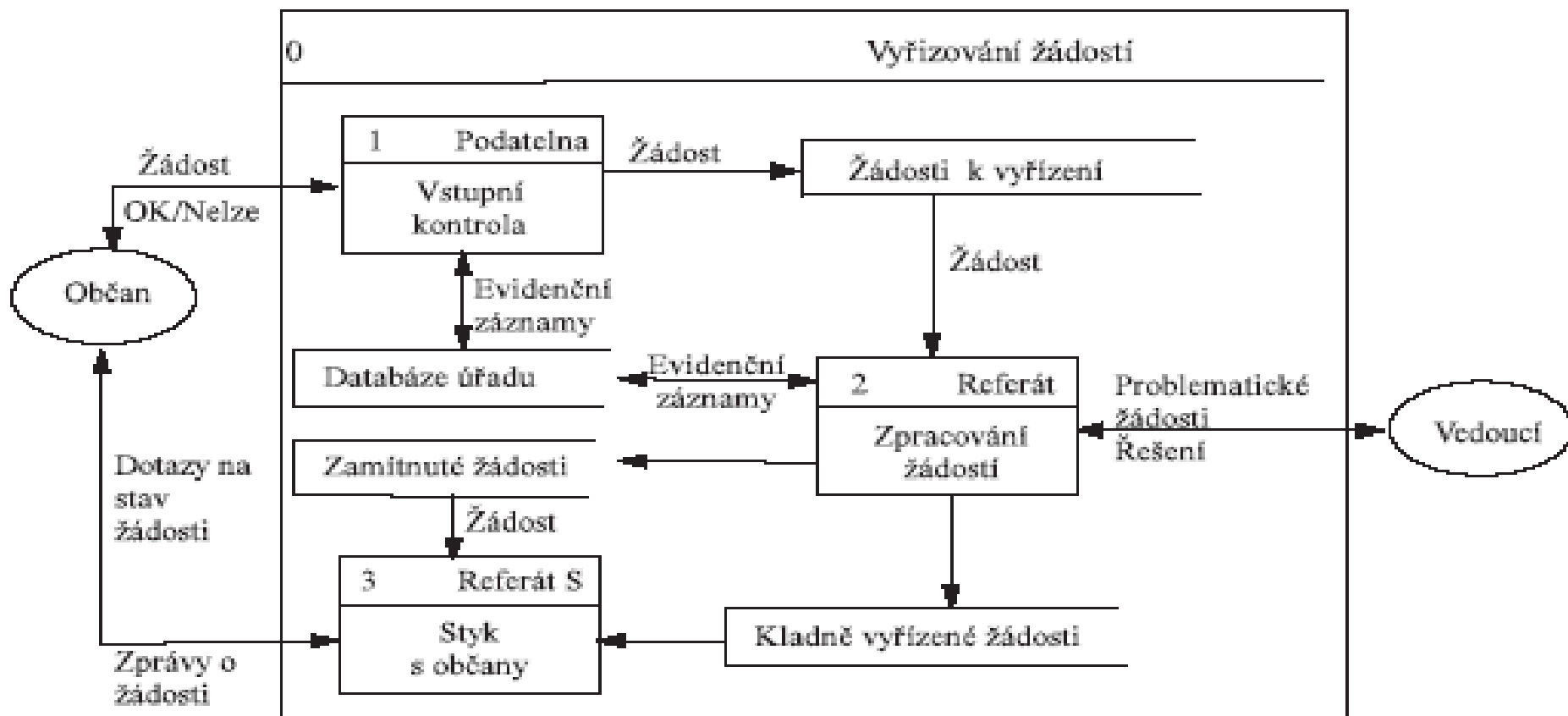
# Jaksonova metoda



Často lze strukturu programu odvodit z dekompozice činností, vhodné pro dávkové zpracování

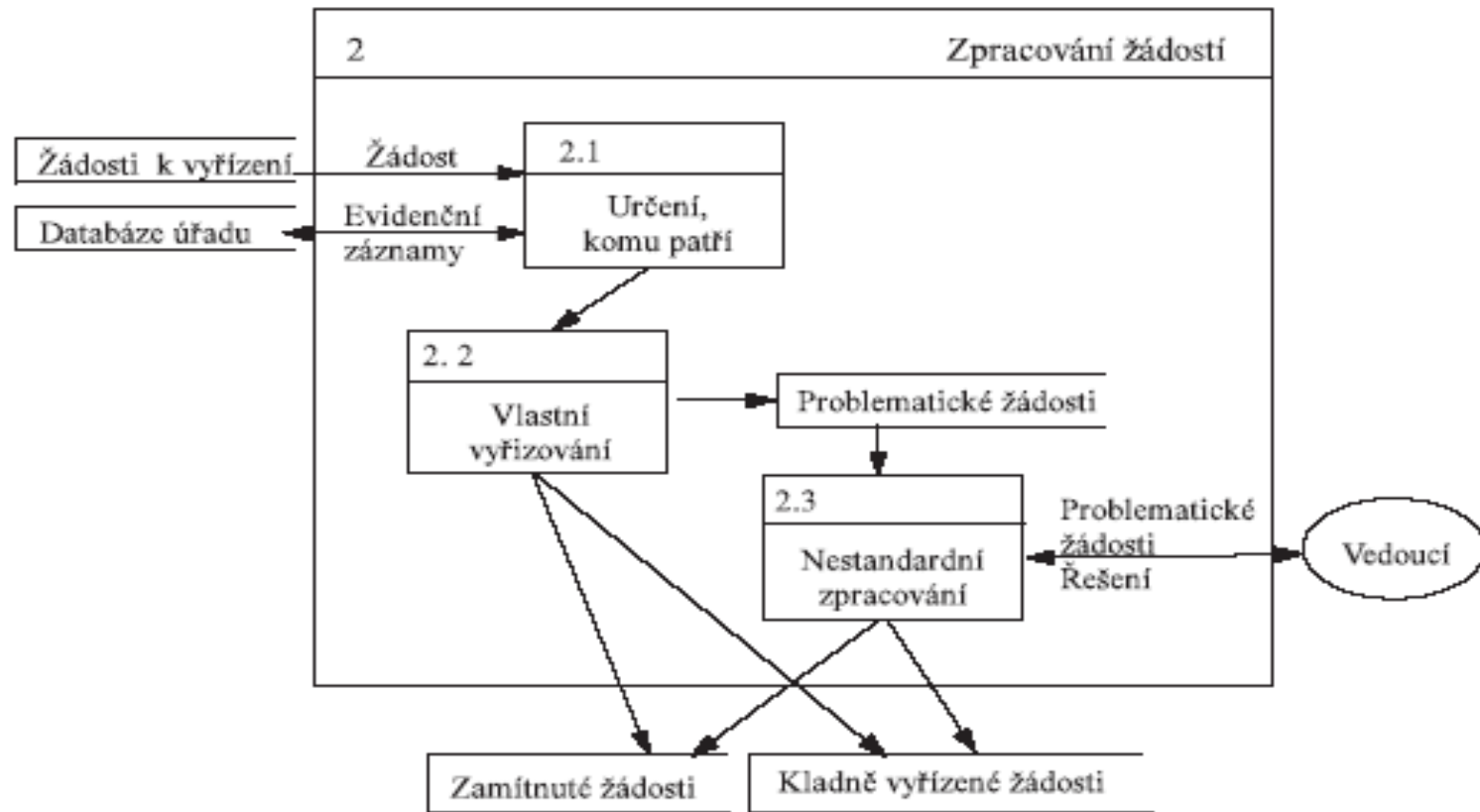


# Dataflow



Obr. 12.9: Diagram toků dat systému Vyřizování žádosti.

# Dataflow, funkcionální dekompozice



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

# Prvky DFD v SOA a cloudu

- Datové úložiště se zapouzdří jako služba
- Rozhraní dávkoé (bulk) i interaktivní)

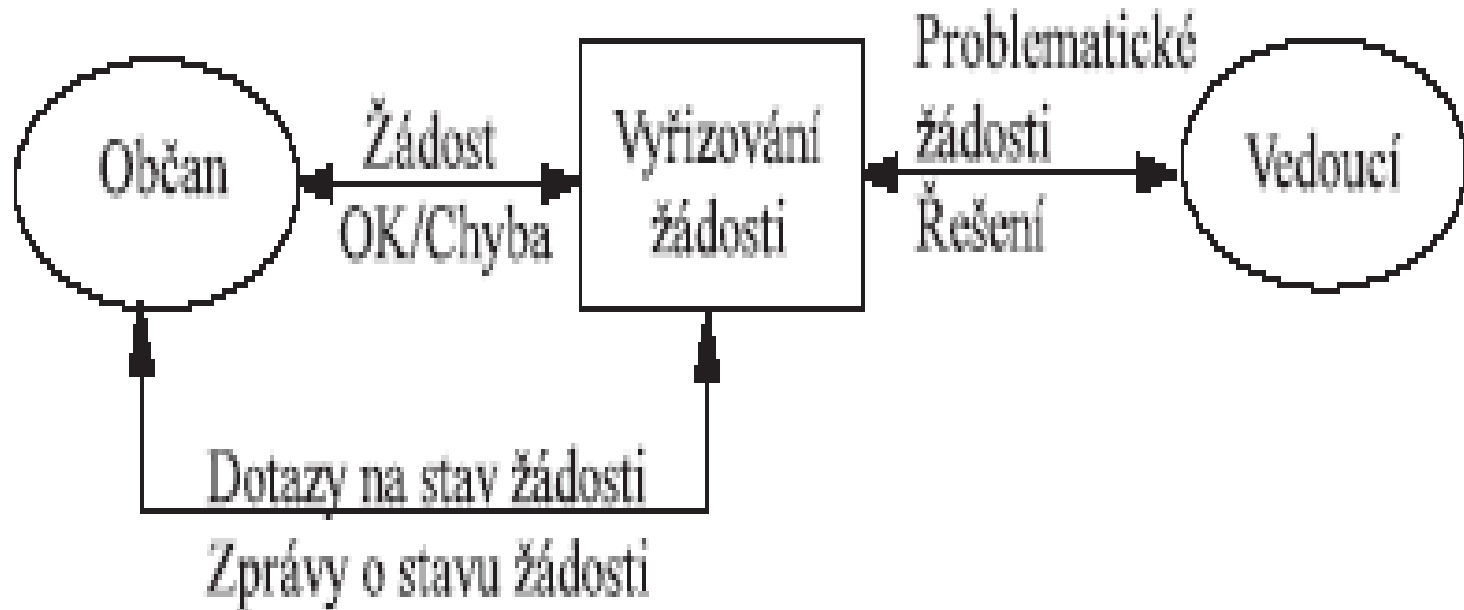
# Odvozená hierarchická dekompozice



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

# Diagram kontextu.

## Dají se použít Use Case



Obr. 12.12: Diagram kontextu systému Vyřizování žádosti.

# Výhody a nevýhody

- Vhodnější pro dávkové zpracování, tam ale významem obdoba SOA pro interaktivní spolupráci
- Pokud je možné použít, může podstatně usnadnit vývoj a modifikace využitím úložišť
- Nevýhoda je, že může omezit použití on-line operací
- Je možná kombinace úložišť a komunikace výměnou zpráv, to je zvláště vhodné pro manažery, viz Generalized Petri Places

# Rozhodovací tabulky

- Umožňuje přehledně zapsat, za jakých podmínek učinit příslušnou akci/akce
- Do horního pole se zapisují požadované pravdivostní hodnoty jednotlivých podmínek (ano A, ne N, na podmínce nezáleží X)
- Do dolního pole se zapisuje značkou x, zda se má příslušná akce pro kombinaci podmínek uvedenou v horní části sloupce provést

# Rozhodovací tabulky

Starý zákazník	A	A	A	A	A	...	N	N
Běžný leasing	A	A	A	A	N		x	x
Rušení nájmu	A	A	N	N	A		x	x
Nový nájem	A	N	A	N	A		A	N
Zařadit zákazníka							x	x
Test platby	x	x	x	x	x			
Zrušení smlouvy	x	x						
Nová smlouva	x		x		x		x	
Faktura	x	x	x	x	x			
Úprava smlouvy	x	x	x	x				



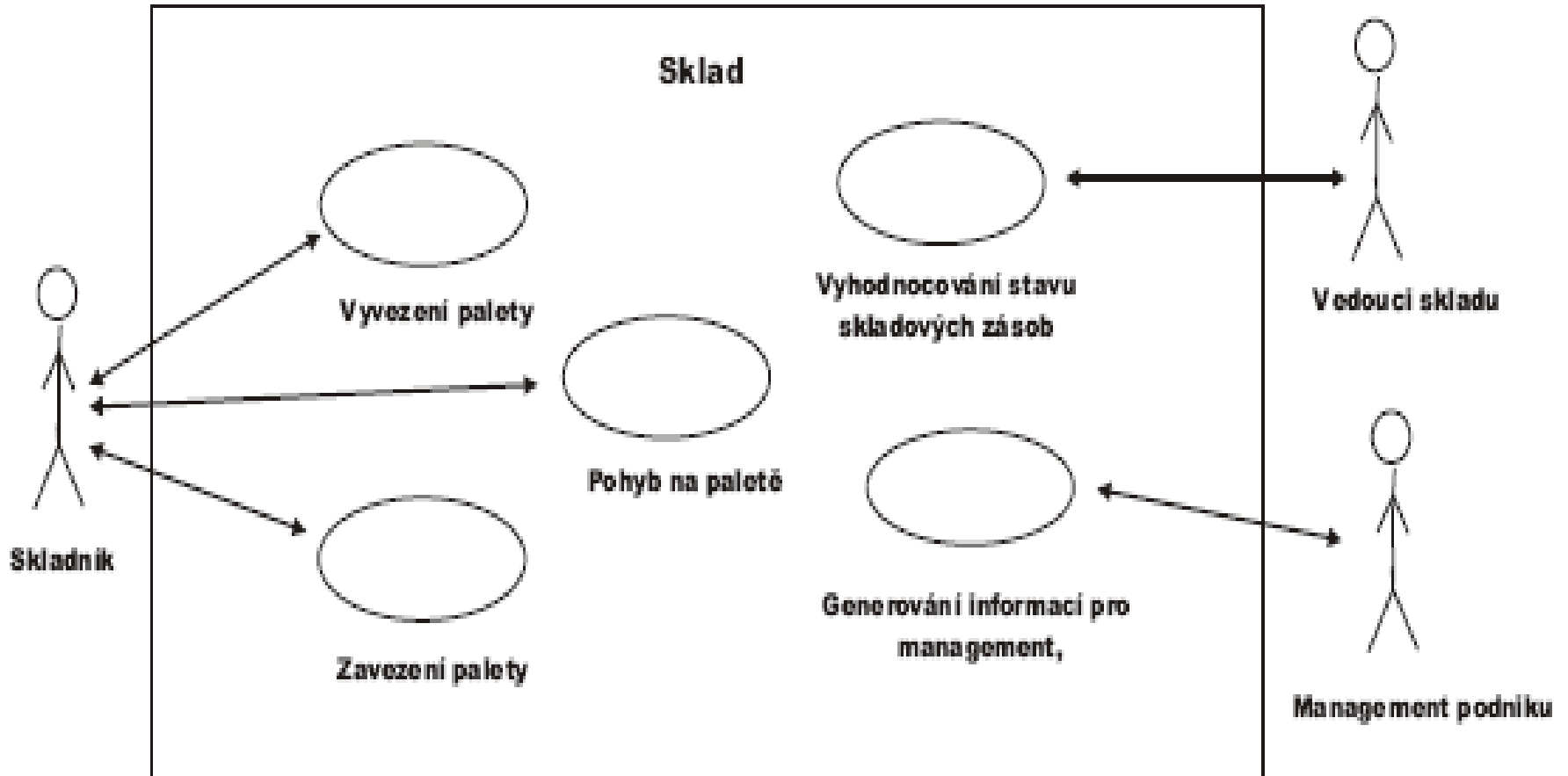
# Rozhodovací tabulky

- Vhodné spíše pro dávky a menší úlohy
- Osvědčuje se pro vyjasnění všech možností
- Podmínek nesmí být příliš mnoho
- Spíše jen okrajová metoda
- Použitelné při specifikaci požadavků i při návrhu systému
- Používá se v podnicích

# Modely pro specifikaci požadavků

- Aktor
- Příklad použití

# Případy použití



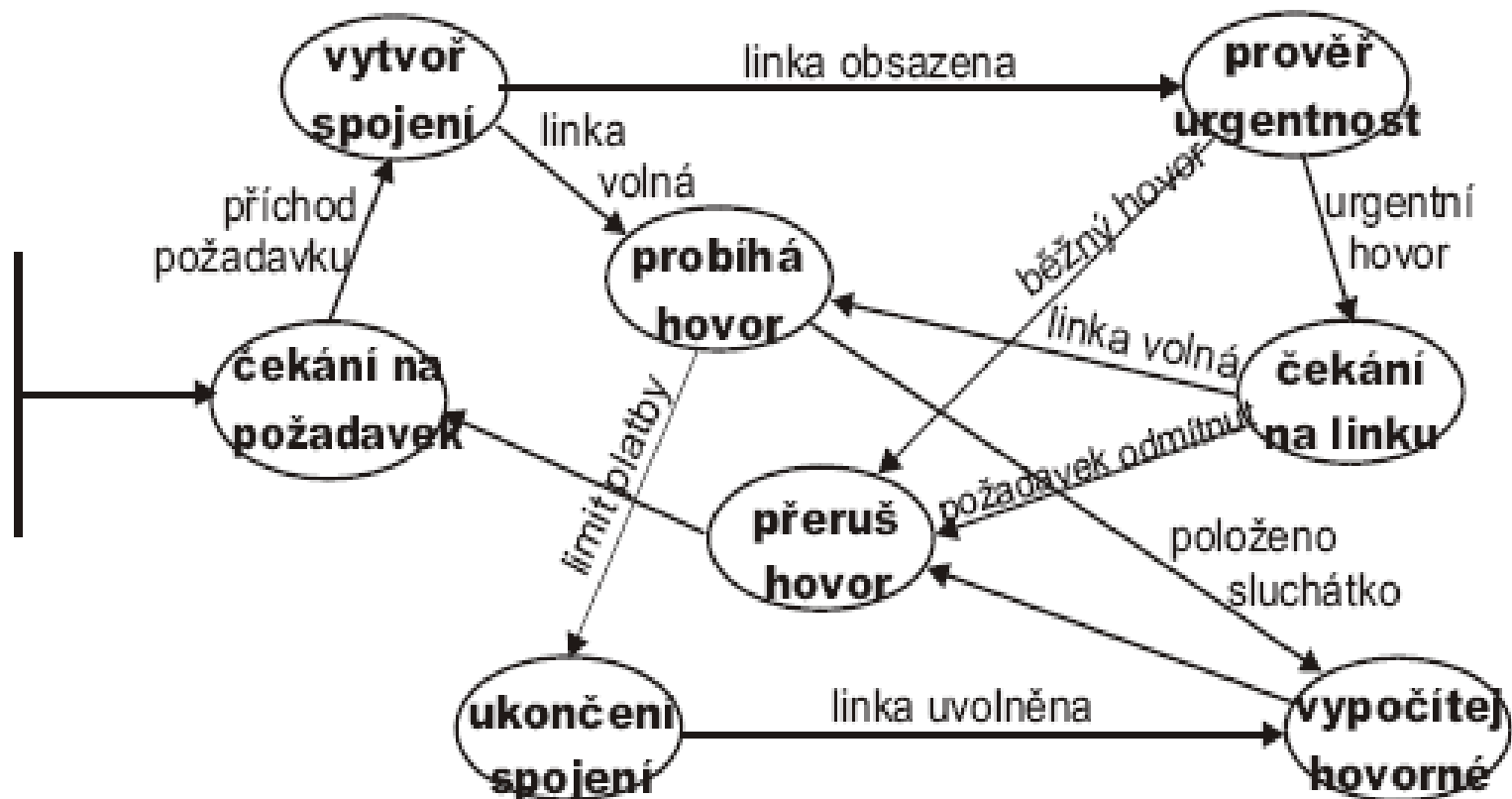
# Případy použití

- Lze použít pro jednotlivé služby (mají-li uživatelsky orientované rozhraní, to by ale mělo být pravidlem, neboť jinak nemá SOA žádoucí vlastnosti) i pro celé SOA
- Měly by být specifikovány v jazyce uživatele a pak zpřesněny pomocí diagramů, k diagramům přecházet až když jazyk uživatele není dostatečně přesný
- Je výhodné použít pojmy z objektové orientace

# Případy použití

- Mezi aktory může existovat vztah dědění
  - vedoucí skladu dědí akce skladníka
- Mezi případy použití jsou vztahy
  - používá
  - rozšiřuje (podobné vztahu dědění)

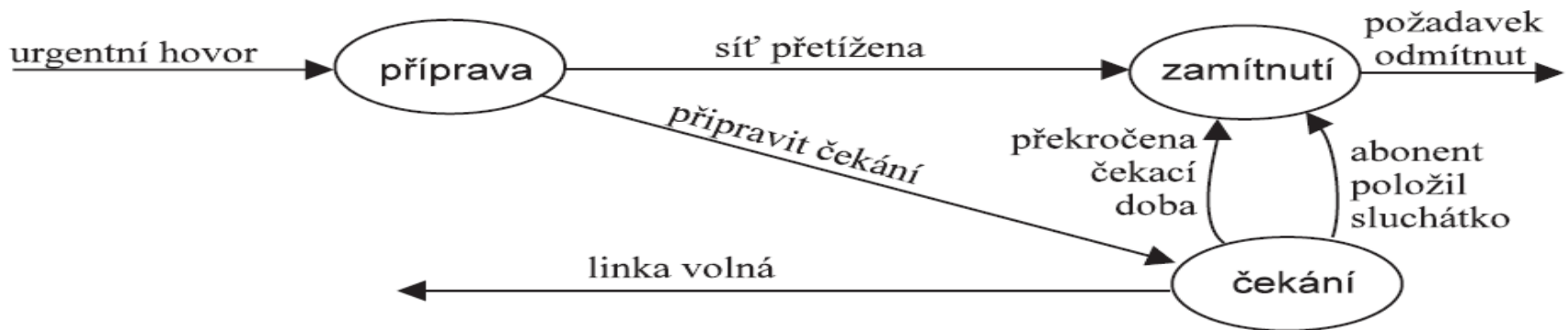
# Pracovní tok



# Pracovní tok

- Vhodné pro popis návaznosti činností, spíše na úrovni systému
- Blízké k pracovním tokům ve smyslu podnikových procesů
- Rozsáhle zobecněno v UML a v různých variantách specifikace pracovních toků. Zobecnění hlavně v oblasti paralelity (souběh, vzájemné vyloučení, následnost)

# Pracovní tok





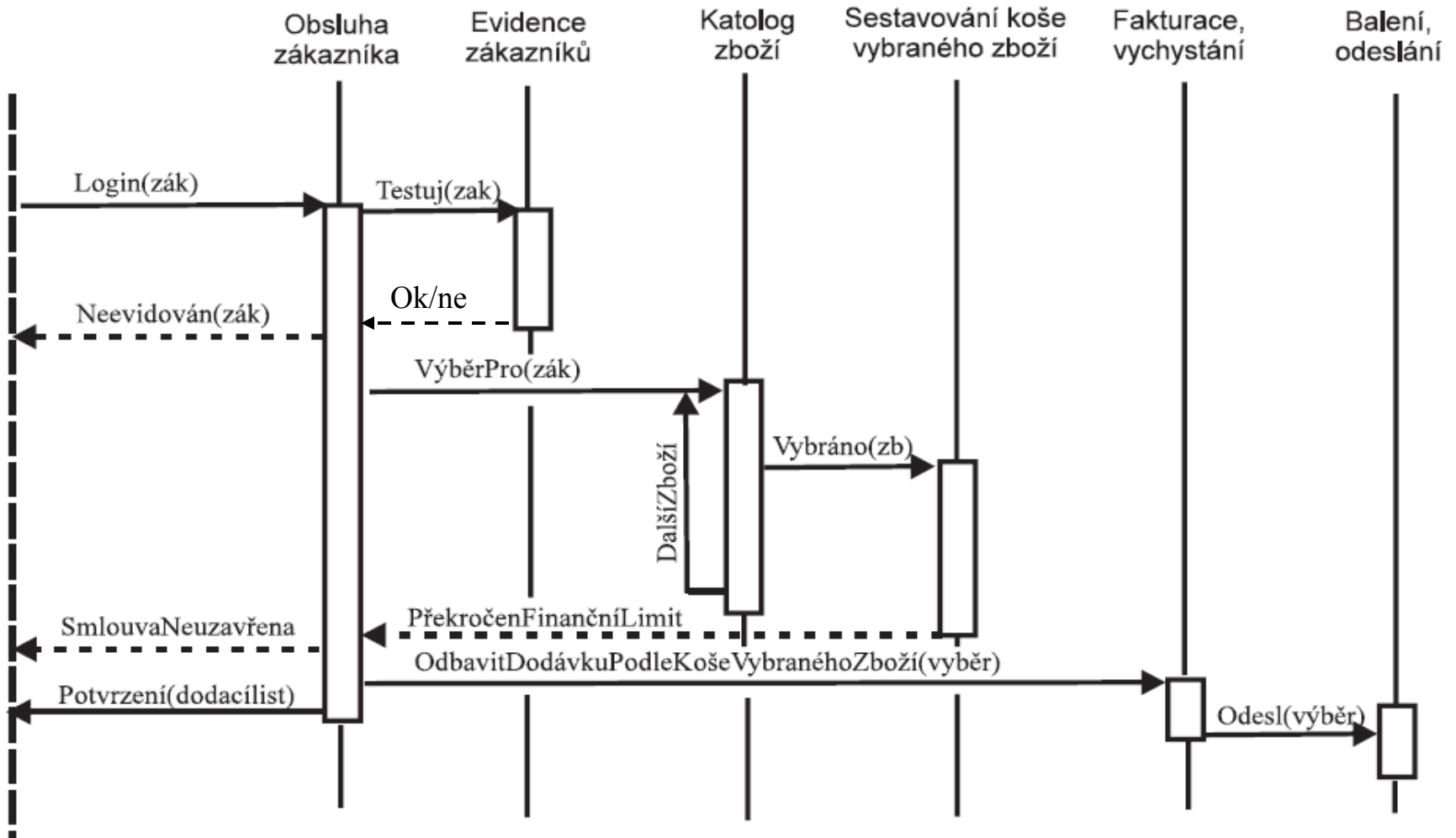
# Pracovní tok, scénář



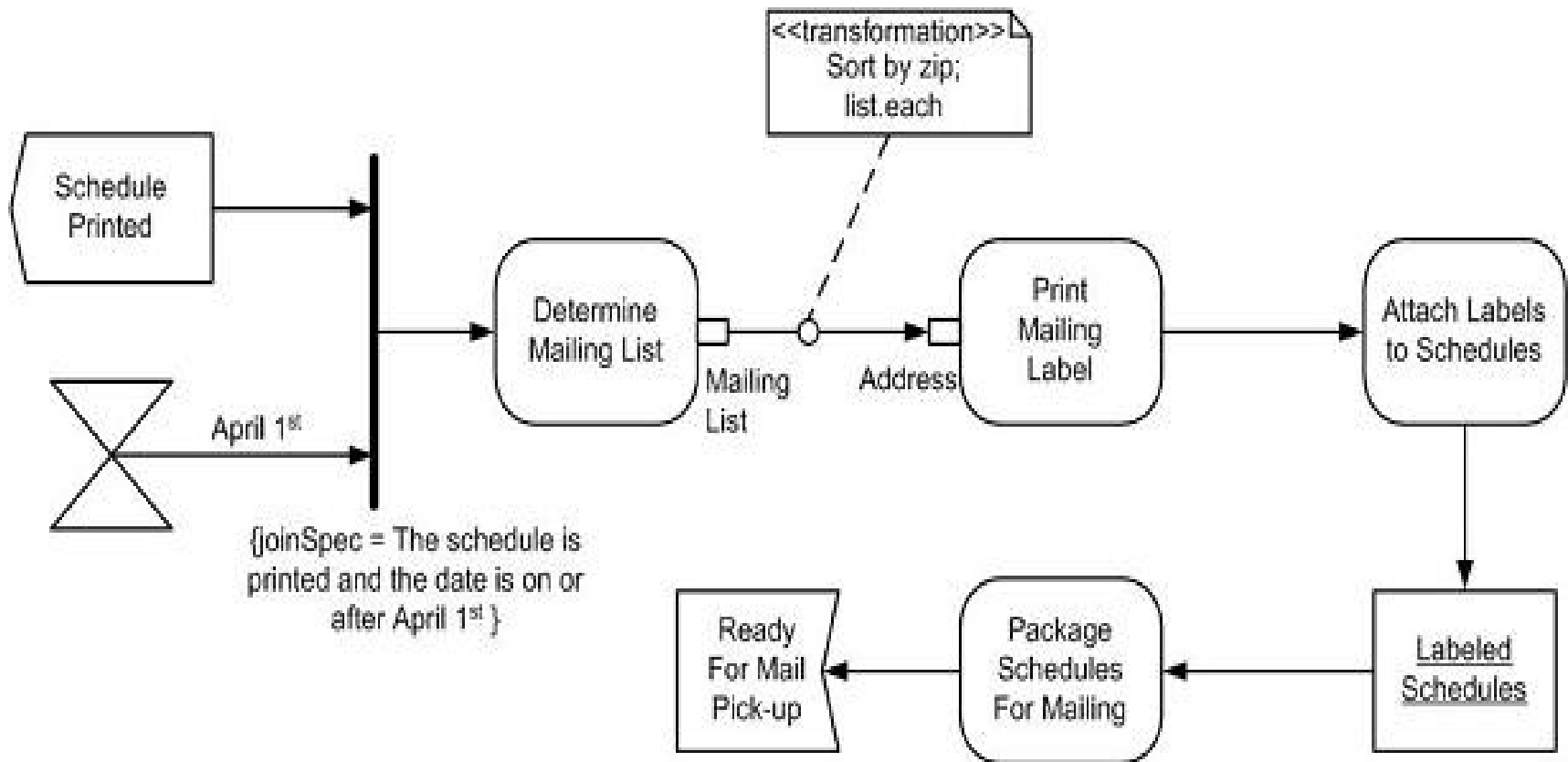
# Pracovní tok, spolupráce komponent

- Vhodné pro popis systému dekomponovaného do komponent spolupracujících prostřednictvím výměny zpráv
- Vhodné i pro vývoj SOA systémů
- Vhodné pro scénáře spolupráce lidí i SW komponent
- Existuje několik variant, včetně té, která místo komponent používá jednotlivé třídy

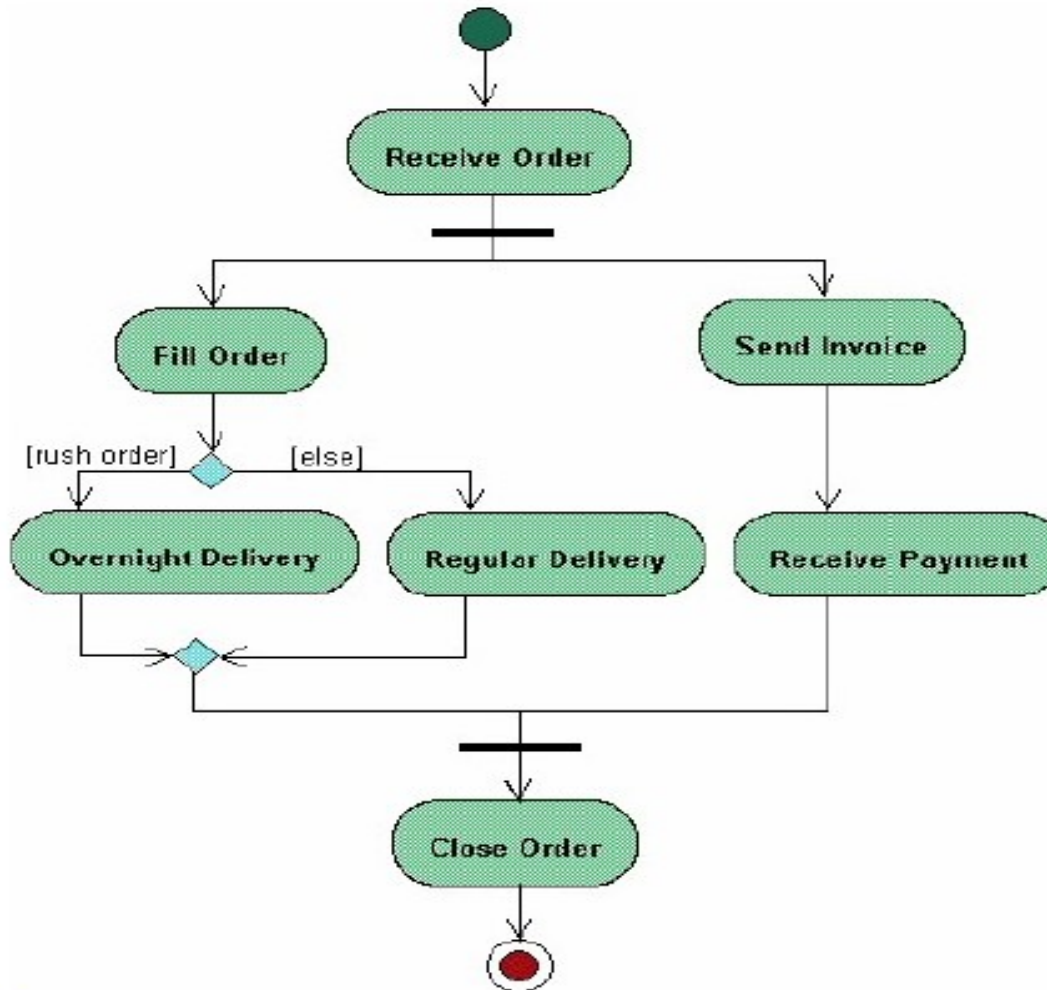
# Pracovní tok, UML



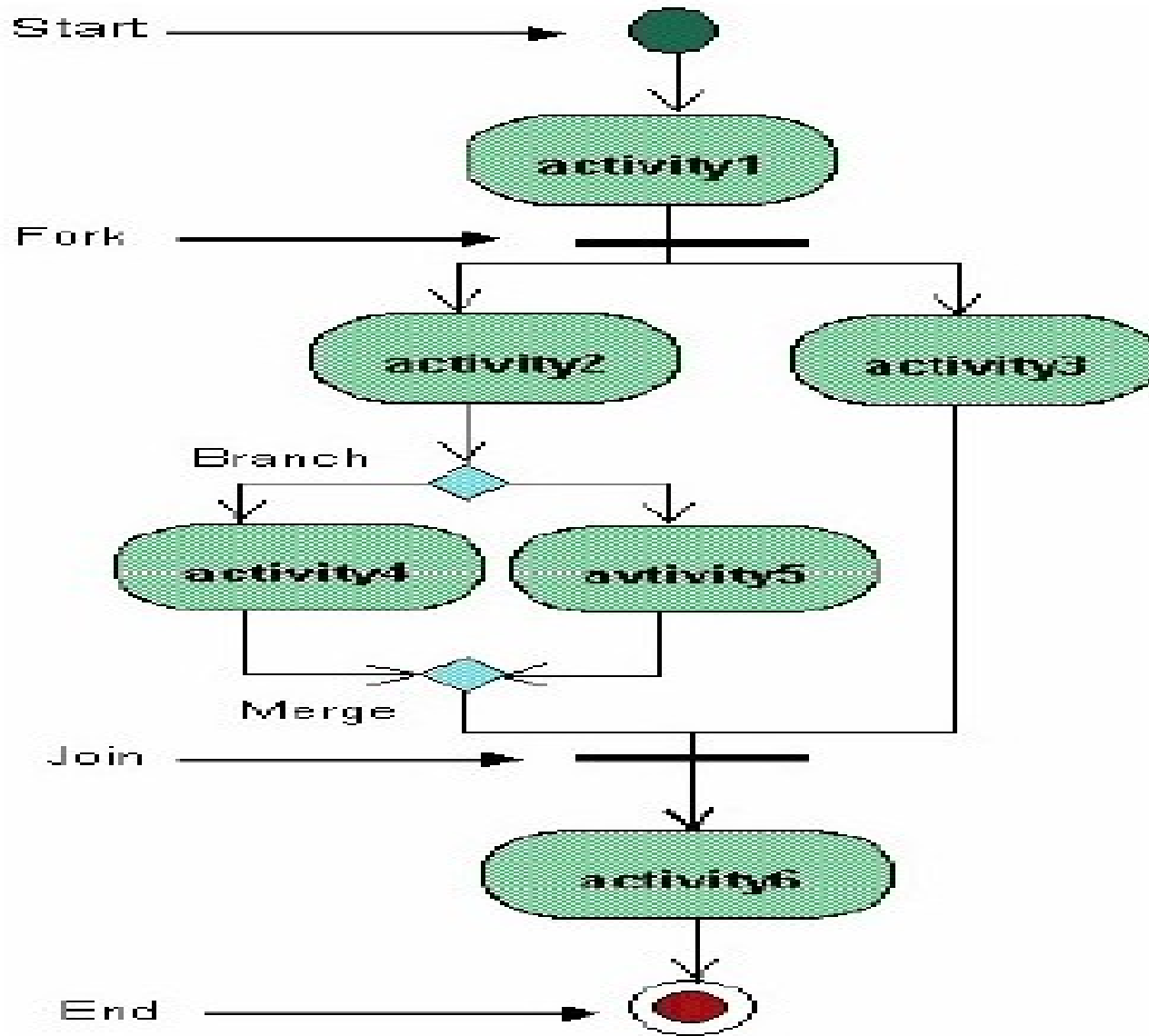
# Příklad diagramu aktivit



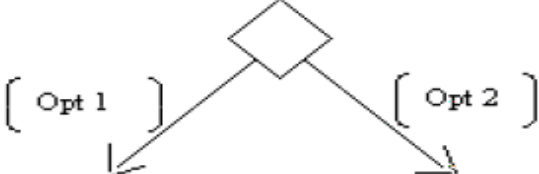
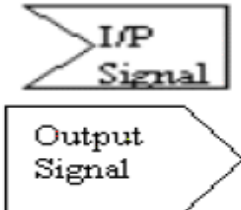
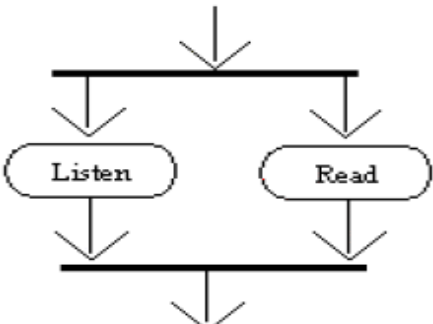



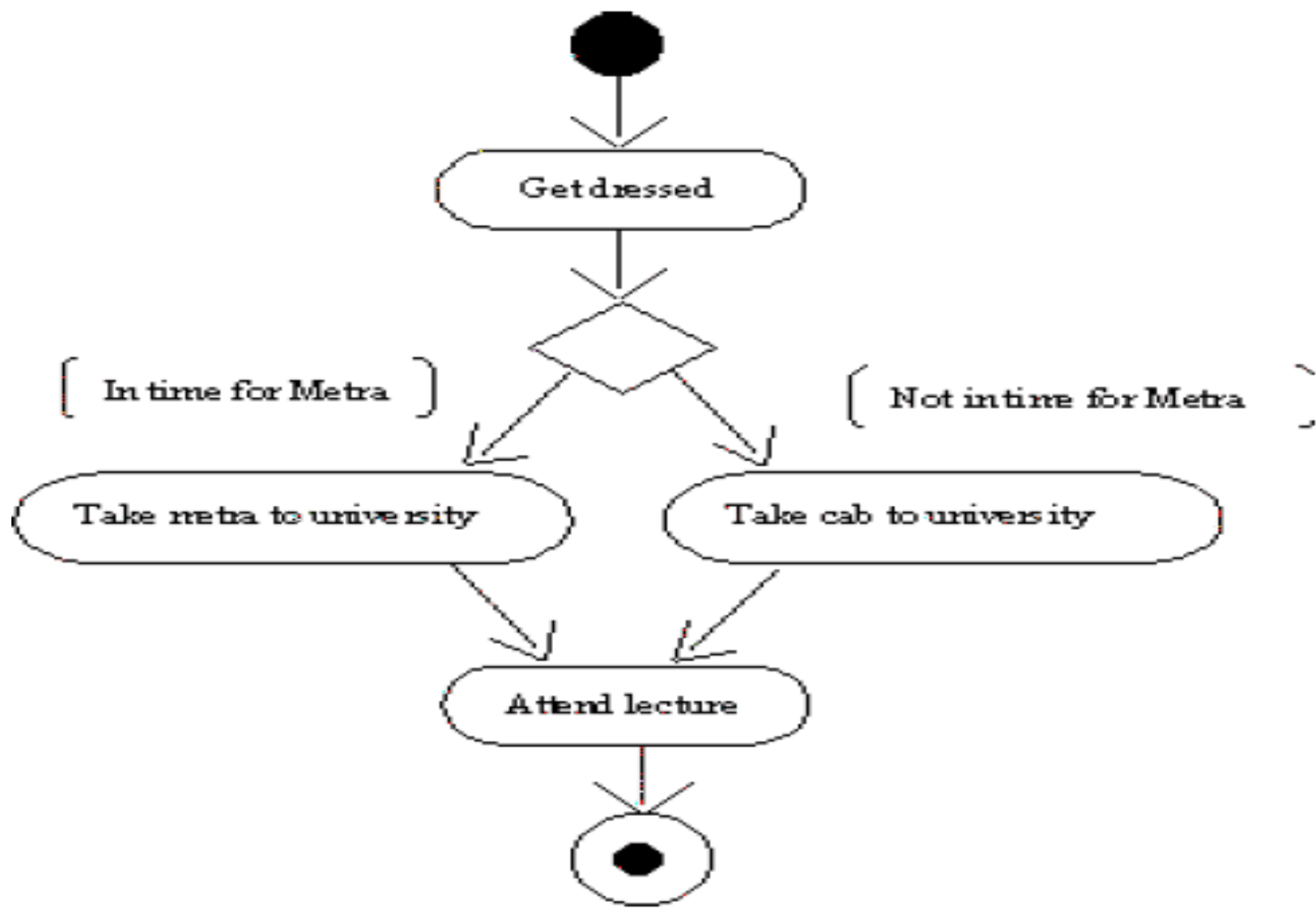
# Diagramy aktivít



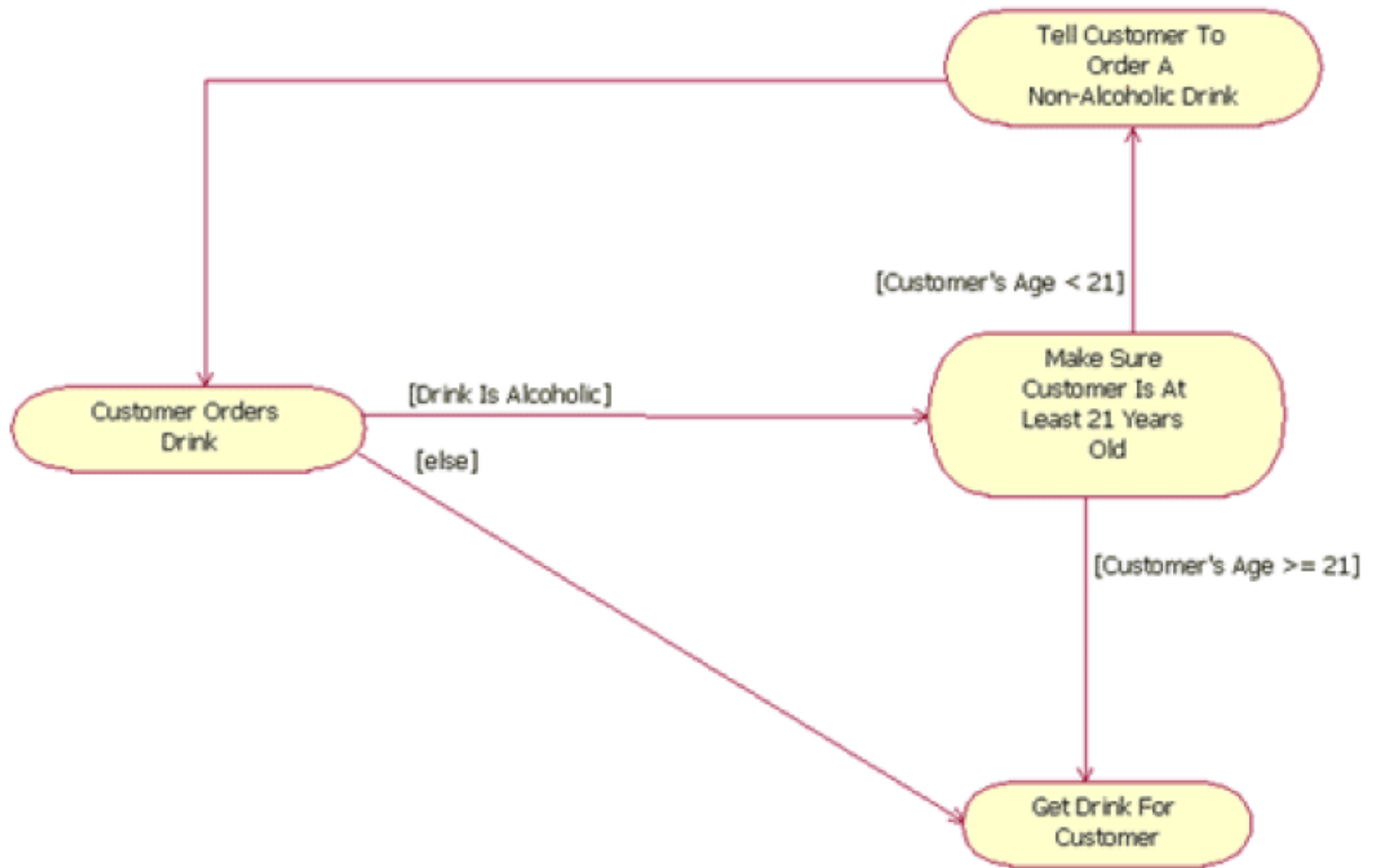
# Diagramy aktivit

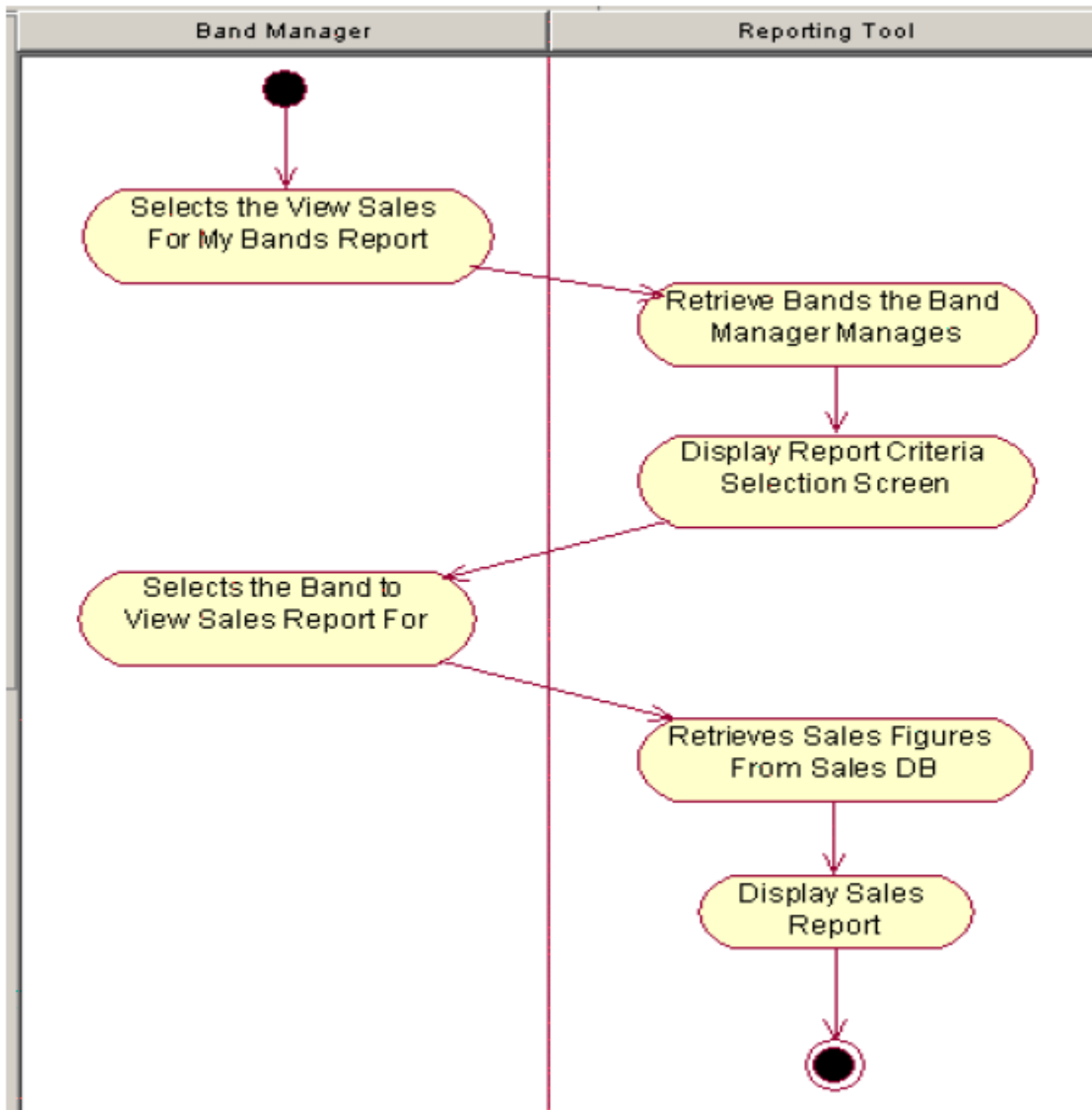


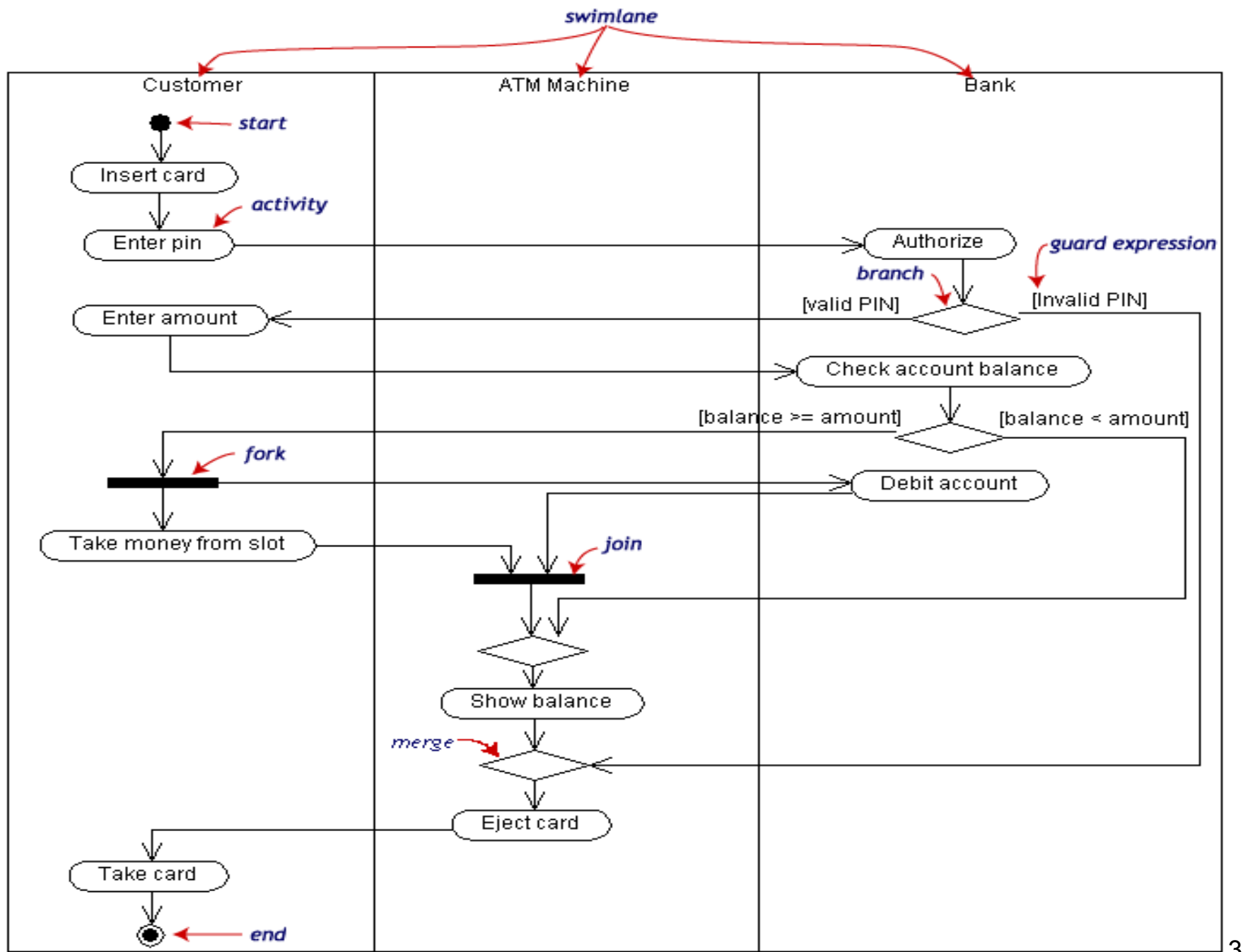
Element and its description	Symbol
<p><b>Initial Activity:</b> This shows the starting point or first activity of the flow. Denoted by a solid circle. This is similar to the notation used for Initial State.</p>	
<p><b>Activity:</b> Represented by a rectangle with rounded (almost oval) edges.</p>	
<p><b>Decisions:</b> Similar to flowcharts, a logic where a decision is to be made is depicted by a diamond, with the options written on either sides of the arrows emerging from the diamond, within box brackets.</p>	
<p><b>Signal:</b> When an activity sends or receives a message, that activity is called a signal. Signals are of two types: Input signal (Message receiving activity) shown by a concave polygon and Output signal (Message sending activity) shown by a convex polygon.</p>	
<p><b>Concurrent Activities:</b> Some activities occur simultaneously or in parallel. Such activities are called concurrent activities. For example, listening to the lecturer and looking at the blackboard is a parallel activity. This is represented by a horizontal split (thick dark line) and the two concurrent activities next to each other, and the horizontal line again to show the end of the parallel activity.</p>	
<p><b>Final Activity:</b> The end of the Activity diagram is shown by a bull's eye symbol, also called as a final activity.</p>	







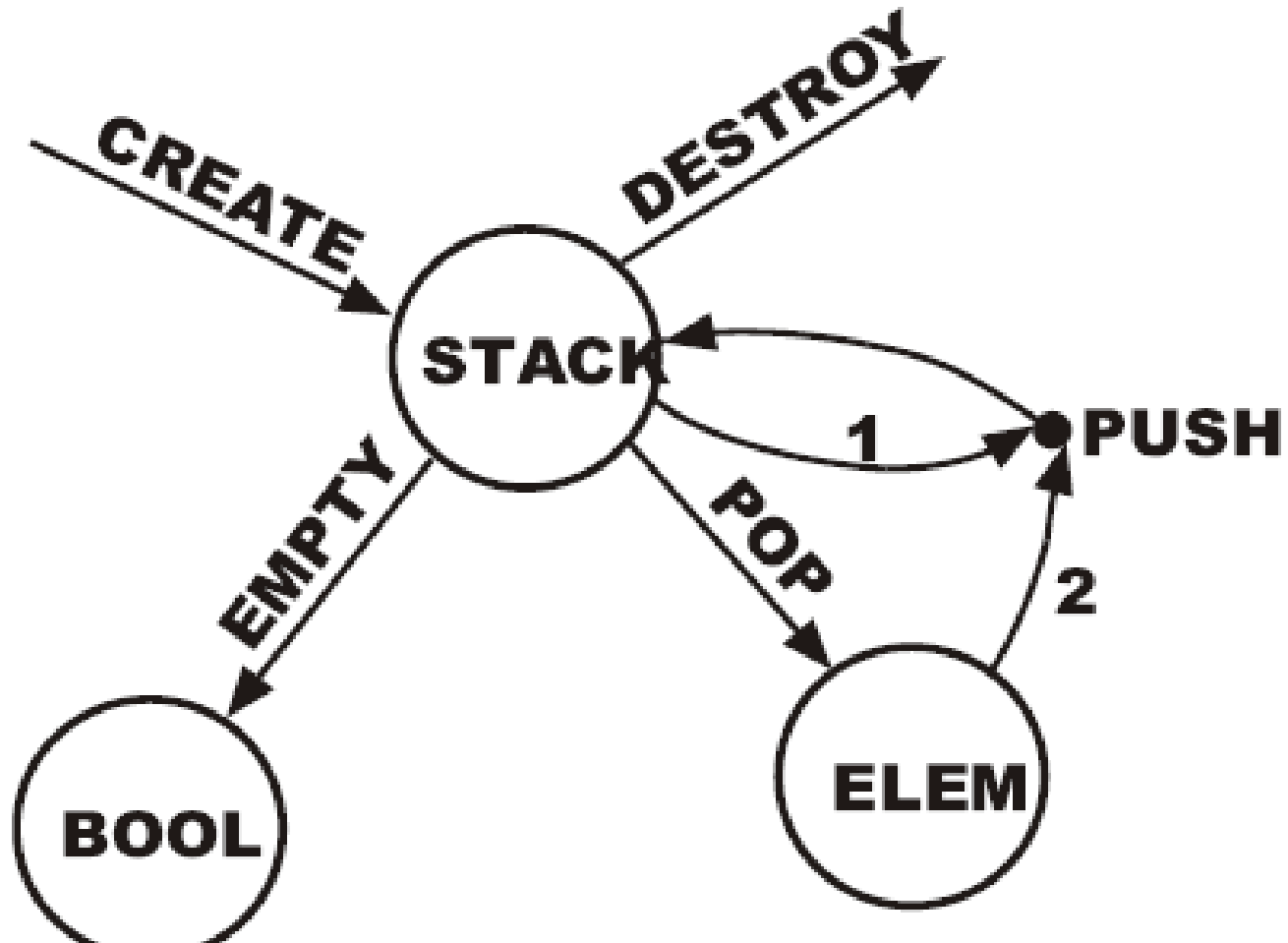




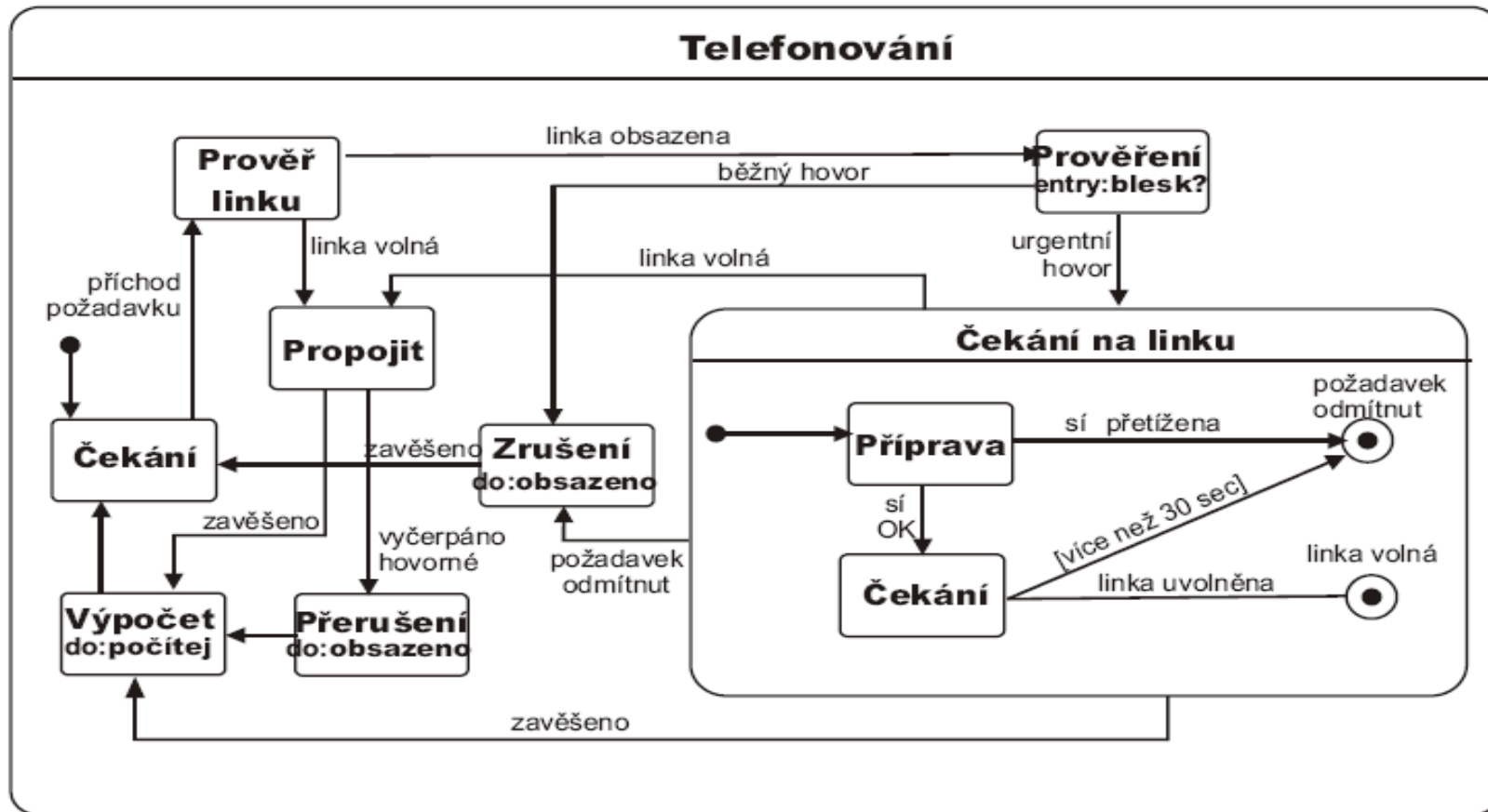
# Diagram aktivit

- Diagram aktivit je prostředkem definování částečně paralelních aktivit v OO
- Všimněme si rozdílu přístupu SOA a OO
  - SOA – služby běží paralelně a jejich synchronizaci je nutné nějak zařídit
  - v OO je naopak nutno programovat paralelitu

# Signature



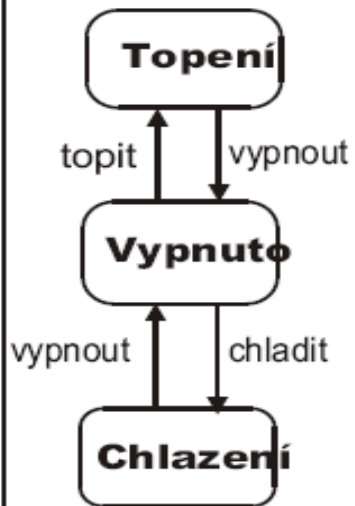
# Koordinace činností (UML)



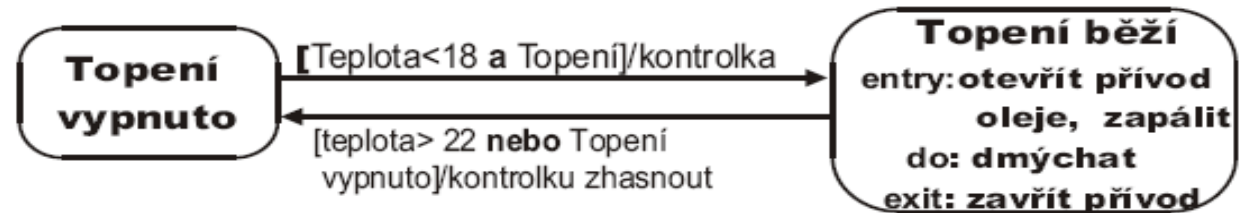
# Koordinace činností (UML)

## Programovatelná klimatizace

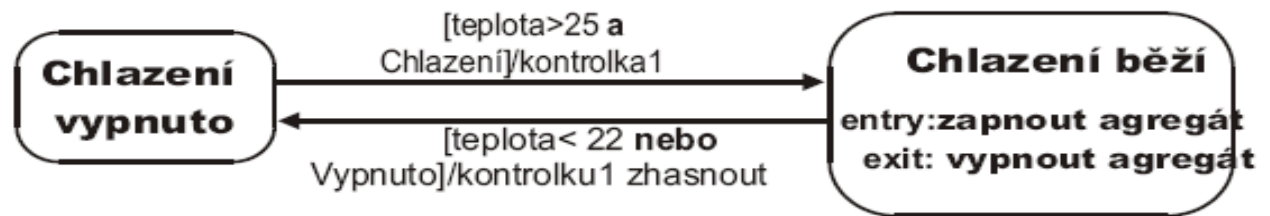
### Hlavní vypínač



### Ovládání topení



### Ovládání chlazení



# SADT

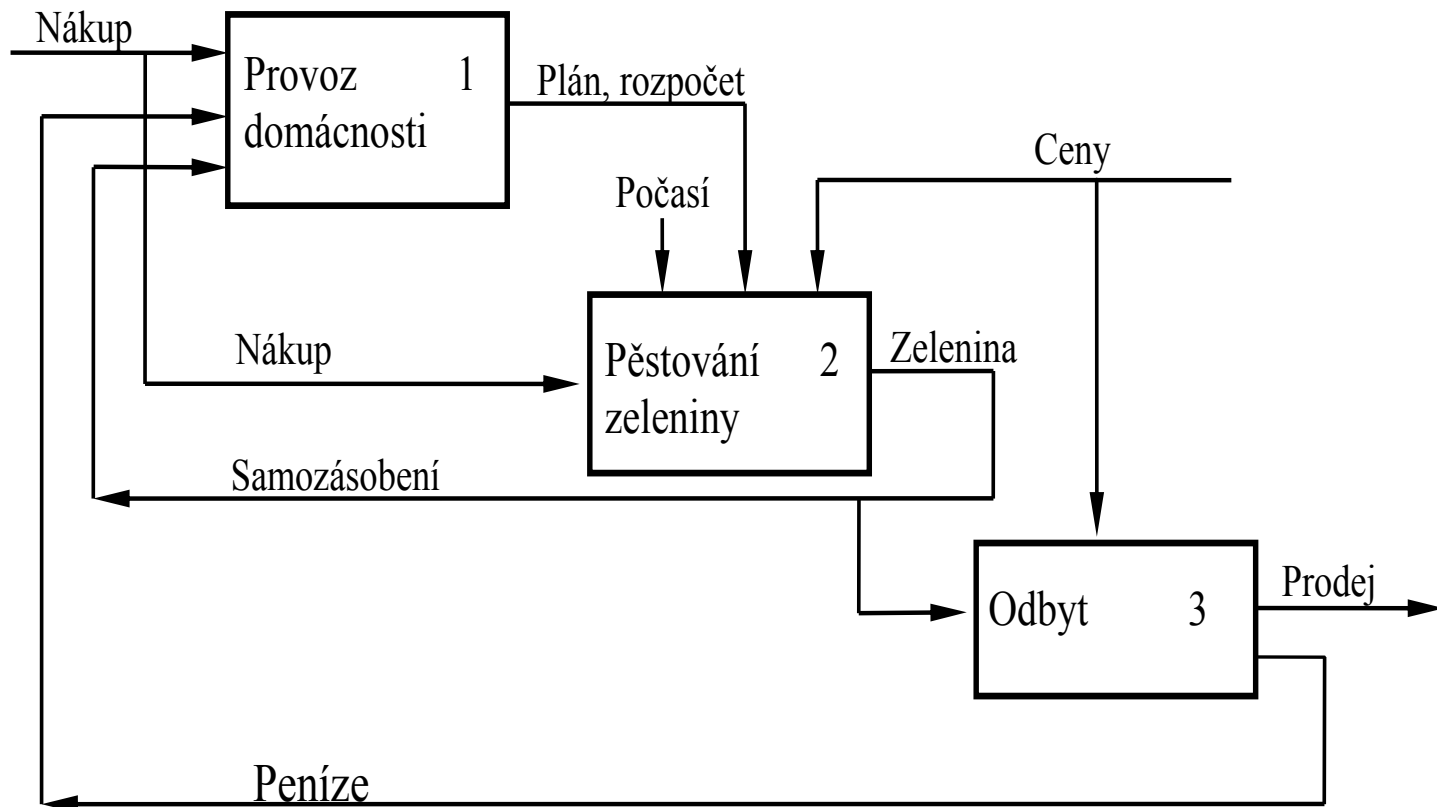
## Structured Analysis and Design Technique

- Starší systém, spíše pro dávky
- Vychází z IDEF norem živých v průmyslu při specifikaci podnikových procesů
- Dnes zřídka používaný



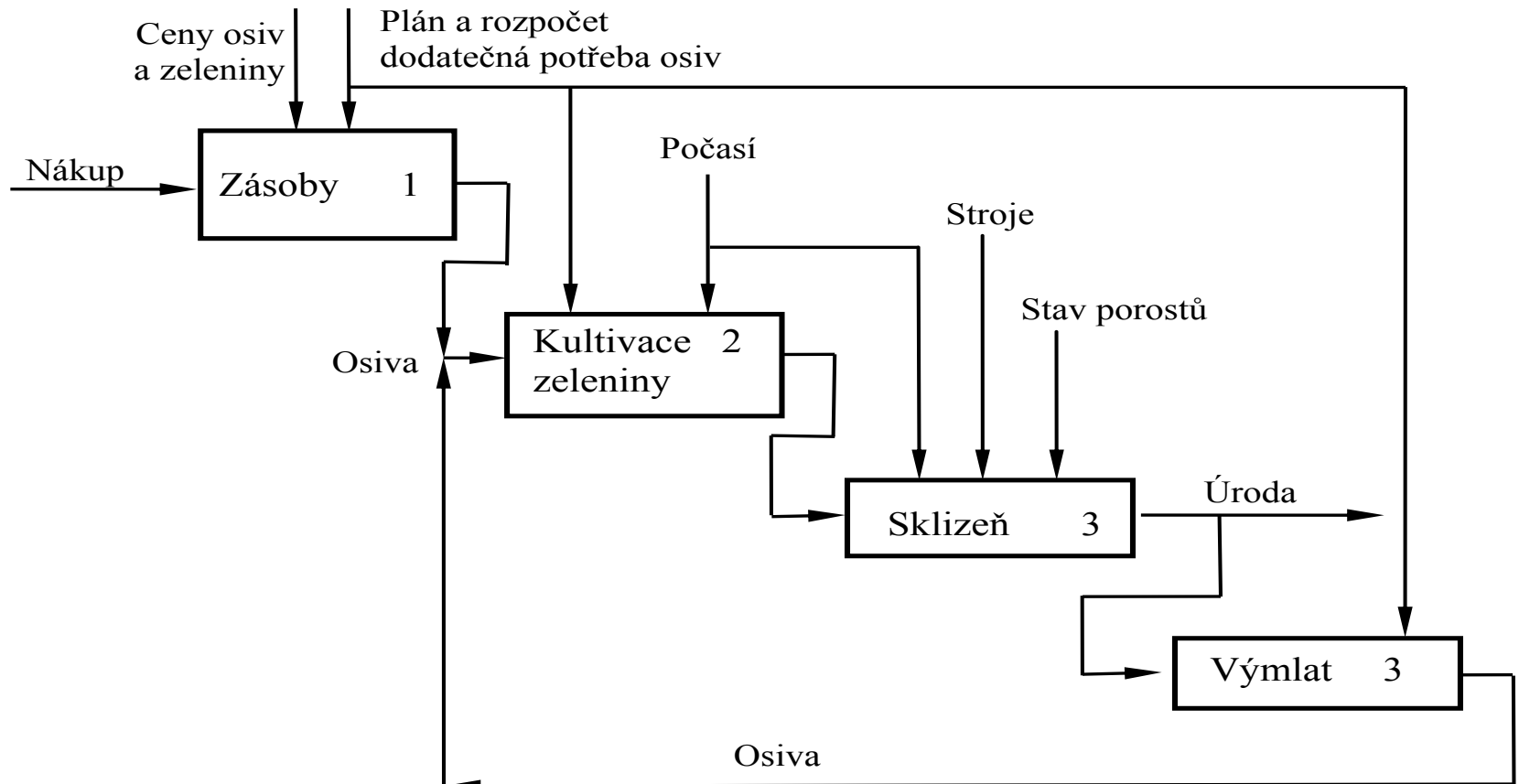
# SADT (IDEF0 až IDEF3)

DIAGRAM AKCÍ A0: *Zahradnictví.*



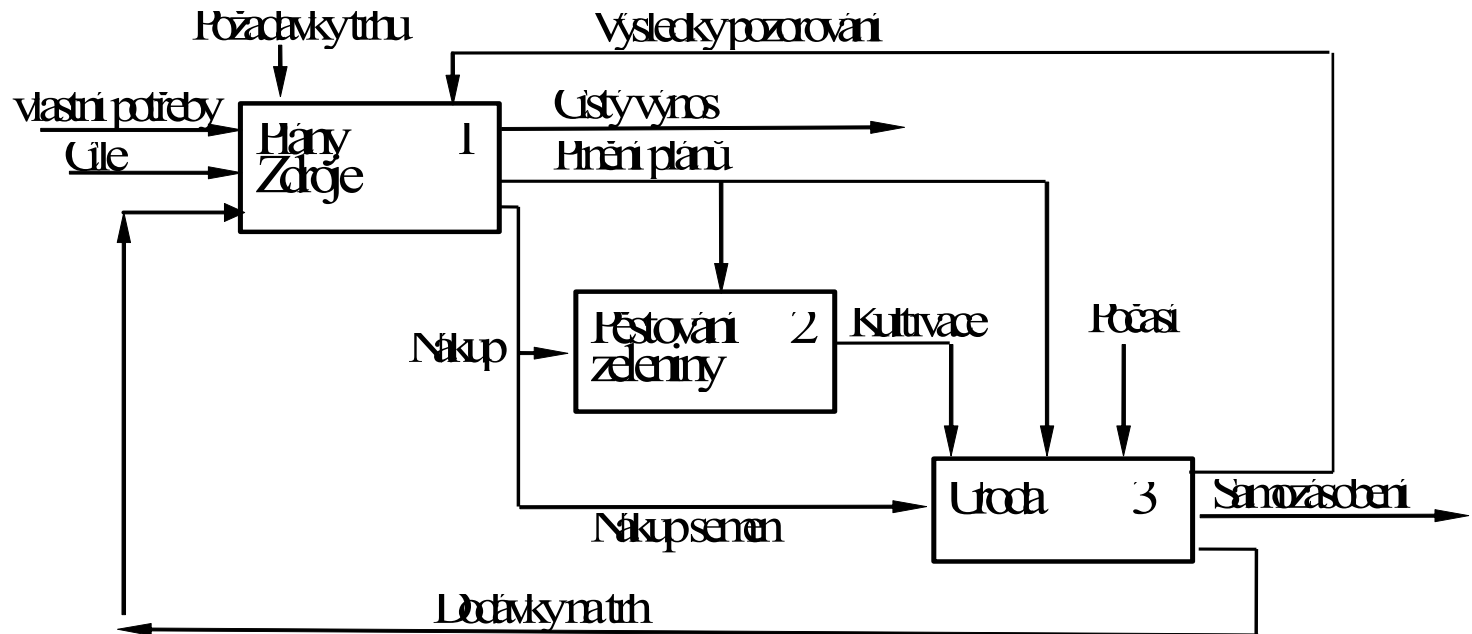
# SADT (IDEF0 až IDEF3)

DIAGRAM AKCÍ A0.1 *Pěstování zeleniny.*



# SADT (IDEF0 až IDEF3)

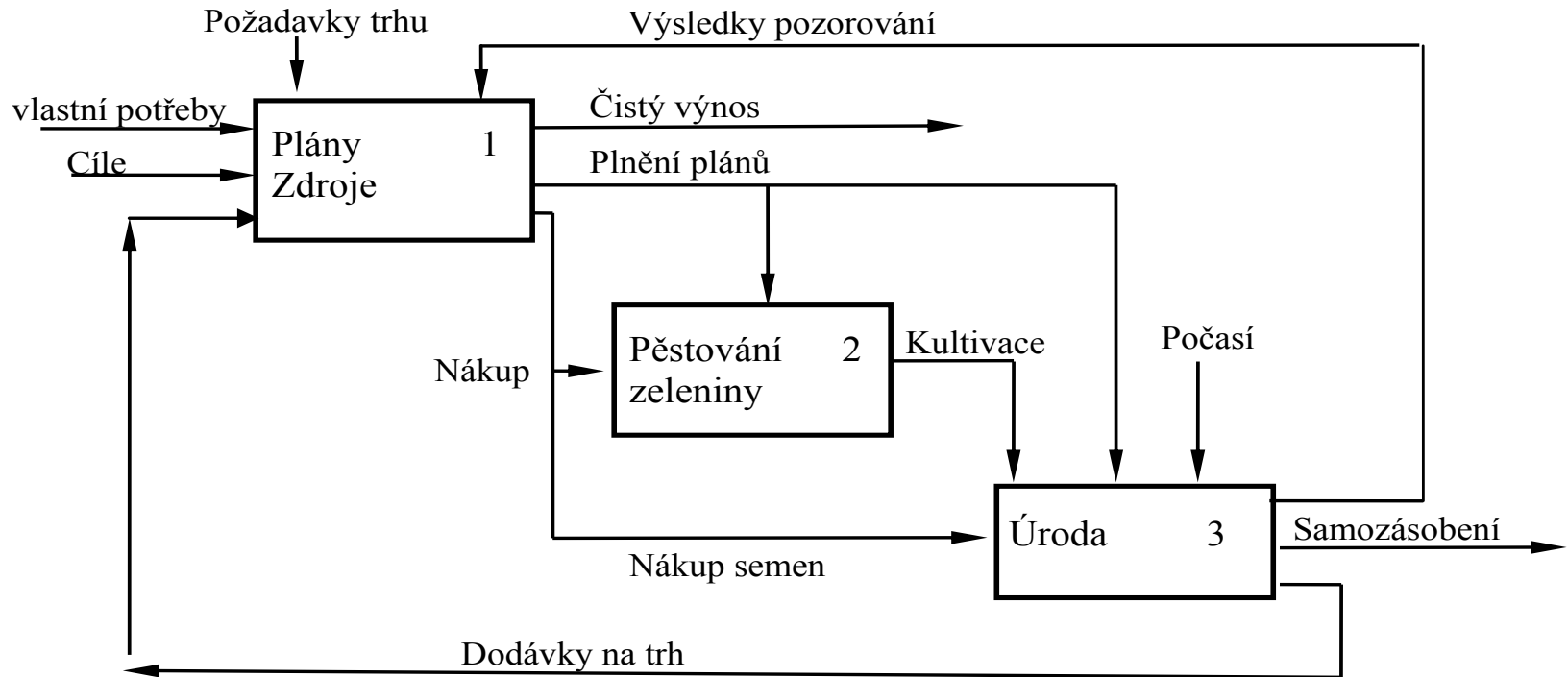
DIAGRAM DATU: Zahrádkovi.



Úr. 1273  
dbc

# SADT (IDEF0 až IDEF3)

DIAGRAM DAT D0: *Zahradnictví.*



Obr. 12.7.3.  
doc

# Kolik investovat do nástrojů

- Kolik dávat do „neproduktivních“ činností, takových, jejichž výstup není částí projektu
  - HW
  - Podpůrný SW
  - Nástroje
    - Kupované
    - Vlastní
  - Vzdělávání lidí



# Etapy zvládnání nové metodiky

- Líbánky
  - Metodika se používá v oblastech, pro které byla především vyvinuta, **tam** se osvědčuje
  - Pracují s ní kvalitnější lidé, kteří se snaží touto cestou získat slávu, nejsou komerční nástroje velkých výrobců
- Vystřízlivění
  - Nehodí se na vše, výrobci SW jsou stále opatrní, náraz na meze
  - Má mouchy (implemetační, metodické)
  - Vužaduje zaškolení a změnu návyků (to někdy až při změně generace vývojářů, viz objektovou orientaci)
  - Inovátoři se zajímají o nové hity
- Soužití (ne vždy k němu dojde)
  - Většina nedostatků se odstraní
  - Systém používají stálí uživatelé a ti si s problémy nějak poradí, nebo si na ně zvyknou. Přibývá komerční podpora od velkých firem a výuka na školách
  - Časem dojde k ustálenému stavu (plató)

# Modernost metodiky

- Nezapomínat na rozumnou opatrnost
- Ale také nezapomínat, že risk je zisk
- Nové zkoušet na menších projektech a nasadit na to kvalitní pracovníky
- **To platí i pro každé nové nástroje a programovací jazyky**



# Řízení konfigurace

Dosáhnout toho, aby byly vyvinuty, prověřeny a do celku se dostaly komponenty (prvky konfigurace), které pro danou verzi výrobku patří k sobě

Obecně technický problém

ISO10007 – obecně řízení konfigurace

ISO 12207, ISO 90003, ISO 15846, ISO 20000 – software process

Jsou normy i pro Plán řízení konfigurace

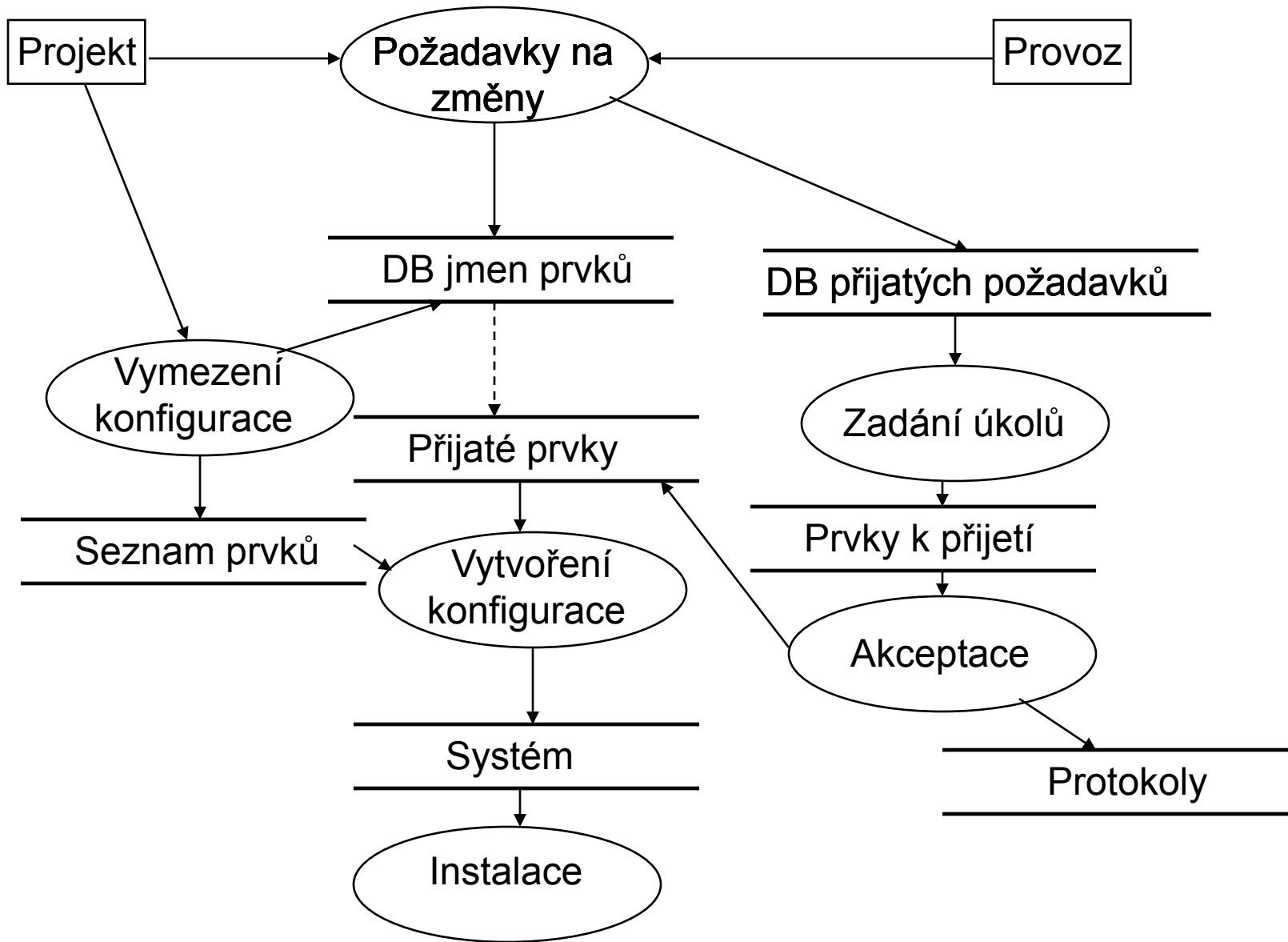
# Řízení konfigurace

Stanoví se (někdy v etapách) prvky konfigurace, každý prvek má jméno a id, který určuje, do které konfigurace patří

Prvky projdou cyklem od zadání k převzetí

Správa konfigurace hlídá, zda jsou prvky k dispozici, tj. zda prošly i řádným vývojem a byly prověřeny

Do konfigurace (verze) x.y.z.w patří prvky se jménem v seznamu a z těch s id mající nejdelší společný prefix s id konfigurace



# Kódování

Psaní programu podle přesných  
specifikací

Člověk jako kompilátor

# Kódování

- Kódování není dnes hlavní problém
- Jedna sedmina pracnosti vývoje a pracnost kódování se dále zmenšuje
  - Ví se, jak na věc a jak to učit a standardizovat
  - Systémy podpory programování (vizuálnost, CASE, Delphi, Power Builder, ...), asi nevhodné pro SOA
  - Servisní orientace, objektová orientace, komponent
  - Problém je ve specifikacích, to je úzké místo, kódování nikoliv
- Při kódování je výhoda mládí. Nebezpečí, že se mladí omezí na kódování a nic dlouhodobějšího se nenaučí

# Programovací jazyky

- Jsou méně důležité, než se má za to, úzké hrdlo je jinde (Prof.Malík simuloval lidské srdce ve Fortranu, ač se tento jazyk pro to nehodí, stálo ho to dva měsíce práce, koncepce byla dobrá díky jazyce Simula, jazyka pro programování simulací. Simulátor byl totiž nejprve napsán v jazyce Simula .Simulátor se používal při testech správnosti aplikace kardiostimulátorů a jejich vylepšování)
- Nejen vlastnosti jazyka, ale také vývojového prostředí, včetně knihoven a hotových částí, zásuvné moduly
- Důležité, jak se jazyk uplatní v podpoře SW architektur (COBOL a dataflow diagramy pro dávkové výpočty)
- Nový jazyk se zpravidla uplatní jen, je-li určen pro volnou niku na trhu (Java pro webové stránky, srv. FORTRAN kontra Algol či PL/1)
- Znalosti vývojářů závisí na jazyce a s ním spojenými nástroji a metodikami

# Testování

- Součást evaluace (ověřování), zda produkt odpovídá potřebám uživatelů
- Nejpracnější etapa vývoje
- Jde automatizovat jen zčásti. Důvody:
  - Testuje se i správnost specifikace a ta je fuzzy a mění se v čase, blokové znalosti
  - Nutné testovat předpoklady o schopnostech a potřebách uživatelů
- To vyžaduje spoluúčast uživatelů

# Testování

Mnohé problémy lze automatizovat jen zčásti i v případě dokonalých specifikací, jaké bývají u kritických aplikací.

Důvody: Churchova téze, co nejde algoritmizovat na Turingově stroji, nelze vůbec, složitost reálného světa – ten není počítačem, je náhodný v principu (kvantová mechanika)

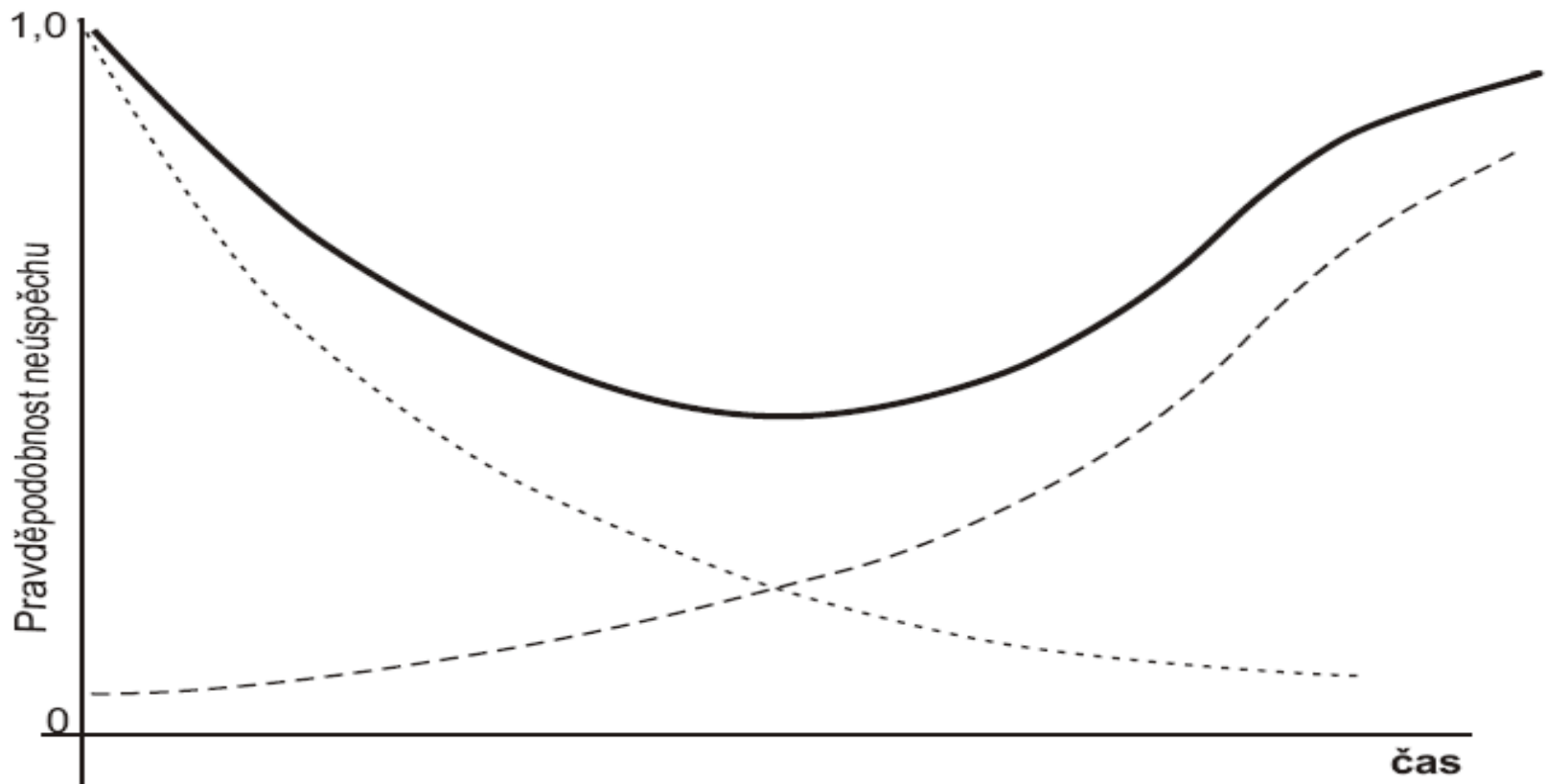
- Algoritmicky nerozhodnutelné problémy při testování, na příklad
  - Detekce mrtvého kódu
  - Otestování všech kombinací návazností větví programu
  - Detekce nekonečného cyklu
- Důsledek: Nelze plně automatizovat, tvůrčí problém,
  - Řeší se heuristicky, vždy ale nějaký problém zůstane



# Testování

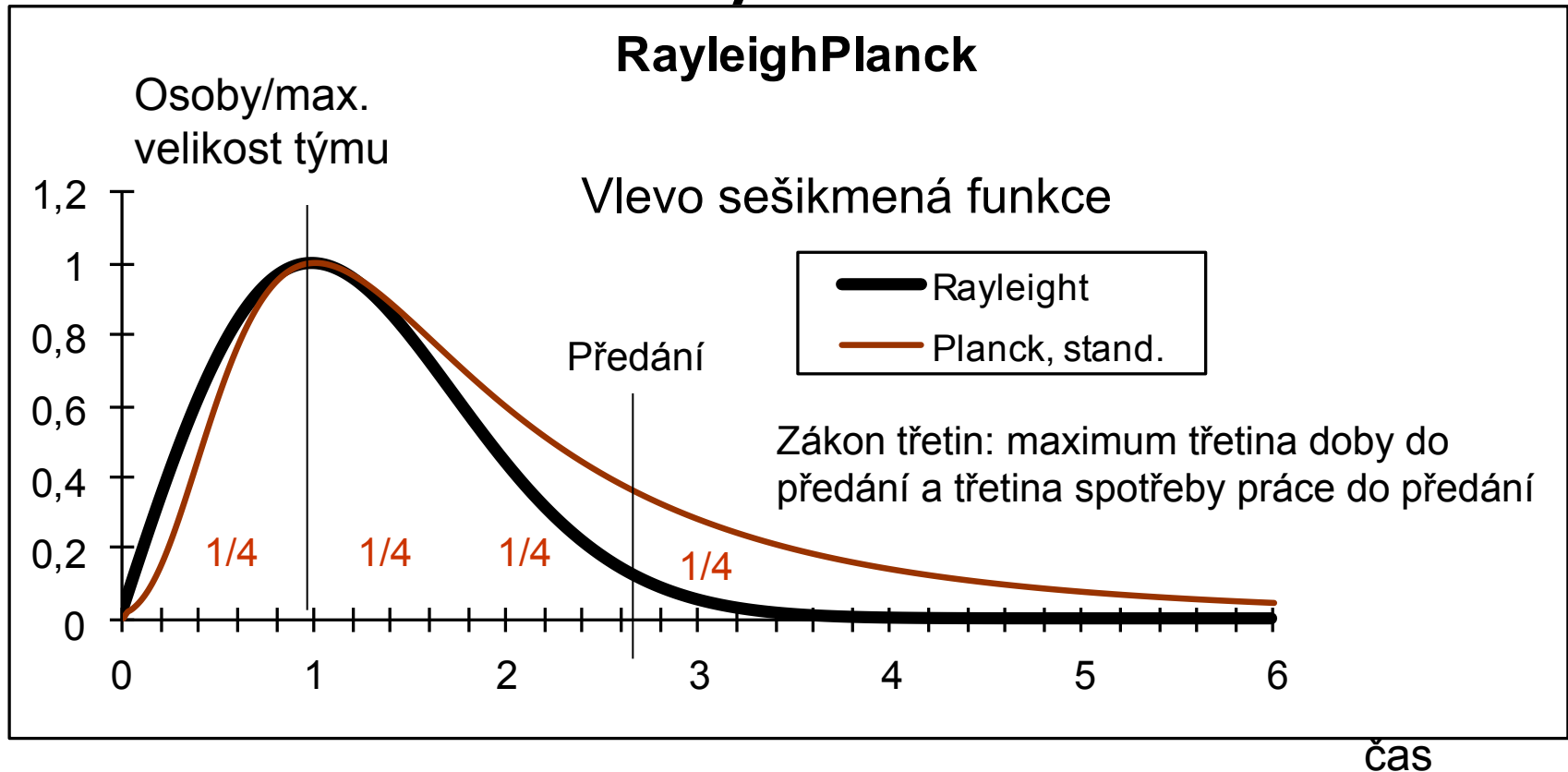
- Součást evaluace (ověřování), zda produkt odpovídá potřebám uživatelů
- Většinou zahrnuje i testování s uživatelem
  - To je největší problém
- Výše uvedené problémy indikují, že nelze prakticky nikdy odstranit všechny problémy, nelze zero defect software

# Pravděpodobnost neúspěchu v závislosti na době testování



- celková pravděpodobnost neúspěchu
- - - pravděpodobnost, že dříve obsadí trh konkurence
- ... pravděpodobnost, že produkt neuspěje pro nedostatečnou kvalitu

# Najímaný tým, vrchol a odhad doby řešení



Transformace proměnných tak, aby max bylo v bodě 1 a mělo hodnotu 1 a v nule byla hodnota funkce prakticky nula. U pevného týmu odpovídá intenzitě práce

# Nelze zero defect software

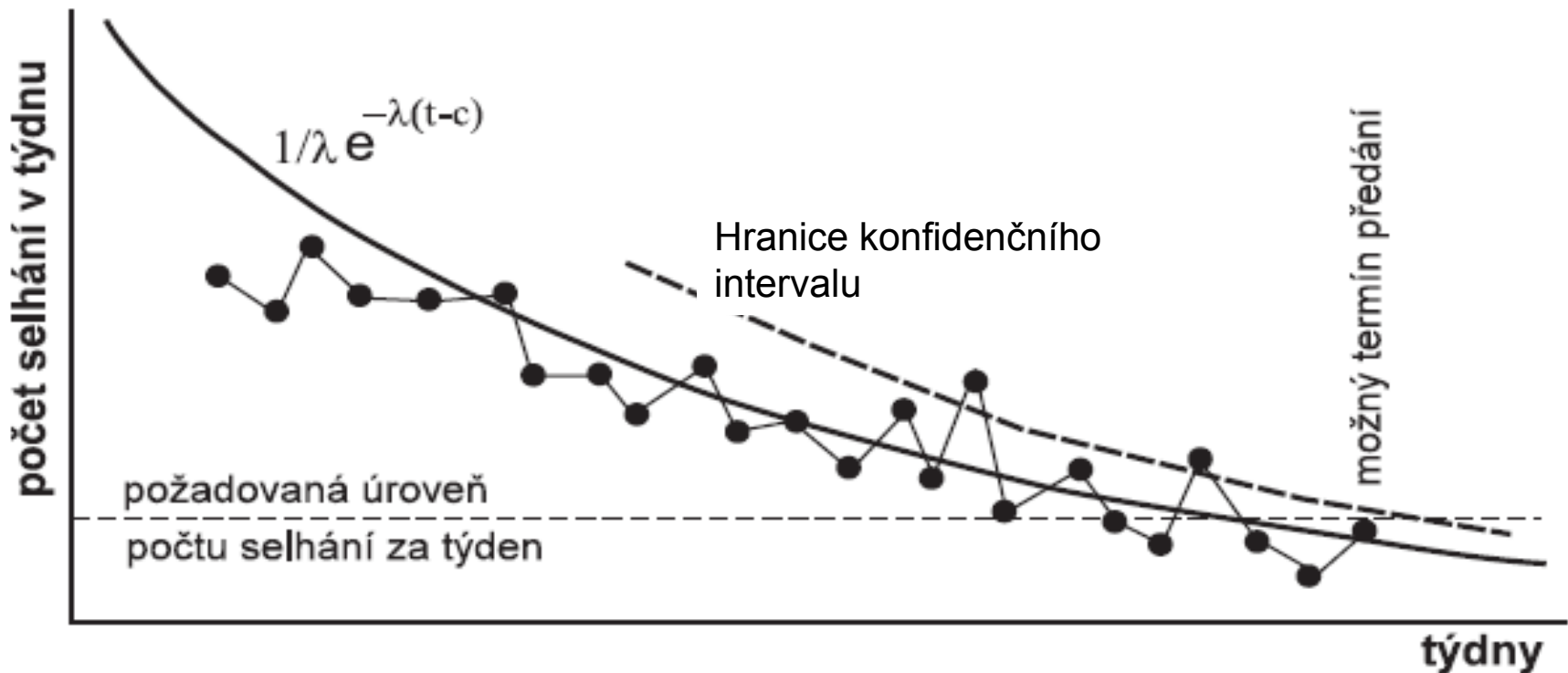
Ale jak řídit, když průběh funkcí  
neznám?

- Intuice manažera - odhadne nejvhodnější dobu
- Obecně je pocit, že lépe předat před minimem než po minimu
- Proč?

# Obecně je pocit, že lépe před minimem než po minimu

- Po minimu z dlouhodobého hlediska zvětším  
možnost, že vyrostete konkurence
- Také ne zcela správný pocit, že už je to dobré

Jak poznat, že je produkt dostatečně spolehlivý,  
použitelné jako test ukončení testování u kritických aplikací



Existují účinnější postupy z teorie spolehlivosti.

Exponenciální model poklesu frekvence selhání je zpravidla správný  
(podmínkou je nezávislost detekce selhání)

# Druhy testů

- **Částí, (unit tests)**
  - samostatných kusů programů
  - dost často testují programátoři sami
- **Integrační**
  - Shora
  - Zdola
  - Selektivní, sendvič, jádro a programy
- **Regresní (zopakování většiny testů)**
  - Může být příliš náročné na uživatele a na provoz, v agilním vývoji spíše pravidlo

# Druhy testů

- **Funkcí**
  - Ucelených akcí systému
- **Systemu**
  - V simulovaném provozu jako celek
- **Předávací**
  - Podle smlouvy
- **Test užíváním**
  - Zkušební provoz
- **Test simulací nebo prototypem**



# Integrace zdola

- Je třeba mnoho pomocných dat a programů
- Funkce systému se testují a mohou předvádět poměrně pozdě
- + Moduly jsou obecněji použitelné (méně závisí na změnách funkcí systému)
- + Ověřují se možnosti implementace

# Integrace shora

- + Je třeba méně pomocných dat a programů
- + Funkce systému a rozhraní se testují a mohou předvádět poměrně brzy
- Moduly jsou použitelné jen v daném prostředí (někdy je to výhoda)
- Chyby na vyšších úrovních mohou být fatální (až příliš pozdě se zjistí problémy s implementací)

# Podpora testů

- CASE systémy mají testovací roboty
  - IBM Rose, RRrobot
  - Agilní přístup, buduje se testovací podsystém
  - Generátory (power builder)
  - Systémy podpory programování usnadňují testování, spíše detailů
- Pracnost testování závisí na architektuře systému, v SOA a při agilním vývoji je menší
  - Znovupoužitelnost a produkty třetích stran
  - Možnost testování služeb přesměrováním zpráv
  - Využívání prototypů a žurnálů
  - Mentálně zvládnutelné
  - Použití ISO norem u kritických aplikací výhodou i nutnosti

# Programátoři a testéři

- Tři varianty „spolupráce“
  1. Programátor je současně testér
    - Populární, rychlé, málo účinné (vadí u kritických aplikací)  
Kompromis: Unit tests. Testy částí.
  2. Testér je specifická role, bílé skříňky
    - Testér spolupracuje s programátorem při nápravě selhání
  3. Testér testuje černé skříňky, testéři nemají zdrojové kódy ani kontakt s programátory
- Nejúčinnější je 3, ale je to velmi drahé a vyžaduje to kvalitní profesionály

# Terminologie testování

- **Selhání** – jiný než očekávaný výsledek
- **Neúspěch testu** – nedetekuje selhání
- **Testový případ**: Data, scénář, výsledky
- **Testová procedura**: Sít' testových případů
- **Test**: Sít' testových procedur
- **Položka k testování**: Vše, co potřebuje testový případ (prostředí, SW, data, scénáře, očekávané chování systému, výstupy)

# Terminologie testování

- U agilního programování se nejprve definují testové případy pro nově vyvíjenou část
- Naprogramuje se část
- Testové případy se použijí k otestování části (fakultativně), provádí programátor
- Testové případy se integrují s ostatními testy, systém se integruje
- Systém se při agilních postupech integruje a testuje jako celek, to lze u menších a nekritických aplikací či u nepříliš složitých služeb v SOA

# Specifikace testů, neagilní případ, složité či kritické části

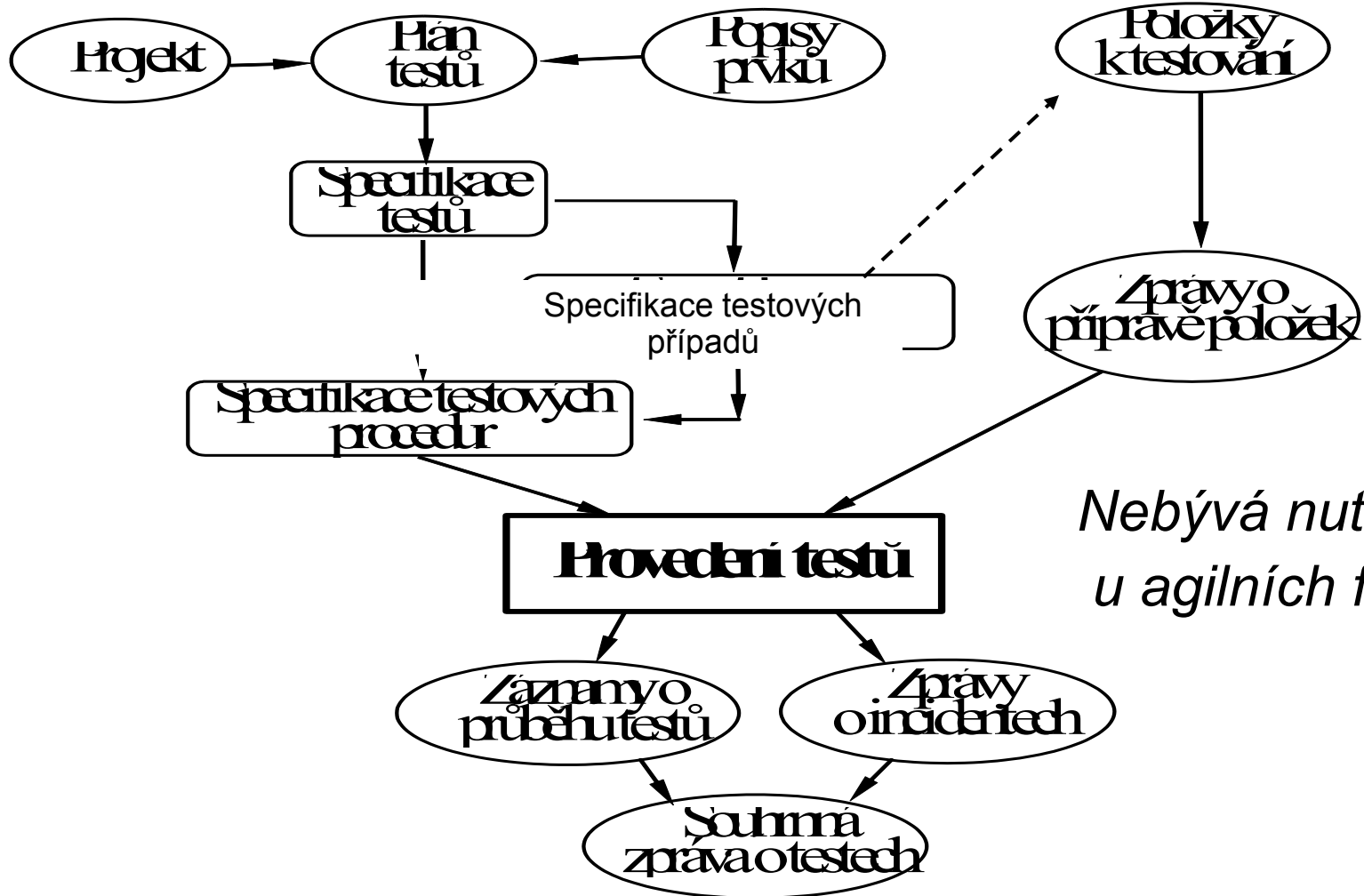
- Id
- Podmínky a způsob provedení
- Popis testu, testové procedury
- Kriterium přijetí/zamítnutí testů
- Kriteria pro přerušování testu
- Rizika

# Popis problému (selhání)

- Prováděný testový případ, procedura, test („místo“)
- Skutečné výsledky ve srovnání s očekávanými
- Popis anomálie
- Doba
- Pokusy o opakování
- Kdo testoval
- **Nemá být spojováno s návrhy oprav**



# Proces testování



*Nebývá nutné  
u agilních forem*

# Souhrnná zpráva o testech

- Zprávy o předání položek
- Žurnál testů
- Zprávy o selháních (incidentech)
- Souhrnné hodnocení
  - Co se vše testovalo
  - Hodnocení výsledku (přijmout/nepřijmout testovaný produkt, případná opatření)

# Testové metriky (Příklady)

1. Počet modulů modifikovaných při vývoji/změně.
2. Počet defektů odstraněných v dané etapě.
3. Pro modul počet defektů na tisíc řádků.
4. Počet změněných příkazů/míst.
5. Doba na lokalizaci a odstranění chyby.
6. Druhy a frekvence selhání systému.
7. Výčet modulů s největším (nejmenším) počtem defektů.
8. Výčet modulů, které jsou nejsložitější, tj. těch, pro něž nějaká metrika nabývá extrémních hodnot nebo překračuje nějakou hodnotu.

# Evidence příčin selhání

- chyba specifikací
- chyba návrhu,
- kódovací chyby,
- selhání hardwaru,
- chyba v reakci softwaru na selhání hardwaru (př. Škoda Plzeň)
- chyba obsluhy

# Využití testovacích metrik

- Kdy ukončit testování
- Dodatečná kontrola efektivnosti oponentur
- Ověřování kvality nástrojů a jejich efektů
- Skrytě hodnocení členů týmu
- Lze použít technologie hodnocení kvality technických výrobků (střední doba mezi poruchami, kritické části)

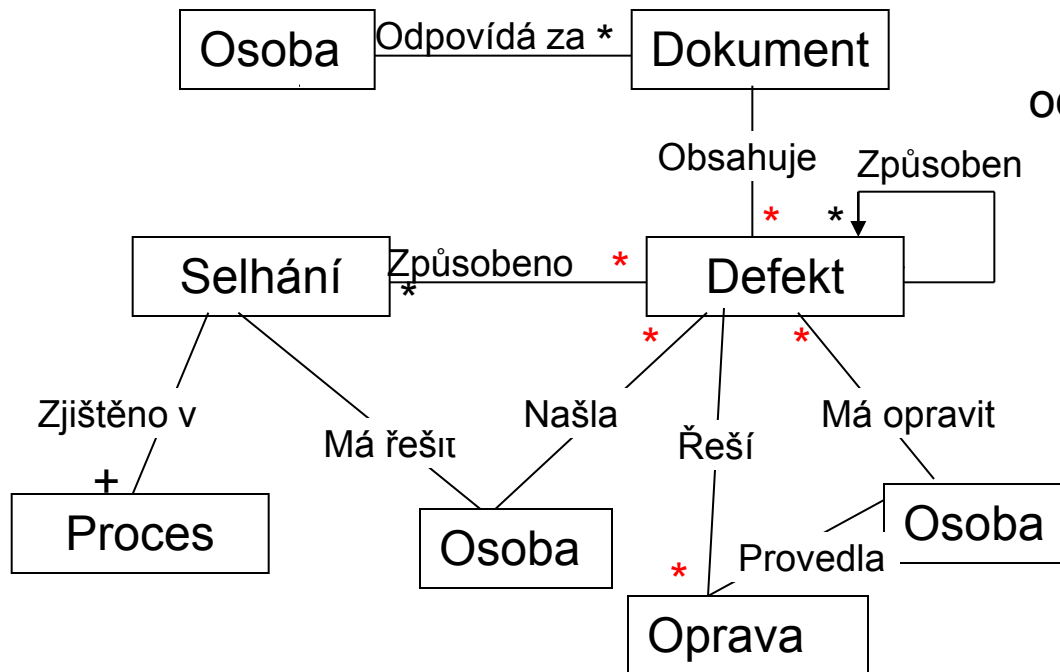
Datová báze výsledků testů (selhání)  
by měla být stejná jako u oponentur  
a u reklamací,

*Cíl je identifikovat (možnost) selhání  
defekty se detekují v následných  
aktivitách*

Je třeba dohledat prapříčiny

# Konceptuální schéma, opakování

- Jednoduchá datová struktura pro vyhledání zdrojů problémů + integrace s oponenturami



V procesu  
odstraňování selhání  
se \* změní na +

Proces je oponentura,  
test, nebo stížnost  
uživatele, technicky  
odkaz na zápis

Jak se dá zjistit, kdo  
udělal opomenutí

# Permanent obsolescence

- Dříve než systém otestuji se změni požadavky a všeobecné podmínky
  - Použité normy zastarají
  - Požadavky se změni
  - Trh se změni, mám jiné obchodní partnery
  - Změni se paradigma
- Řešení
  - Dělam po částech (komponenty, služby)
  - Koupím, znovu použij
  - Outsourcuji
  - Zlepším své procesy vývoje a údržby



# Zavedení

Jak instalovat a zavádět

Na koho se obrátit

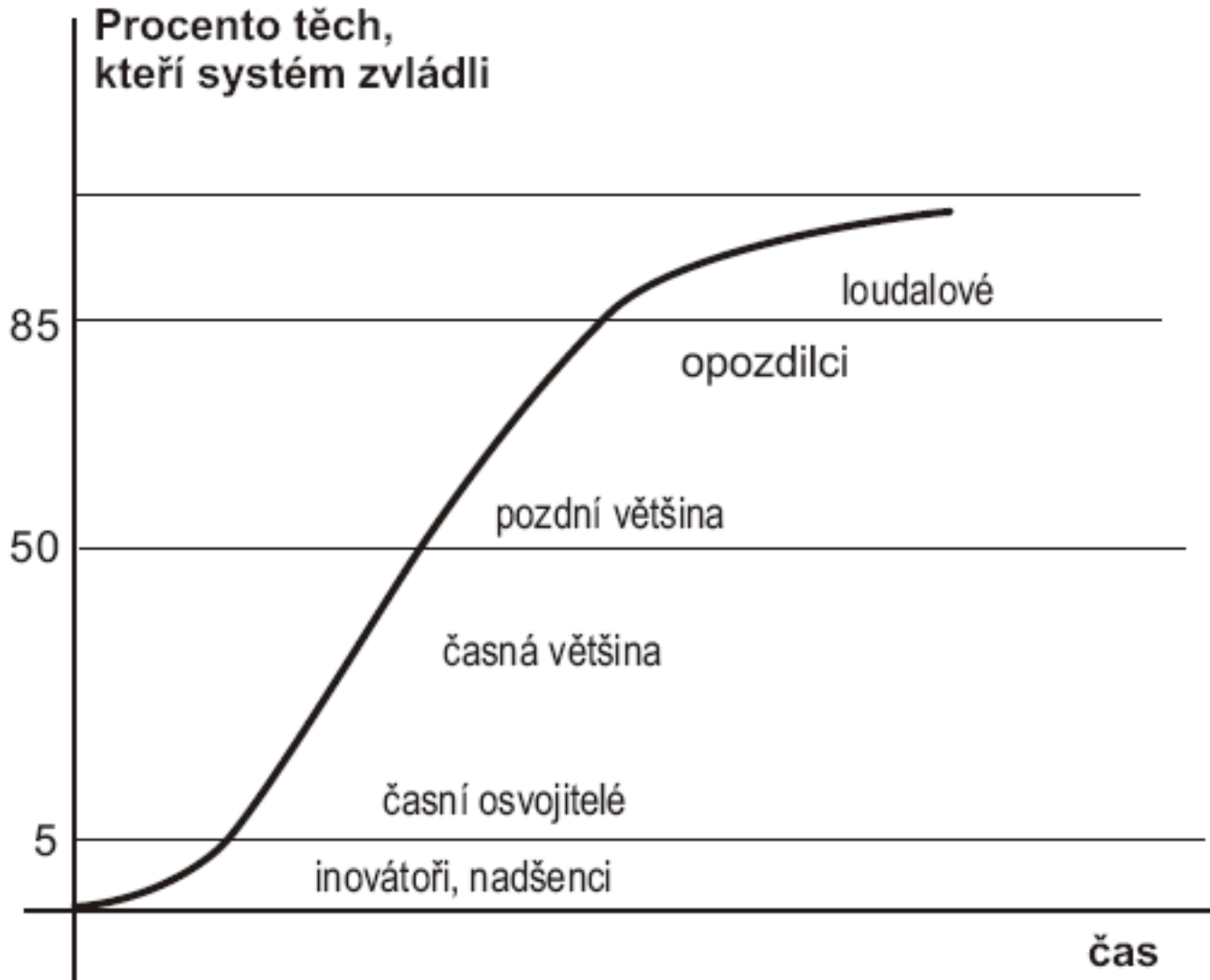
# Zavedení informačního systému

- Předání (instalace, předávací testy, zkušební provoz)
- Školení pracovníků
  - Spoluúčast tvůrců (nebývají ale pedagogicky zdatní, nevyužívá se jejich odbornost)
  - Vhodní jsou dobří lektori, dokonce specializované firmy
- Plánování přechodu
  - Konverze dat Postup překlopení
  - Vhodné je inkrementální zavádění, lze-li. Podmínkou je vhodná architektura.
  - Je vhodné stanovit kriteria úspěchu zavádění

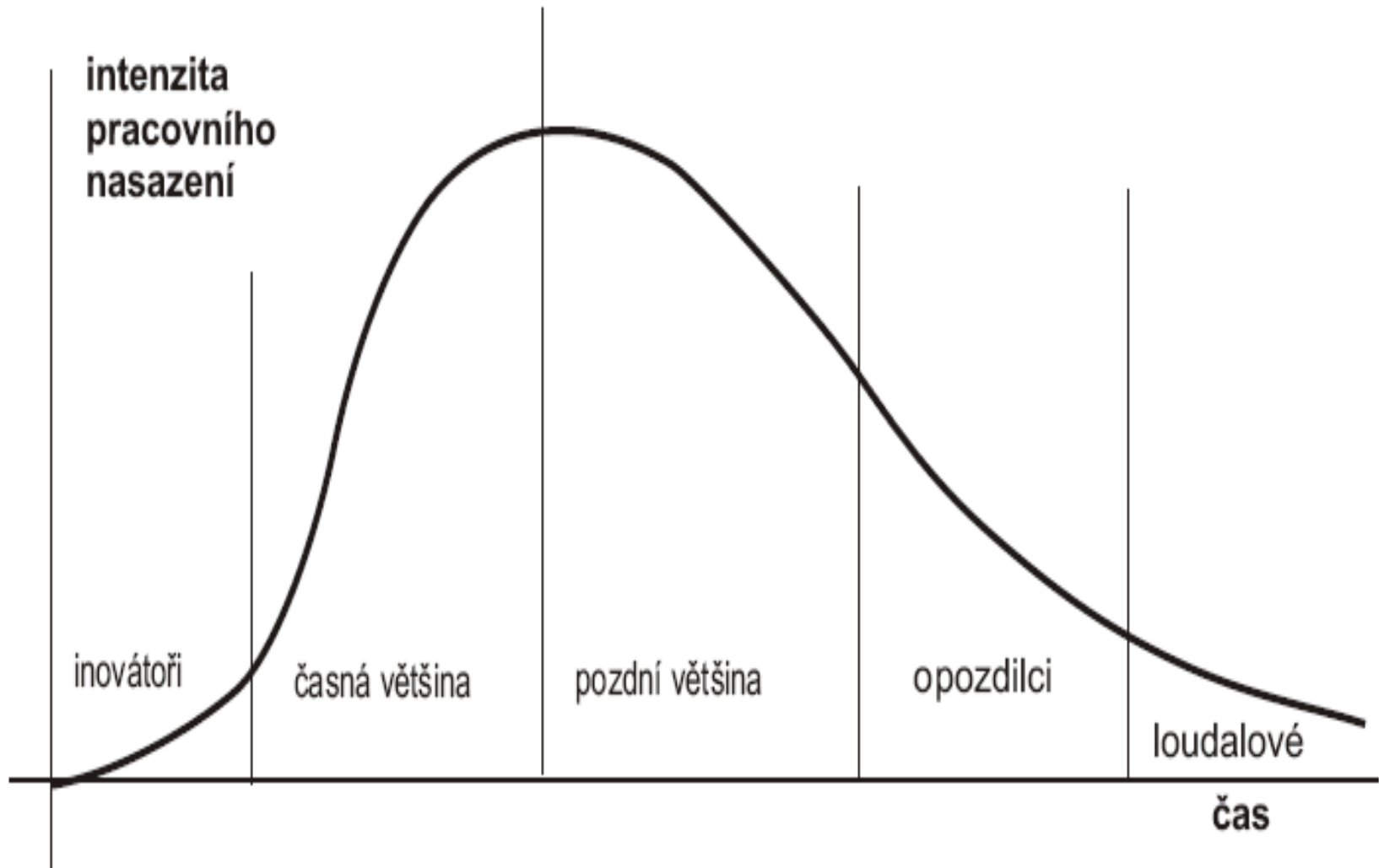
# Dokumenty při zavádění

- Vize systému a specifikace požadavků
- Návrh a zdrojové texty (nedělá-li dodavatel údržbu)
- Dokumenty o testech, případně deník projektu
- Manuály
- Dohoda o zkušební době a procedurách odstraňování chyb
- Záruky a rizika

# Křivka učení a typy uživatelů



# Křivka učení a typy uživatelů



# Koho získat jako spojence

- Inovátoři jsou motivováni novinkami, nikoliv zlepšováním své práce,
  - Nemívají prestiž mezi spolupracovníky, nerozumí jejich potřebám
  - Brzy ztrácí zájem
- Časní osvojitelé chtějí zlepšovat svou práci a noviny při tom vítají, mívají vysokou reputaci u uživatelů, to jsou ti praví spojenci
- Časná většina inovace spíše vítá, ale chce mít svůj klid

# Křivka učení

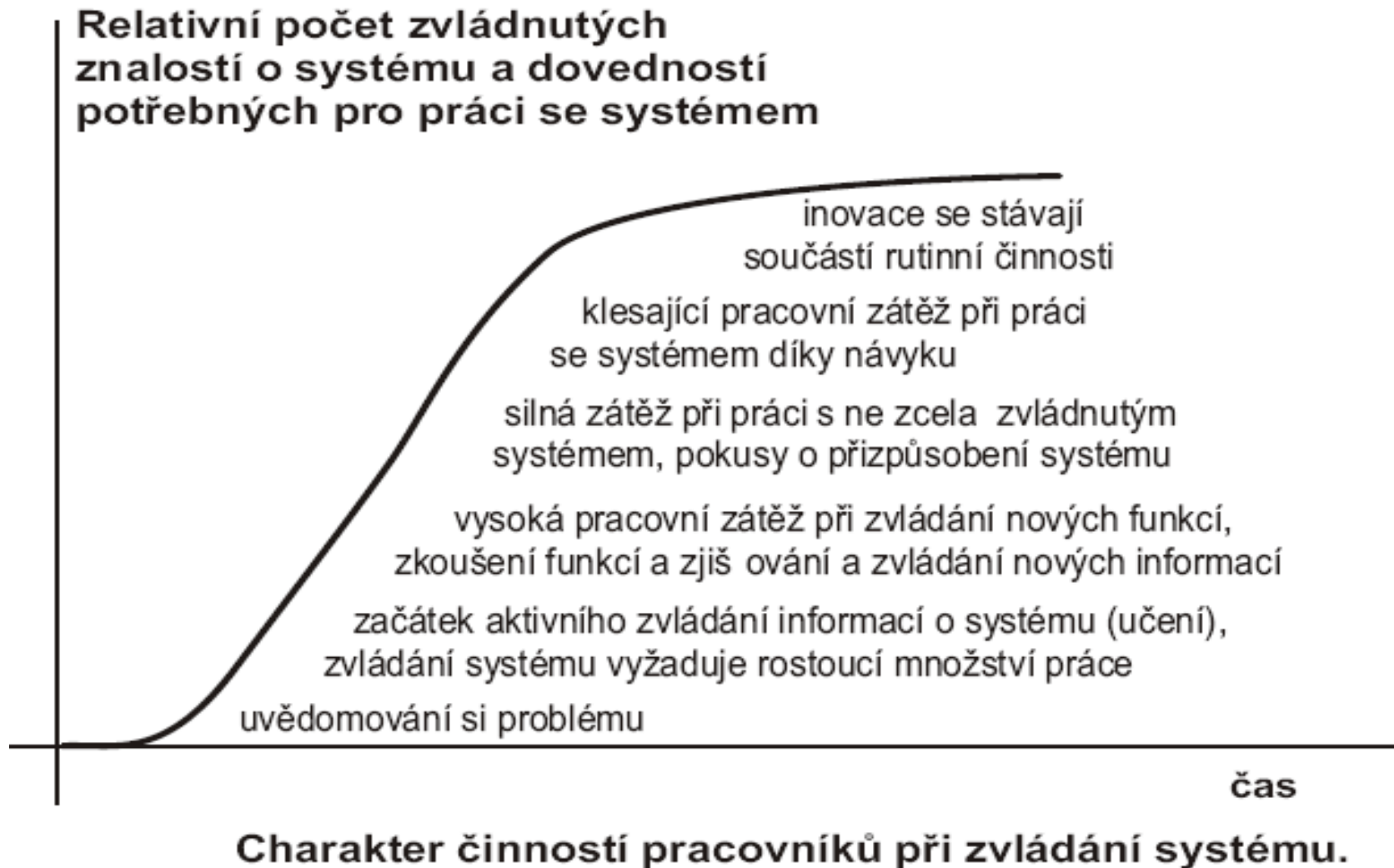
- Nemá být příliš strmá (intenzita učení příliš vysoká) ani příliš plochá
- Je nutné zvládnání chápat jako učení, je to tedy specifický úkol a je proto třeba zajistit
  - Kvalitní vyučující
  - Čas, pomůcky a prostředí
  - Hodnocení pokroku („známkování“)
- Často se to podceňuje

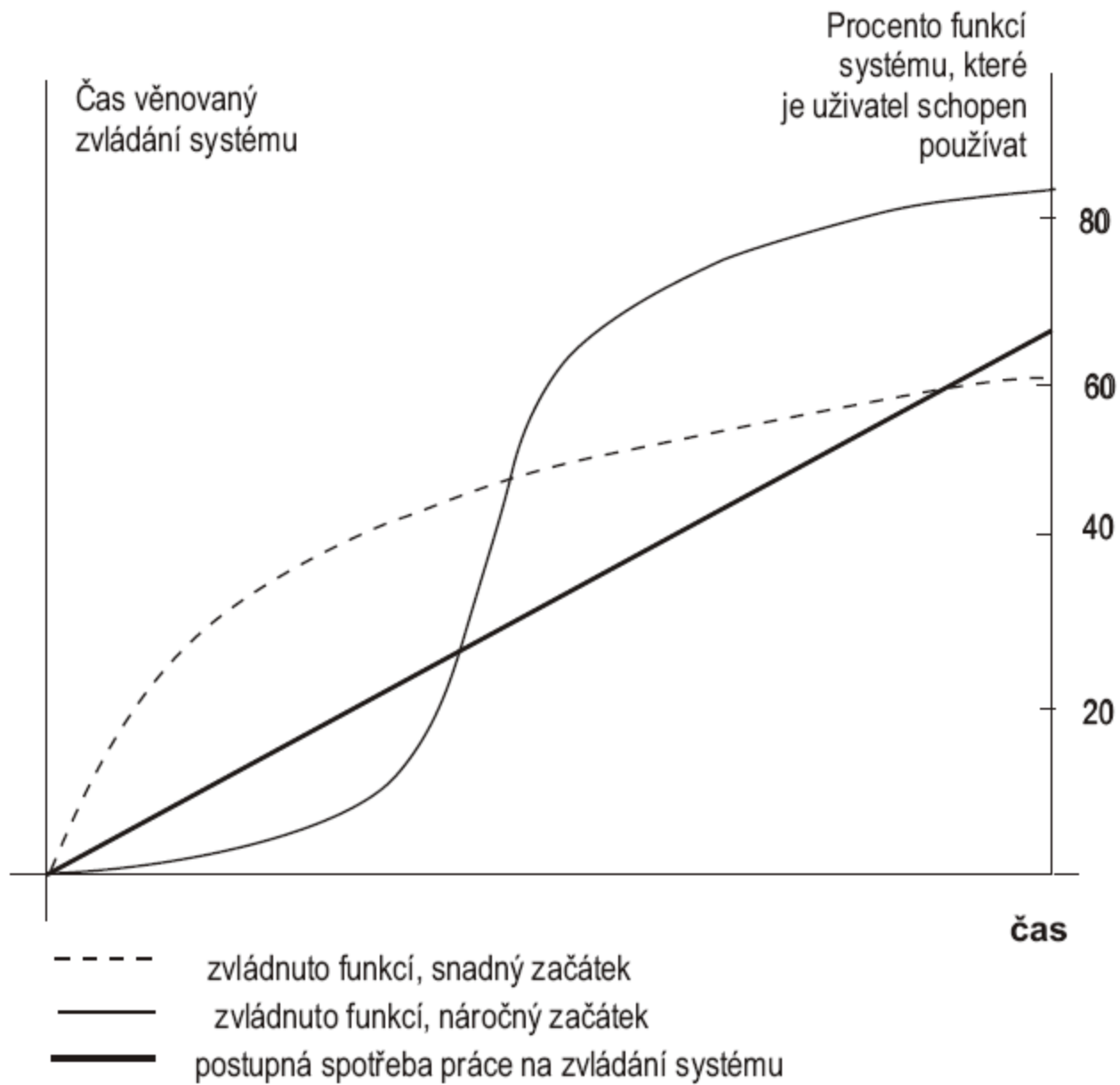


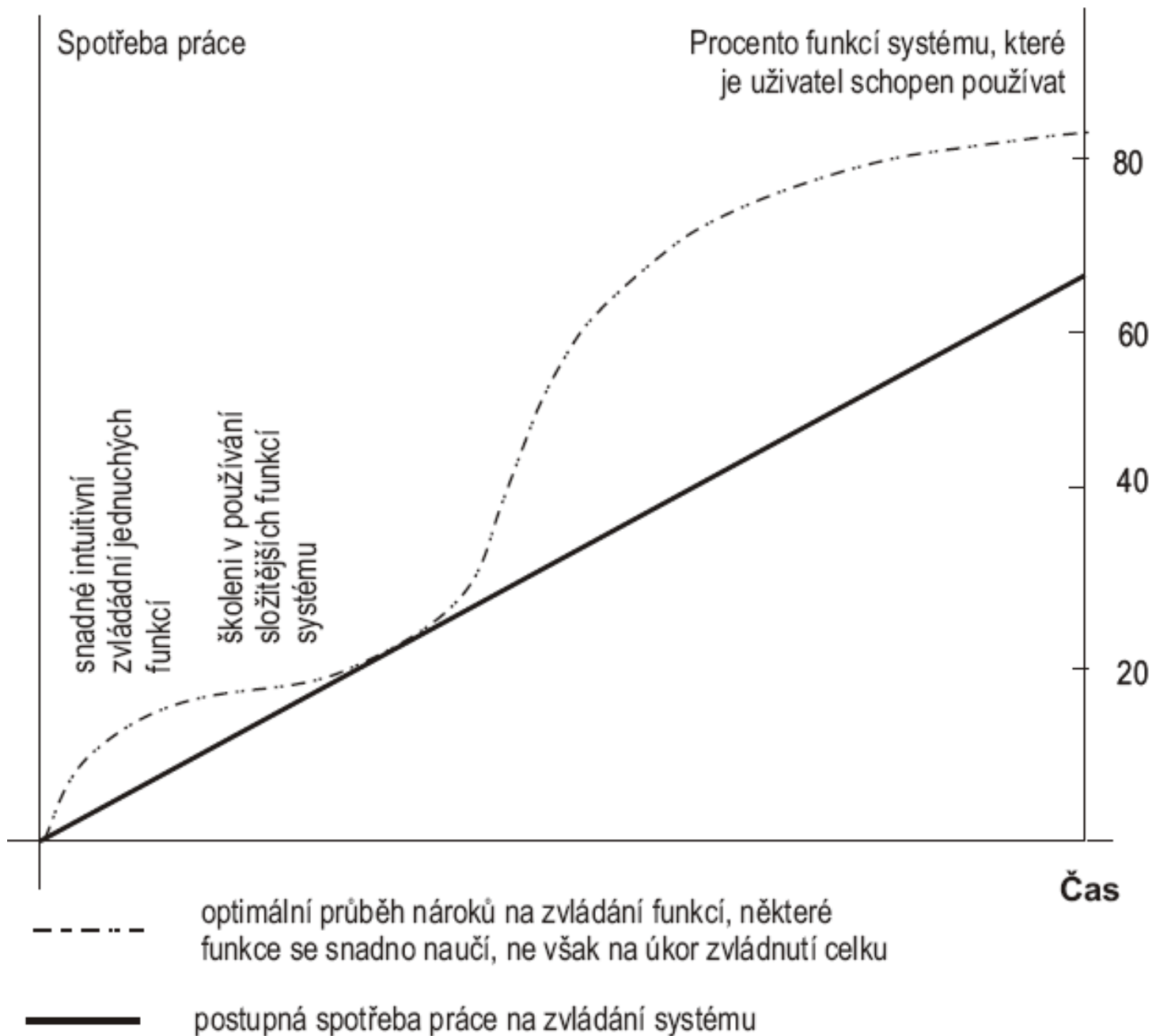
## Učení metodou pokusů a omylů

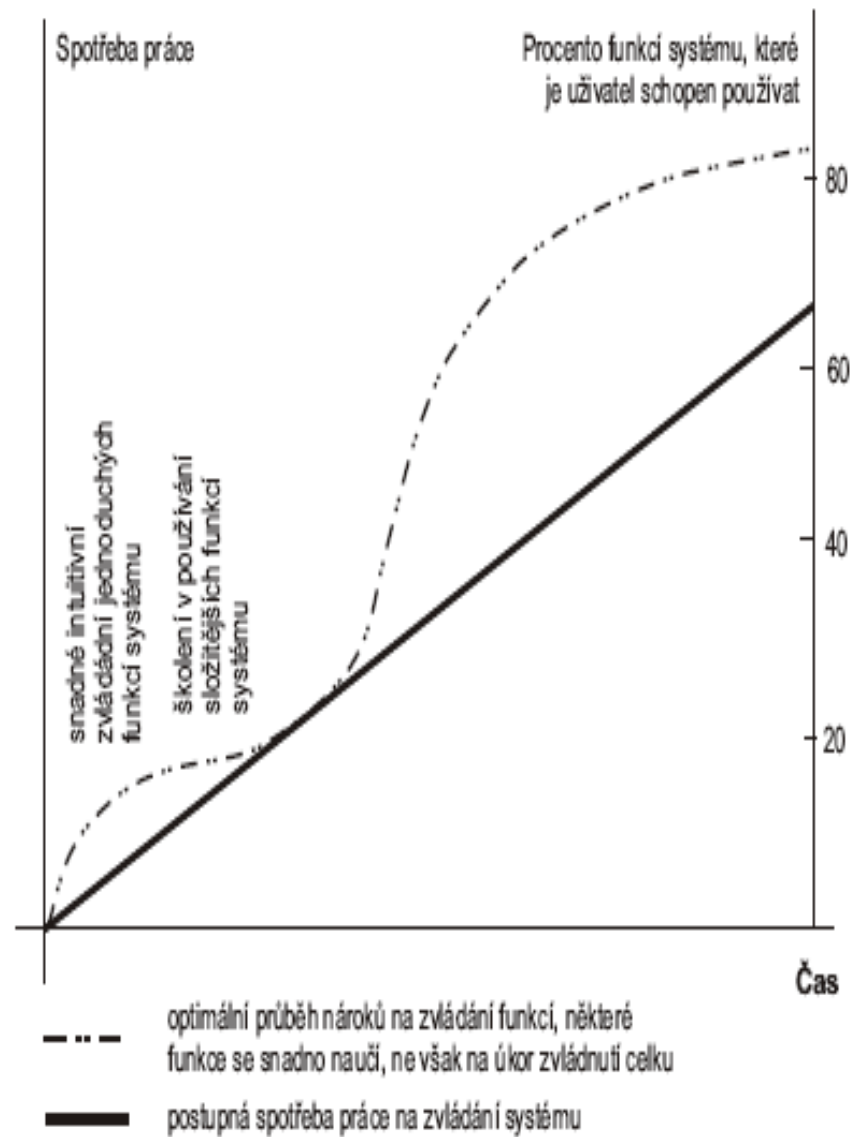
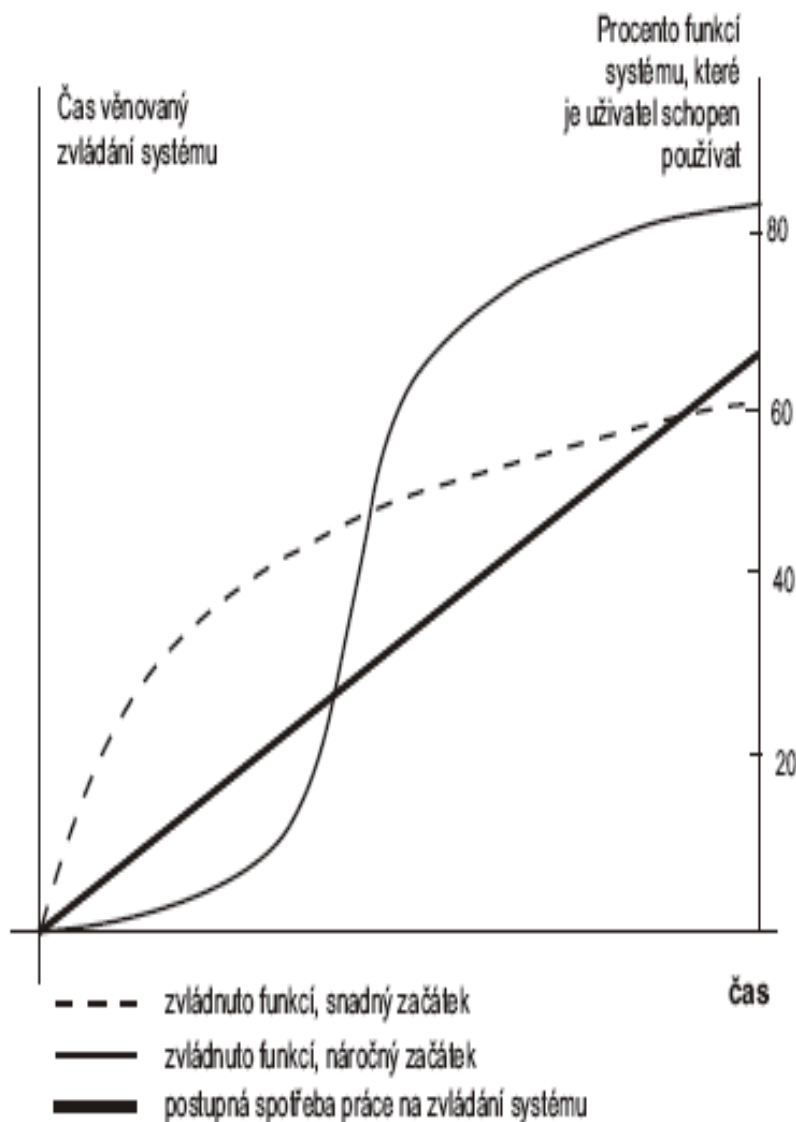


# Křivka učení a typy zvládnání









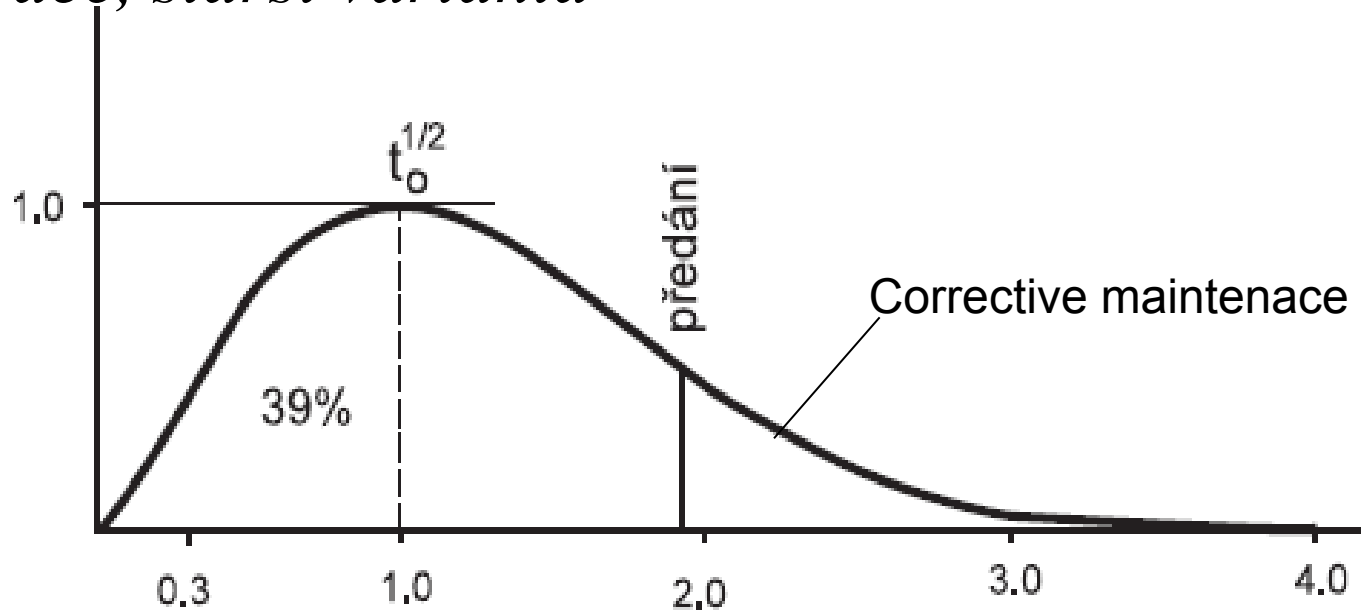
# Údržba

I instrukce se z logického pohledu  
(potřeb) opotřebují

# Druhy údržby, opakování

- Corrective odstranění defektů, které nebyly detekovány oponenturami a testováním, vlastně ty, na které nebyl čas ani prostředky
- Enhance – vylepšování
- Adaptive – přizpůsobování změnám platformy, přenos, změny norem

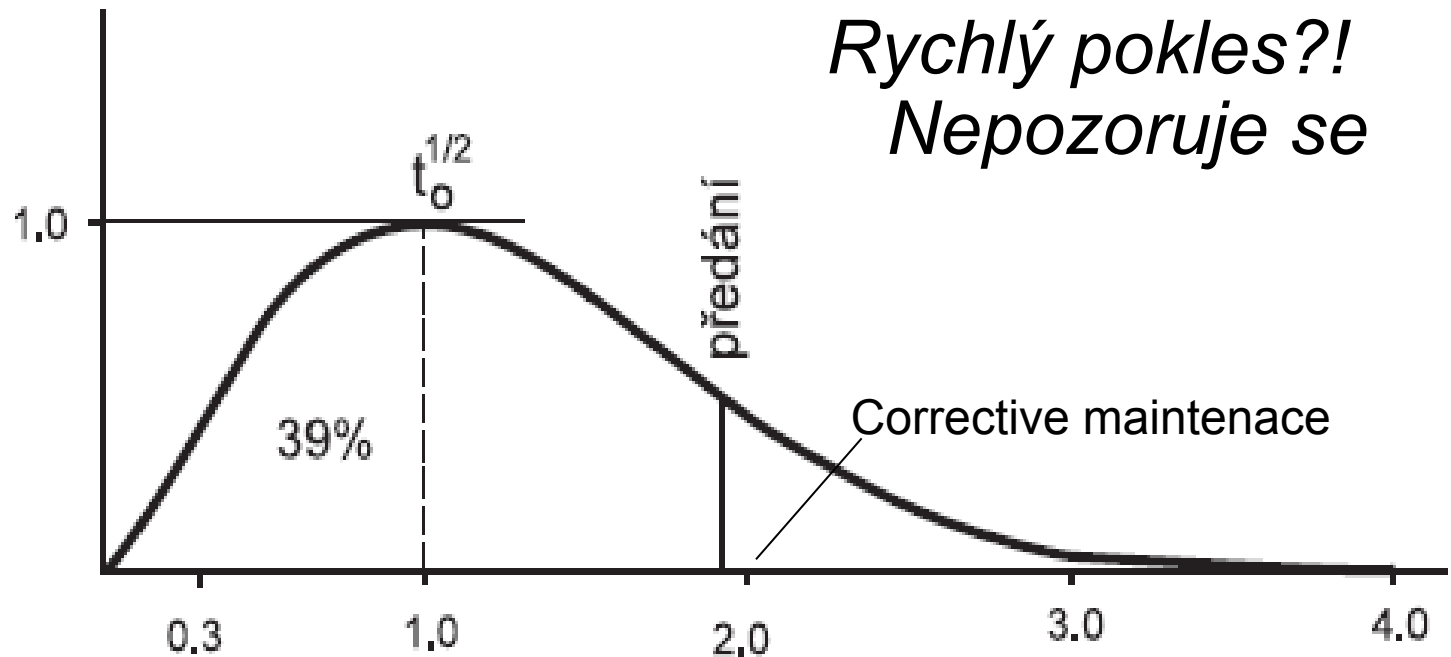
## *Model průběhu velikosti týmu a tedy intensity práce, starší varianta*



Obr. 15.12: Rayleightova křivka. Část plochy pod křivkou po předání jsou práce, které budou provedeny v rámci údržby (corrective maintenance). To, co se do okamžiku předání nestihne udělat, přechází do údržby, kde se projevuje jako neodstraněné chyby.

$$team(t) \hat{=} K \cdot t/t_0 \cdot \exp\left(-\frac{t^2}{2t_0}\right)$$

*Rychlý pokles?!  
Nepozoruje se*



Obr. 15.12: Rayleightova křivka. Část plochy pod křivkou po předání jsou práce, které budou provedeny v rámci údržby (corrective maintenance). To, co se do okamžiku předání nestihne udělat, přechází do údržby, kde se projevuje jako neodstraněné chyby.

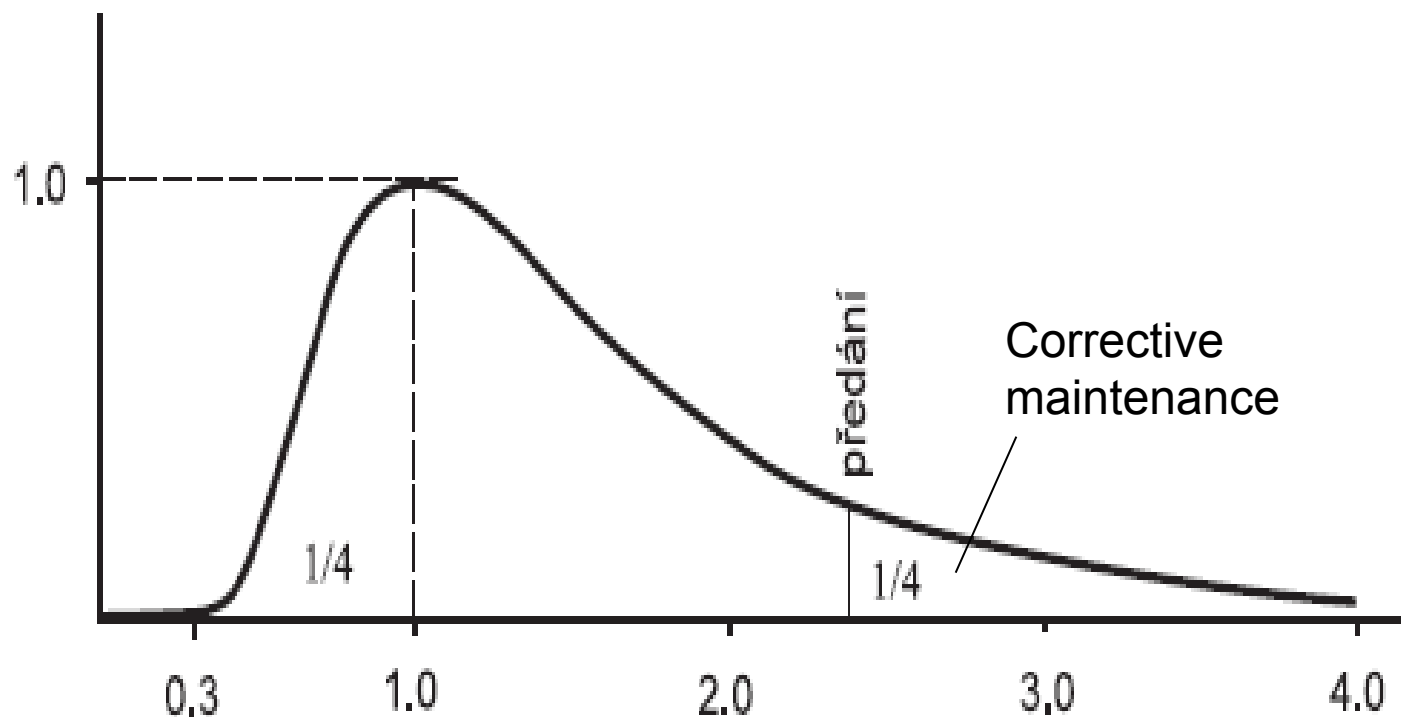
*Pro  $t > 3$  je velikost týmu prakticky nula  $\Rightarrow$  není co dělat, nebyla by corrective maintenance, to se nepozoruje. Pravidlo polovin: ve vrcholu jsem v půli se spotřebou práce i s dobou řešení, to je příliš optimistické*

$$team(t) \hat{=} K \cdot t / t_0 \cdot \exp\left(-\frac{t^2}{2t_0}\right)$$



# *Jsou důvody pro komplikovanější model*

*Zobecnit na  $t = aT+d$*

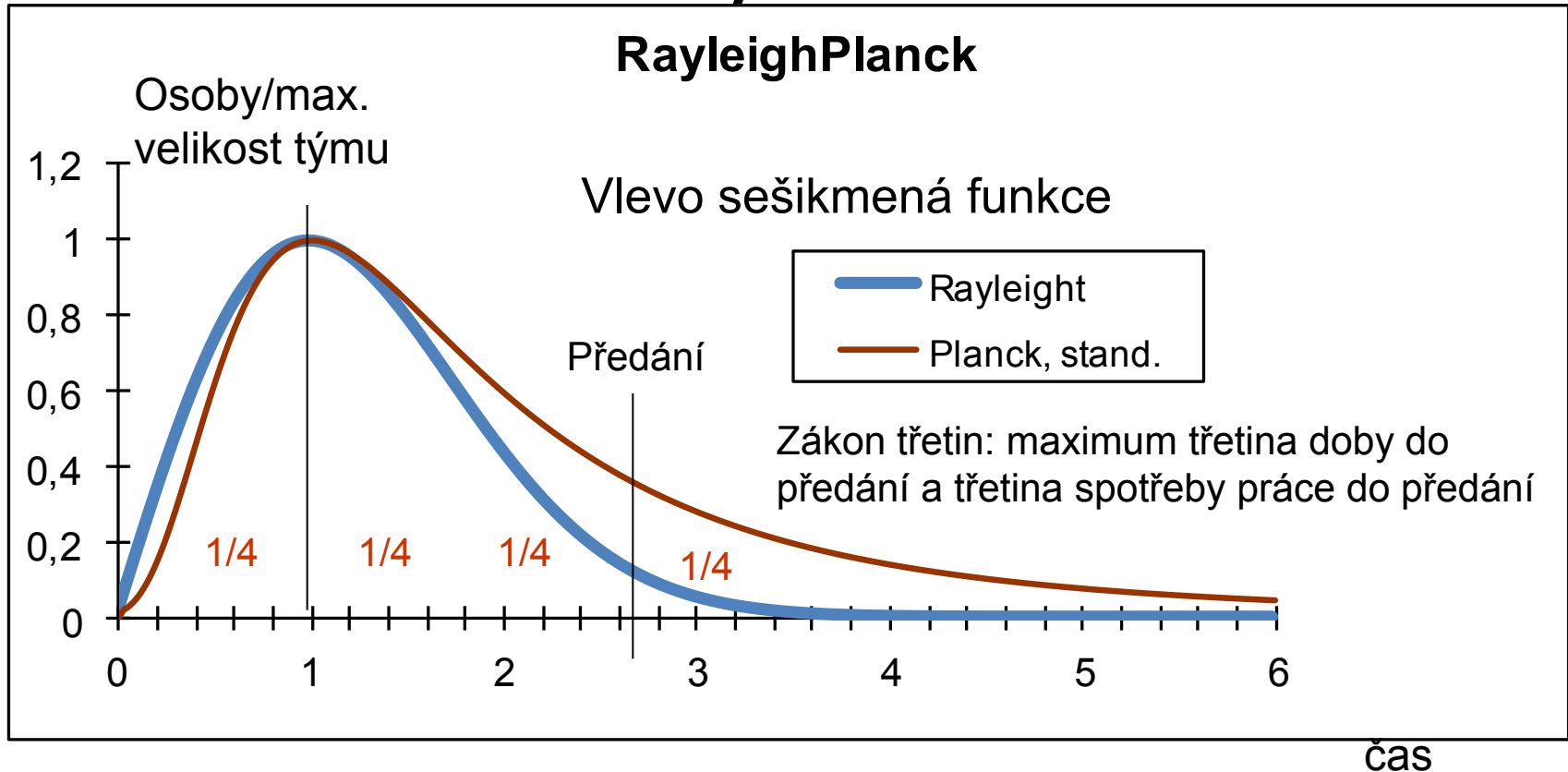


Obr. 15.13: Normalizovaný tvar Planckovy křivky  $Planck(t) = 142.32 \cdot t^{-5} / (\exp(4.9651/t) - 1)$ .

*Mnoho práce se vykoná i pro  $t > 5$ ,*

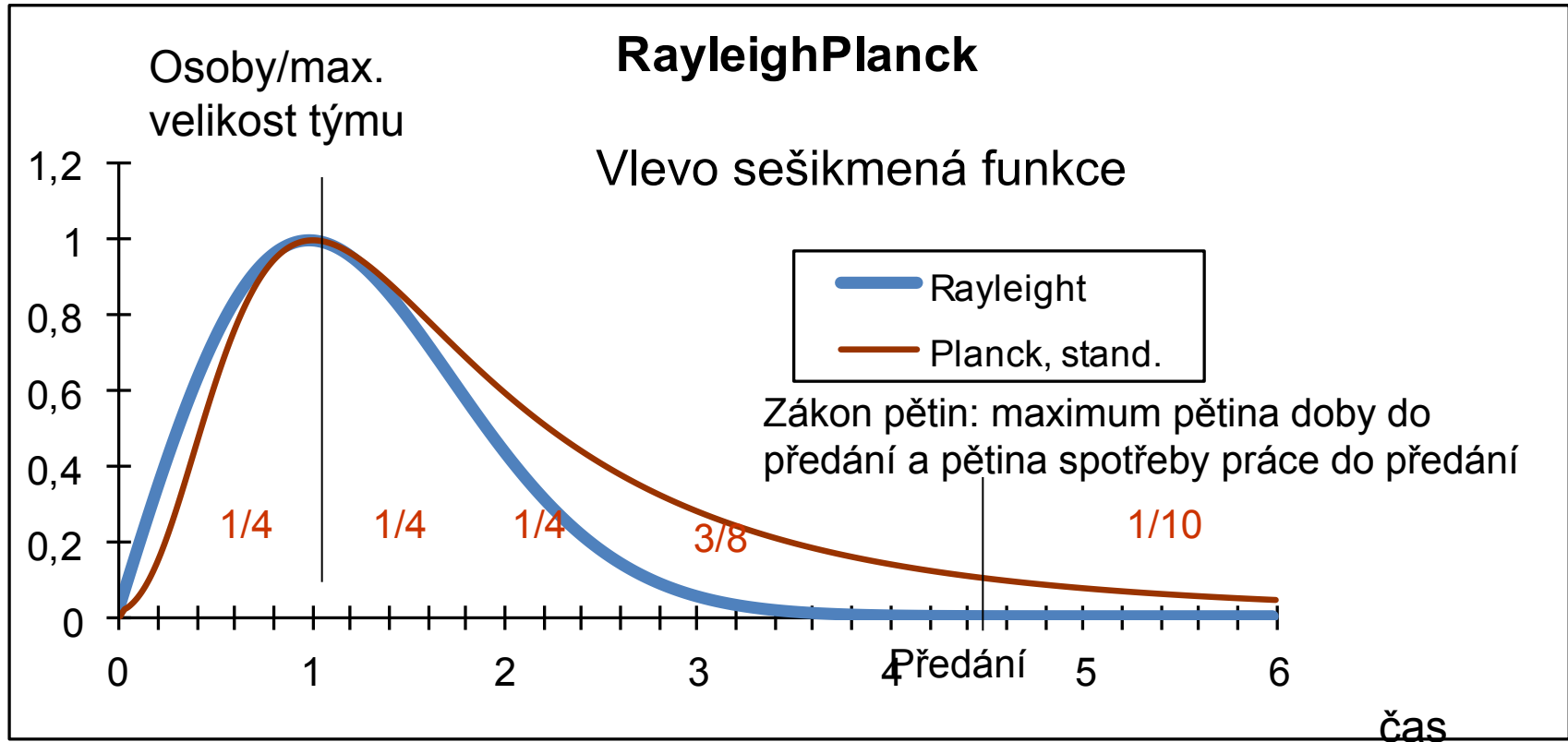
*Posun vlevo lineární transformací nezávislé proměnné*

# Najímaný tým, vrchol a odhad doby řešení



Pravidlo třetin. Do vrcholu  $1/3$  prací a  $1/3$  doby (za běžných okolností). Transformace proměnných tak, aby max bylo v bodě 1 a mělo hodnotu 1 a v nule byla hodnota funkce prakticky nula. U pevného týmu odpovídá intenzitě práce

# Najímaný tým, vrchol a odhad doby řešení krirických systémů

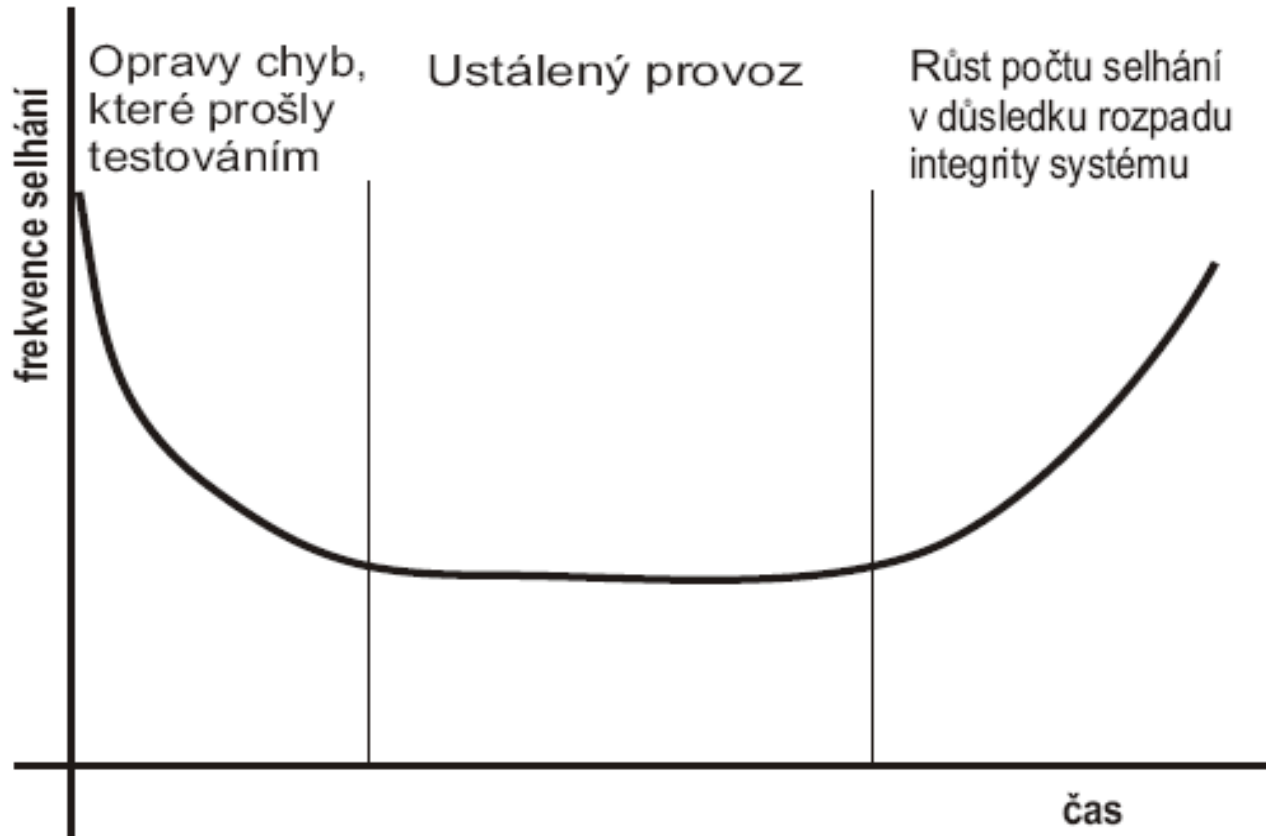


Ve vrcholu méně než  $1/5$  doby do předání a málo více než  $1/4$  (0.277) prací

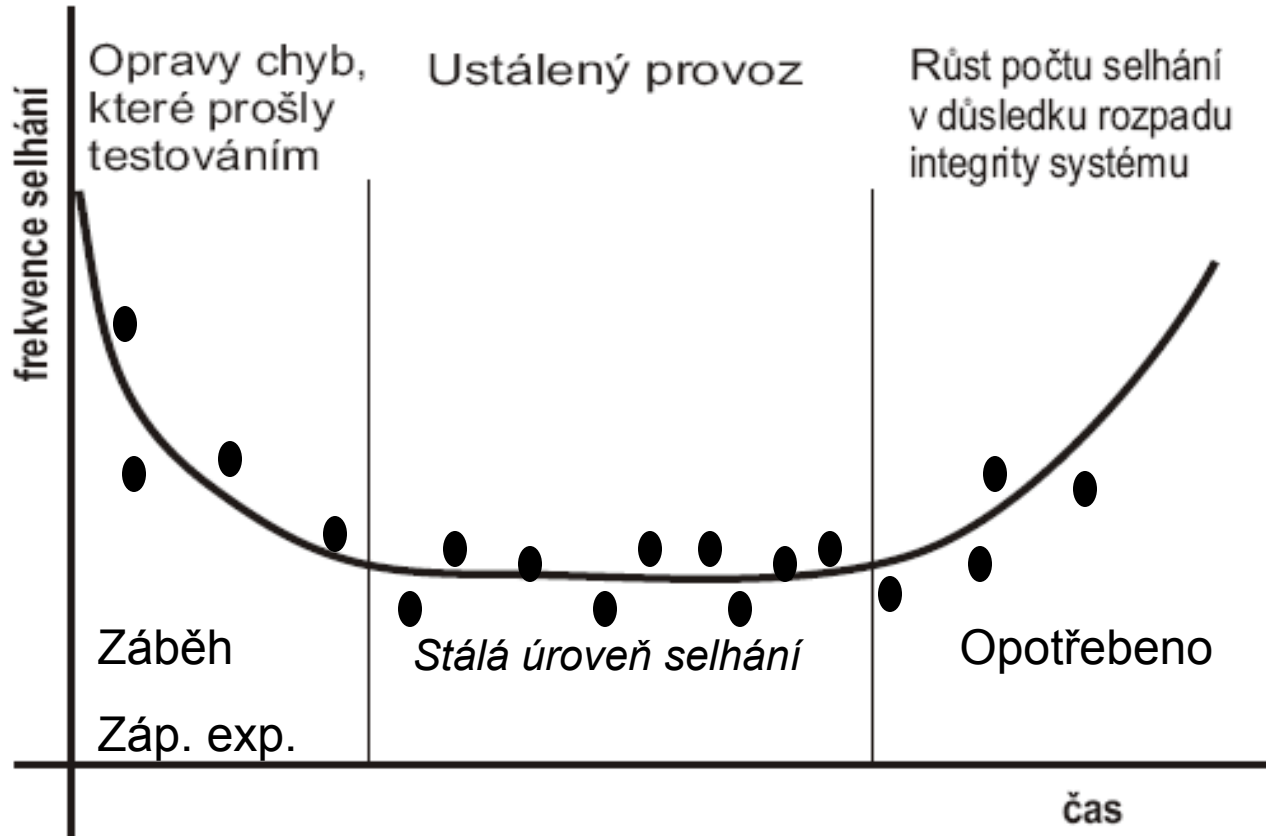
# Etapy údržby

- Převzetí
- Etapa investic
  - Corrective maintenance, prvá vylepšení a přizpůsobení
- Etapa maximálního užitku
  - Vylepšení požadované uživateli, dnes skoro permanentní vývoj
  - Regresní testy, stabilní provoz
- Zmenšování užitku
  - Vylepšení pro další uživatele, zlepšování výkonu, roste počet problémů

# Vanová křivka. I SW se opotřebí



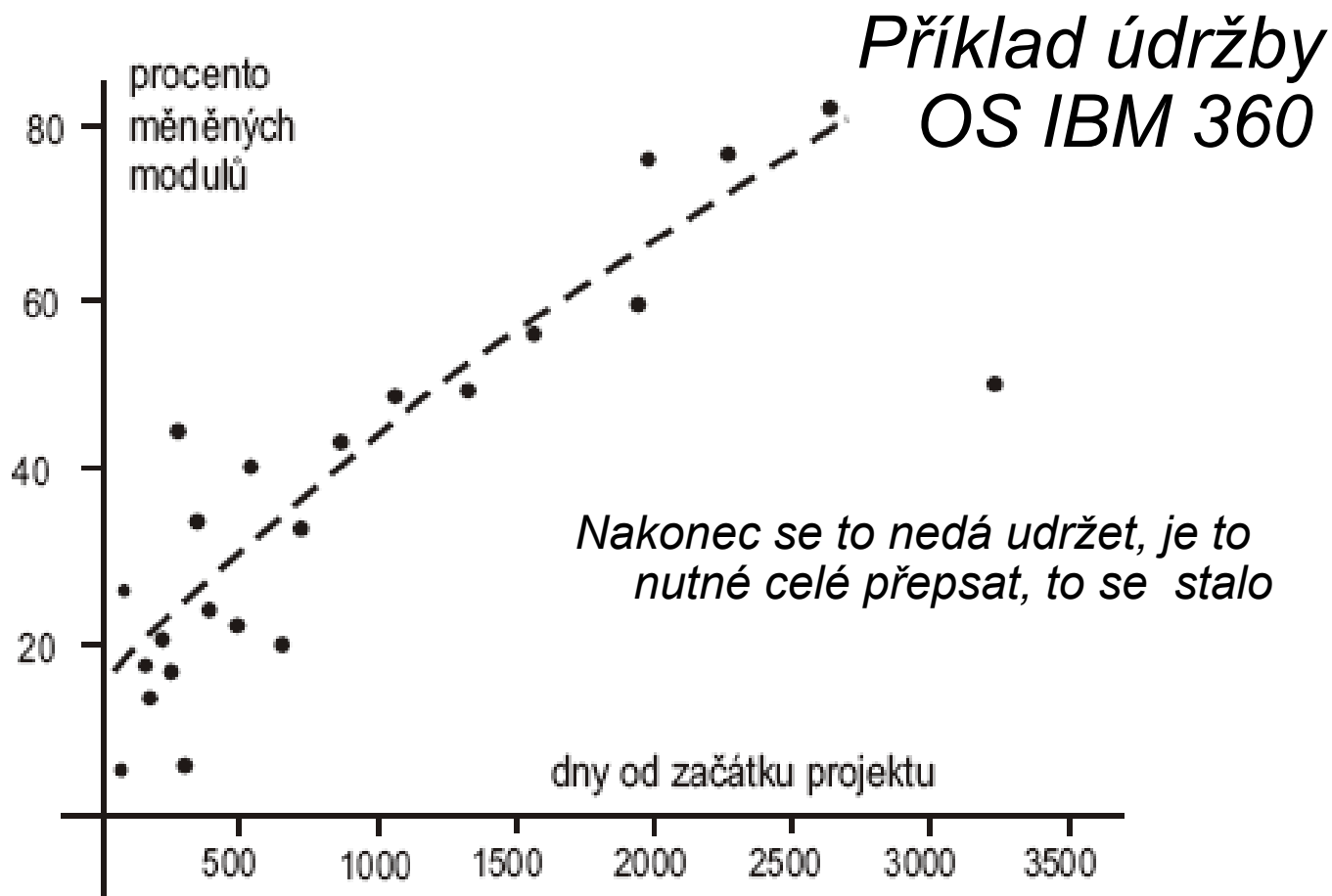
# Vanová křivka. I SW se opotřebí



# Dno vanové křivky implikuje **nenulovou** střední frekvenci selhání

- Opravou se zanesou další chyby
- SW se stále upravuje (vylepšuje, přenáší na nové platformy) a tím se zanáší problémy a další chyby, tím se údržba stává stále pracnějšší.
  - Za odstraněný defekt přibude často nový, někdy i více než jeden
  - Není proto možné zero defect software

# Složítost údržby roste s časem, důvod stoupající větve vanové křivky

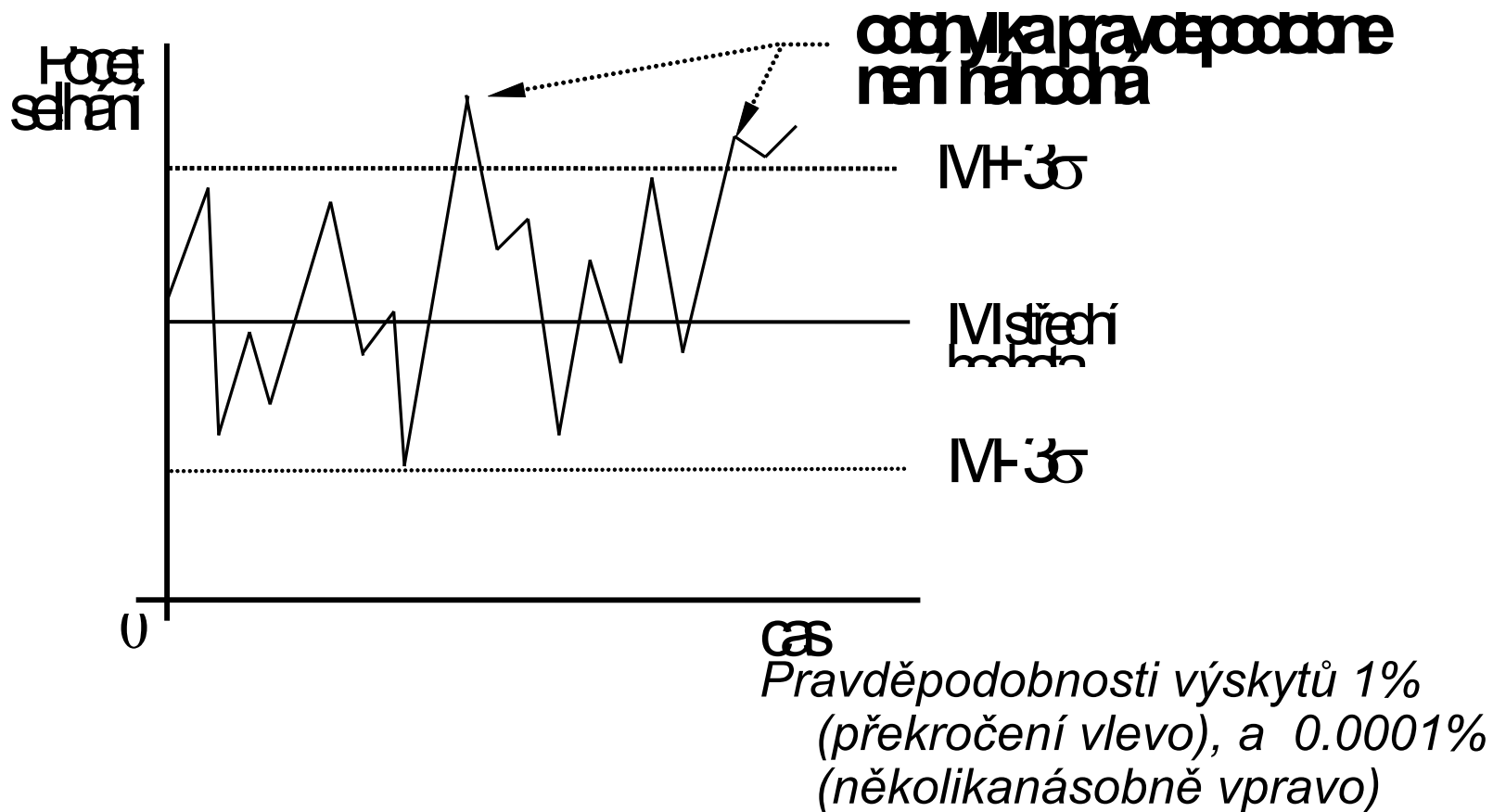




# Hlubší důvod k vyhození systému

- Pozoruje se, že si systém zachovává své základní vlastnosti plynoucí z filosofie volby požadavků a technologie návrhu a vývoje
- Nectnosti systému se spíše zviditelňují a zesilují filosofie zastarává, není „in“,
- Platí to i pro jiné technické prostředky (nákladní auta)

# Jednoduchá kontrola toho, zda došlo k opotřebení



# Co se tím vlastně prokazuje

- Je málo pravděpodobné ( $p \ll 0,001$ ), že *nedošlo* k posunu střední hodnoty směrem nahoru, tj. je pravděpodobné, že k němu došlo (tak se testují statistické hypotézy)
- Lze použít efektivnější prostředky mat. statistiky, např. *t* test
  - je účinnější a přesnější, není ale tak názorný a je proto méně vhodný pro on-line zásahy

# Co snižuje pracnost údržby

- Správné specifikace (správné požadavky ale také správná dekompozice )
- Vhodná architektura celku (SOA),  
Struktura částí (Objektová orientace, komponenty)
- Převzetí existujícího (knihovny, produkty třetích stran existující domácí SW)
- Správné procesy vývoje (inkrementálnost, agilnost) a kvalita dokumentace, kvalita oponentur a testů, normy
- Přenositelné technologie (Java), normy, PaaS
- Vývojové nástroje (menší rozsah dokumentů, srozumitelnost, podpora korektnosti oprav,...)

# Co snižuje pracnost údržby

- Dobře vedené programování (Gunderloy)
  - Používám, co je napsáno (existující aplikace, produkty třetích stran, knihovny), normy
  - Používám moderní metodiky (OO, SOA, metanástroje jako XML a s ním spojené jazyky a DB) a vhodný jazyk
  - Agilní metody vývoje
  - Dohodnuté standardy
    - Komentáře, volby identifikátorů, pravidla strukturování, oponování, vývoj testů, ....

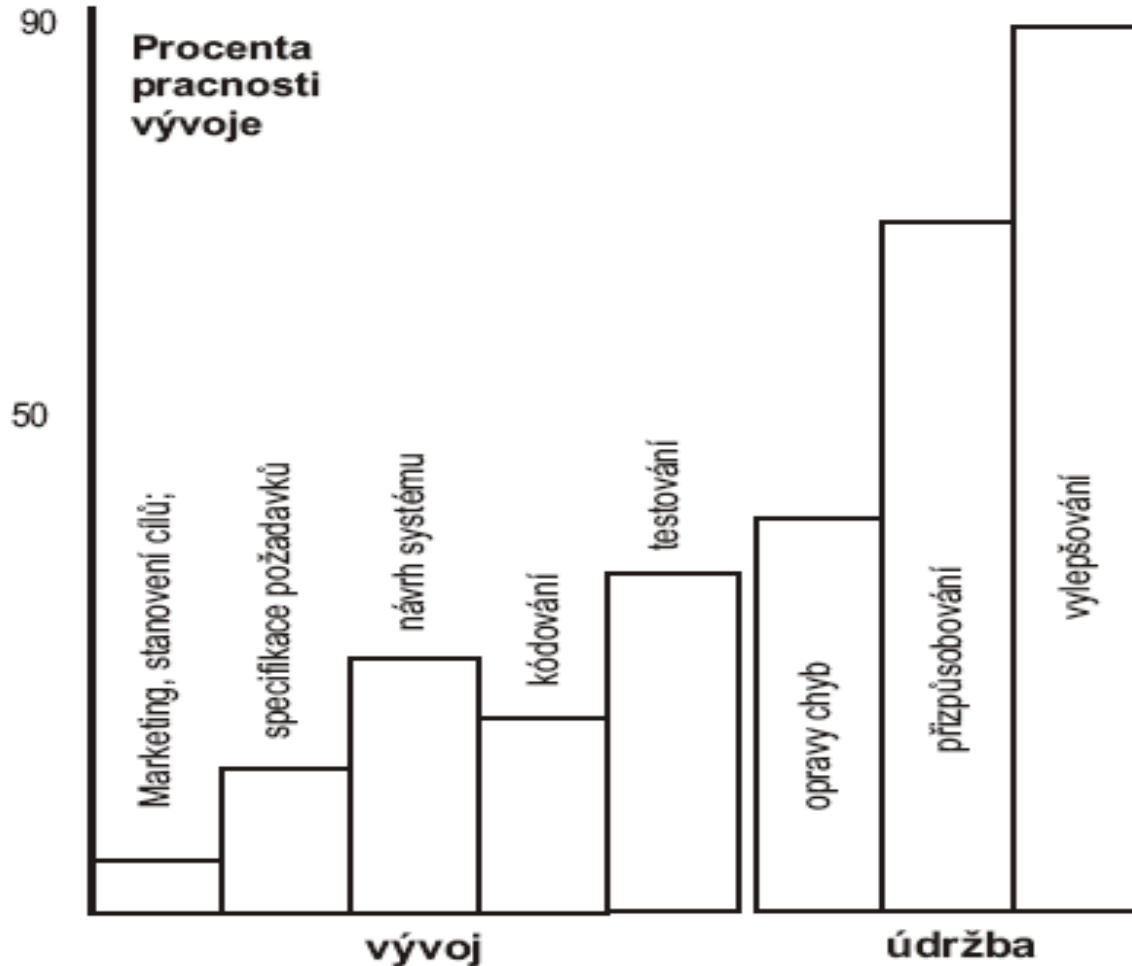
# Co snižuje pracnost údržby

- Kvalita technik údržby
  - Automatizace opakování testů
  - DB reklamací a defektů
  - Sledování trendů defektů (frekvence)
  - Sledování prapříčin, jaká chyba člověka je prvotní a které chyby jsou důsledkem
  - Použití logů komunikace přes uživatelské rozhraní a mezi komponentami (výhodné v SOA) a rozhraní

# Reinženýring

- V podstatě kompletní přepsání systému jako jediná alternativa k jeho zrušení
- Varianty
  - Enhansive, obvykle s novými funkcemi, nová architektura
  - Adaptive, změna platformy a jazyka či databáze
  - Vyjíměčně větší spolehlivost (SLA)
- **Rizika:** ovlivnění starými praktikami, neochota k žádoucím změnám, riziko, že to nestojí za to

# Podíly pracnosti



Údržba stojí podstatně více než vývoj (obvykle dvakrát více, to závisí na počtu uživatelů).

U customizovaných systémů je podíl údržby pro jednotlivé instalace podstatně menší (platí se tím, že se zákazník musí přizpůsobovat



# Jaké techniky snižují pracnost údržby

- Vždy: Kvalitní architektura, kvalitní specifikace, kvalitní vývojové prostředí
- Corrective
  - Architektura SW, možnost agility
  - Moderní metody vývoje moderní vývojová prostředí, vodný jazyk
- Adaptive
  - PaaS, nástroje nezávislé na platformě
- Enhansive
  - SOA s hrubozrnným rozhraním, automatické testy, kvalitní specifikace

# Jak se projeví SaaS, software as a service

A také cloud computing

Platform as a service

Zatím ve fázi líbáněk

# Měření softwaru

# Výsledek měření je metrika

- Bez měření nelze kvalitně řídit ani hodnotit kvalitu SW, atributy kvality mohou ale být různé
- Softwarová metrika je nějaký údaj, který lze nakonec převést na číselné hodnocení nějakého atributu softwaru
- DB softwarových metrik je paměť firmy a prostředek optimalizace softwarových procesů (procesů vývoje softwaru) a managementu (např. méně rizik při uzavírání smluv),
- Je součástí business intelligence SW firmy a předpoklad uplatnění moderních způsobů řízení, např. CMMI

# Použití SW metrik

- a) **Výzkum:** podklad pro **hledání metod realizace** softwarových produktů, které by přinesly podstatné zvýšení jeho kvality a snížení nákladů a doby vývoje a hlavně rozsahu prací při údržbě softwaru (výzkum metod a zákonitostí vývoje softwaru).
- b) **Normy:** základ pro **stanovení technicko-ekonomických podkladů** pro řízení prací při tvorbě softwaru (normy pracnosti, odhady takových metrik, jako je pracnost či doba řešení) a uzavírání smluv (cena, termíny), předpoklad CMM
- c) **Kontrola kvality:** prostředek **sledování spolehlivosti** softwaru při provozu a podklad pro řídicí zásahy během údržby, procesy zajišťující kvalitu
- d) **Operativa:** prostředek **sledování průběhu prací** při vývoji (dodržování termínů, procento testovaných komponent, trendy počtů chyb, počty nově zanesených chyb, komponenty s největším počtem chyb, atd.).

# Použití SW metrik

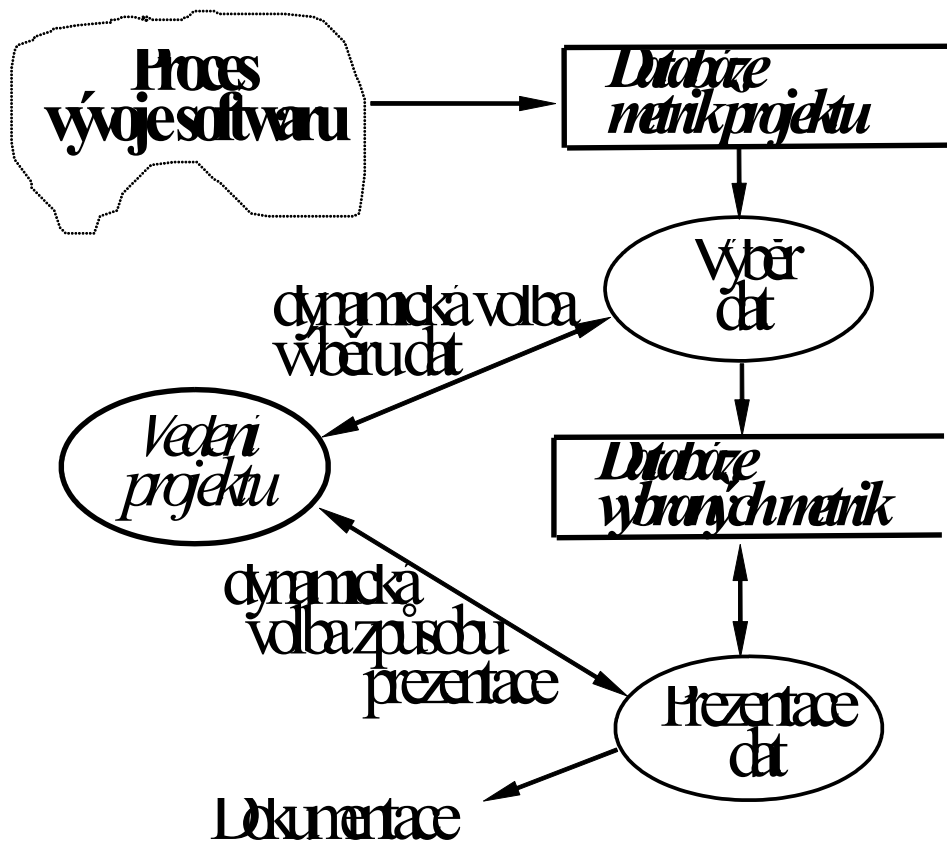
Výsledkem výzkumu SW metrik jsou metodiky vývoje (SOA, OO, strukturované programování, znalosti a vlivu kvality specifikací atd.) a metodiky odhadu pracnosti a doby řešení COCOMO, funkční body, a filosofií SOA ITIL

ISO 9126, ISO 250xx, ISO 12207, ISO 20000

ISO 250XX, **Software engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Planning and management**

ISO 270XX [Information Security Management System](#)

# Infrastruktura sběru metrik



# Problémy se systémem měření

Hlavním cílem je systém měření umožňující snadný přístup k metrikám a efektivní metody vyhledávání a vyhodnocování informací.

Cílem měření není pouze každodenní operativní dohled a kontrola.

- Systém orientovaný na kontrolu a dohled má tendenci ustrnout a nevyvíjet se. Navíc hrozí, že se nevyužijí možnosti, které systém nabízí pro vrcholové řízení.



# Problémy se systémem měření

- Systém měření nemá vyvolávat odpor, jinak je neefektivní
- Odpor může být způsoben nevhodnými opatřeními managementu.
  - Zviditelnění dobrých výsledků může vést management k rozhodnutí nadměrně redukovat zdroje pro daný úkol.
  - Zviditelnění nevýkonnosti může vést k posílení zdrojů, později však i k postihům. Pozdní posílení může být kontraproduktivní
- Je důležité, aby většina cítila, že jsou metriky užitečné a poctivé zvýhodňují a zvyšují šance na úspěch.

# Problémy se systémem měření

- Využívání metrik může být ztíženo krátkozrakostí a profesionálními předsudky (nekritické přeceňování účetnictví a finanční operativy atd.).
- Informace a rozhodnutí mohou složitým způsobem záviset na několika metrikách. Snaha o zjednodušení může vést k nesprávným závěrům. Management by měl své závěry konzultovat s členy týmu.
  - Chybovost modulu může být náhoda
- Efektivnost využití systému sběru a vyhodnocování metrik závisí na tom, do jaké míry bude všemi akceptován a kvalifikovaně užíván. Při tom lze využívat matematickou statistiku

# Vlastnosti systému měření

- Jasně definované cíle měření
  - U všech sbíraných dat musí být jasné, že jsou potenciálně užitečné pro všechny, byť se hned nevyužívají,
  - Jinak je sběr metrik chápán jako šikana
- Otevřenost, modifikovatelnost
  - Znalosti o metrikách (state of the art) se mění, mění se i potřeby managementu a standardy

# Vlastnosti systému měření

- Vychází z potřeb managementu
  - Integrální součást řízení
- Měření odděleno od vyhodnocování
- Lidé by měli cítit systém měření jako podporu jejich práce a ne jako hrozbu. Neměli by být dominováni systémem

# Kvalita dat

- V managementu se musí používat data, která nejsou zcela spolehlivá a relevantní
- Podpora managementu se stává hlavním úkolem informatiky. Operativa je už do značné míry vyřešeným úkolem (neplatí pro zábavu)

# Kvalita dat

<b>Kategorie</b>	<b>Dimenze</b>
Vnitřní, intrinsická (Intrinsic)	Přesnost (Accuracy) Objektivnost (Objectivity) Důvěryhodnost (Believability) Reputace (Reputation)
Dostupnost (Accessibility)	Dostupnost (Accessibility, též Availability) Bezpečnost přístupu (Access security)
Kontextuální (Contextual)	Relevantnost (Relevancy) Přínos (Value added) Včasnost (Timeliness) Úplnost (Completeness) Rozsah (Amount of data)
Reprezentační (Representational)	Interoperabilita (Interoperability) Srozumitelnost (Easy of Understanding) Výstižná a stručná reprezentace (Concise representation) Konsistentní reprezentace (Consistent representation)

# Problémy s kvalitou dat pro řízení

- Relevantnost a včasnost závisí na frekvenci zjišťování nebo na tom, jak je časově náročné data vytvořit (např. data rozvrhu)
- Kvalita dat může implikovat vytvoření datového úložiště v SOA, aby management mohl ovlivňovat chod systému

# Problémy s kvalitou dat pro řízení

- Kvalita dat může implikovat filosofii řešení
  - Kritická cesta a kritický řetězec
- Kvalitu je nutno měřit či odhadovat
- Kvalitu dat můžeme zlepšovat
  - Okrajová data, odstranění
  - Chybějící data pro parametry, pro regresi, doplnit
  - Opakovaná data, odstranit
  - Rozsah dat, měl by být dostatečný



# Přesnost SW metrik je malá

- Hodnoty metrik závisí
  - Na typu projektu, projekty se málo opakují (výjimka: SAP atp.)
    - Velikost projektu a ostrost termínů
    - Typu úlohy RT\*dávka bez přímých následků
  - Na stavu oboru
    - Výkon HW, sítě, vývojová prostředí,
    - paradigmatata

# Přesnost SW metrik je malá

- Hodnoty metrik závisí
  - Dosažitelné technologii
  - Kvalitě programátorů (produktivita 1:20)
    - Kvalitnější vývojáři píší lepší programy (rychlost 1:10 a to s méně chybami)
    - Programátoři jsou schopni silně ovlivnit určitou SW metriku, je-li jim dovoleno nebrat ohled na jiné metriky (rychlost a úkor délky)
  - Do jaké míry se řeší podobné problémy (viz minulý semestr)

# Přesnost SW metrik je malá (velký rozptyl)

- Výhrady k přesnosti metrik neplatí pro metriky sledující vlastnosti systému za provozu
  - Frekvence chyb/střední doba mezi poruchami
  - Počet stížností
  - Frekvence nalézání problémů a detekce defektů

# Pracnost závisí zejména na

- druh softwaru: míra interakce od dávkových až po tvrdé systémy reálného času, závažnost
- důsledků selhání, od nevýznamných škod, přes ekonomické ztráty až k ohrožení životů. V neposlední řadě
- velikosti systému;
- ostré až obtížně splnitelné termíny realizace;
- použití moderních projekčních technik a technik vývoje;
- kvalita zúčastněných;
- omezení hardwaru a softwaru. Pokud systém využívá zdroje jako je rychlost a paměť více než na dvě třetiny, vzrůstá značně pracnost řešení.

# Interní a externí metriky (ISO 9126, ISO 250XX)

- Též implicitní a explicitní metriky (in process, after process)
- Interní – známo jen týmu během vývoje na základě sledování vývojových procesů: např. spotřeba práce, a také doba řešení
- Externí: dají se zjistit na hotovém produktu, např. délka
- Některé metriky je možné chápat obojím způsobem (počet selhání při testování, externí – zjištěno uživateli)

# Datové typy SW metrik

- Příslušnost k třídě (id. - číslo tramvaje)
  - Trendy počtů objektů s daným id, operace rovnost =)
- Fuzzy (id hodnocení, dobrý, lepší, nejlepší, známky ve škole)
  - Operace = test na rovnost, < test na „velikost“. Obvykle se převádí na číselné hodnocení. Př. COCOMO, známky ve škole (tam se používají přímo fuzzy hodnoty ve formě čísel)

# Datové typy SW metrik

- Interval
  - Čas, teplota
  - Je možná lineární transformace, =, <
  - Využitelný je rozdíl jako číselná metrika
- Číselná metrika
  - Délka programu, doba řešení
  - Jsou smysluplné všechny operace pro reálná čísla

# ISO 9126 (4 části)

- Stovky metrik, rozumně použitelné většinou jen u velkých firem
- Není jasné, k čemu se to všechno použije
- Jádro (principy a některé metriky) použitelné
- Modernizace normy - sada norem 250xx
  - Základní principy a základní metriky stejné



# Externí metriky

- *Del* – délka produktu v řádcích. U programů se nepočítají komentáře. *Del* programů se někdy udává v lexikálních atomech. Do metriky *Del* se někdy nezahrnují deklarace proměnných a záhlaví podprogramů.
- *Srnd* – rozsah slovníku operandů. Tato metrika se týká programů. Operand je buď konstanta (např. celé číslo 10, nebo řetězec znaků „xyz“, v terminologii programování literál), nebo proměnná, např. *x*. *Srnd* je pak počet logicky odlišných operandů vyskytujících se v programu.

# Externí metriky

- *Nrnd* – počet výskytů operandů v programech
- *Noper* – Počet výskytů delimiterů a znaků operací a podprogramů v programech.
- *Soper* – rozsah slovníku operací, podprogramů a delimiterů. Tato metrika udává, kolik program obsahuje významem různých znaků operací (\*, C, atd.), jmen podprogramů (sin, tan, put), delimiterů (: **if begin real** atd.).

# Externí metriky

*Users* – Maximální počet uživatelů, pro které je systém plánován.

*McCabe* – počet podmíněných příkazů (if), příkazů cyklu (for, while, . . . ) a přepínačů (case) v programu. Metrika *McCabe* je dobrým indikátorem složitosti programů. Jejím nedostatkem je, že není citlivá na hloubku a způsob vložnosti podmíněných příkazů (tvar rozhodovacího stromu (případně lesa)).

# Externí metriky

*In, Out, Qer, File, Filee* – složitost příkazů vstupu, výstupu, dotazů na terminál a operací se soubory interními a se soubory společnými s jinými aplikacemi. U databází složitost SQL dotazů. Tyto metriky se používají v odhadech pracnosti a doby řešení v metodě funkčních bodů.

*FanIn, FanOut* – míry indikující složitost rozhraní tříd, modulů a aplikací. *FanIn* udává počet logicky různých

typů dat vstupujících do dané entity (např. modulu).

*FanOut* je obdoba *FanIn* pro vystupující datové toky.

Často se používá metrika  $Fan1 = \sum_{modul\ i} (FanIn_i * FanOut_i)^2$ .

# Problém metrik pro SOA

- Obtíže s měřením složitosti komunikace,
- Jak měřit složitost hrubozrnných zpráv
-

# Interní metriky

- *Prac* – spotřeba práce, obvykle v člověkoměsících
- *Doba* – doba provádění
- *Prod* – jednotky délky za člověkoměsíc (řádky za měsíc)
- *Fail* – počet selhání za týden (den)

# Interní metriky

- *Prod* – produktivita, počet jednotek délky vytvořených za člověkoměsíc.
- *team(t)* – velikost týmu (počet osob) v čase  $t$ , měřeno od začátku prací. Tato metrika umožňuje postupné zpřesňování odhadů pracnosti a doby řešení během vývoje projektu.
- *Team* – průměrná velikost týmu.

# Interní metriky

*Fail(t,p)* – počet selhání systému /části *p* detekovaných při testování či provozu v čase *t*. Při provozu hlásí selhání zákazníci. Obvykle se udává po dnech nebo týdnech. Tato metrika je důležitou mírou kvality. Při inspekcích je hodnota metriky *Fail* dána počtem chyb zjištěných při inspekci.



# Interní metriky

*Defect(t,p)* – počet míst v programech, která bylo nutno opravit pro odstranění selhání nebo pro nápravu selhání v čase  $t$  (den, týden) v části  $p$ , normalizovaný pro 1 000 řádků (1000 \_ počet defektů\_délka).

*Defect1(e1,e2,t,p)* – počet defektů v části  $p$  vzniklých v etapě řešení  $e1$  a zjištěných v etapě  $e2$  v době  $t$ . Tato metrika a metriky z ní odvozené jsou účinnou mírou efektivnosti inspekcí a testů. Důležitá externí metrika pro vyhodnocování kvality produktu

# Interní metriky

*Satisf(t)* – průměrná míra spokojenosti zákazníků včase  $t$ . Spokojenost se udává ve stupnici 1 až 5 (nejlepší). Lze vyhodnocovat trendy. Existují metody odhadu vývoje úspěšnosti produktu na trhu z aktuálních hodnot metrik *Satisf* (Babich, 1992).

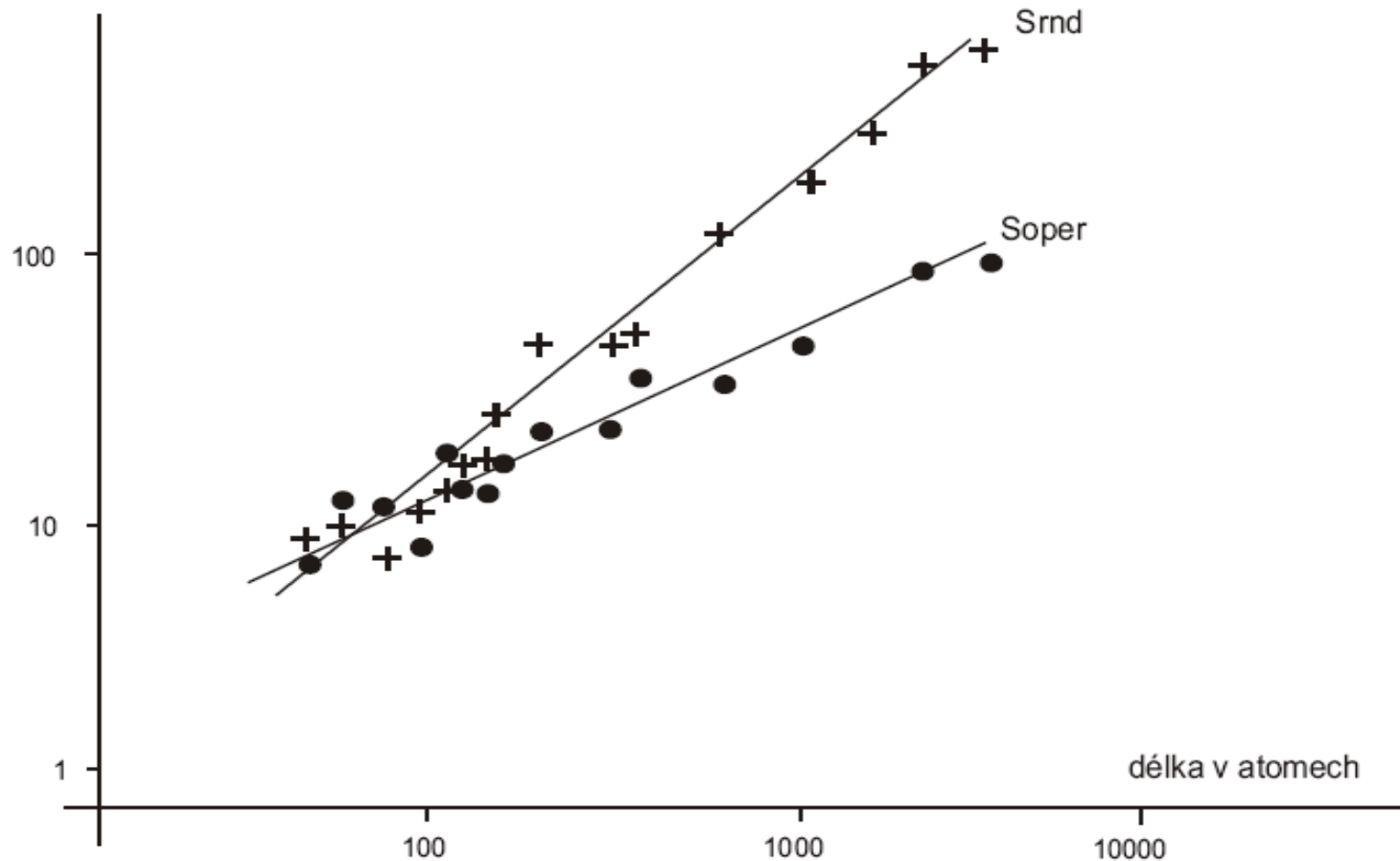
*DobaOpr* – průměrná doba opravy selhání.

# Interní metriky

$Zmeny(f,t)$  – počet změněných míst souboru  $f$  včase  $t$  (týdnu/dni). Tato metrika se snadno zjišťuje a její trendy mohou v průběhu prací poskytnout cenné informace.

$MTBF(t)$  – (Mean Time Between Failures): střední doba mezi poruchami (v určitém období, např. týdnu,  $t$ ).

# Vztah mezi rozsahy slovníků operandů a operací



# Výpočet charakteristik programu

	<i>Del</i>	<i>Noper</i>	<i>Nrnd</i>	<i>Soper</i>	<i>Srnd</i>
<code>begin</code>	1	1		1	
<code>var x,y:real;</code>	7	5	2	5	2
<code>x:= y*2.0+sin(x)+2.0</code>	12	7	5	5	1
<code>end.</code>	1	1		1	
<b>Celkem</b>	<b>21</b>	<b>14</b>	<b>7</b>	<b>12</b>	<b>3</b>

# Halsteadův vztah pro malé programy

Odvozeno z analýzy počtu rozhodnutí při psaní

$$Prac \cong C Del Nrnd (Soper/Srnd) \log(Soper+Srnd)$$

V dobře napsaných programech jsou  $Srnd$  a  $Nrnd$  úměrné délce. Pro velké  $Soper$  a  $Srnd$  lze logaritmus nahradit konstantou. Pak ale bude

$$Prac \cong C' Del Soper$$

# Halsteadův vztah pro malé programy

Poněvadž je pozorováno, že

$$Soper \cong C \cdot Del^b$$

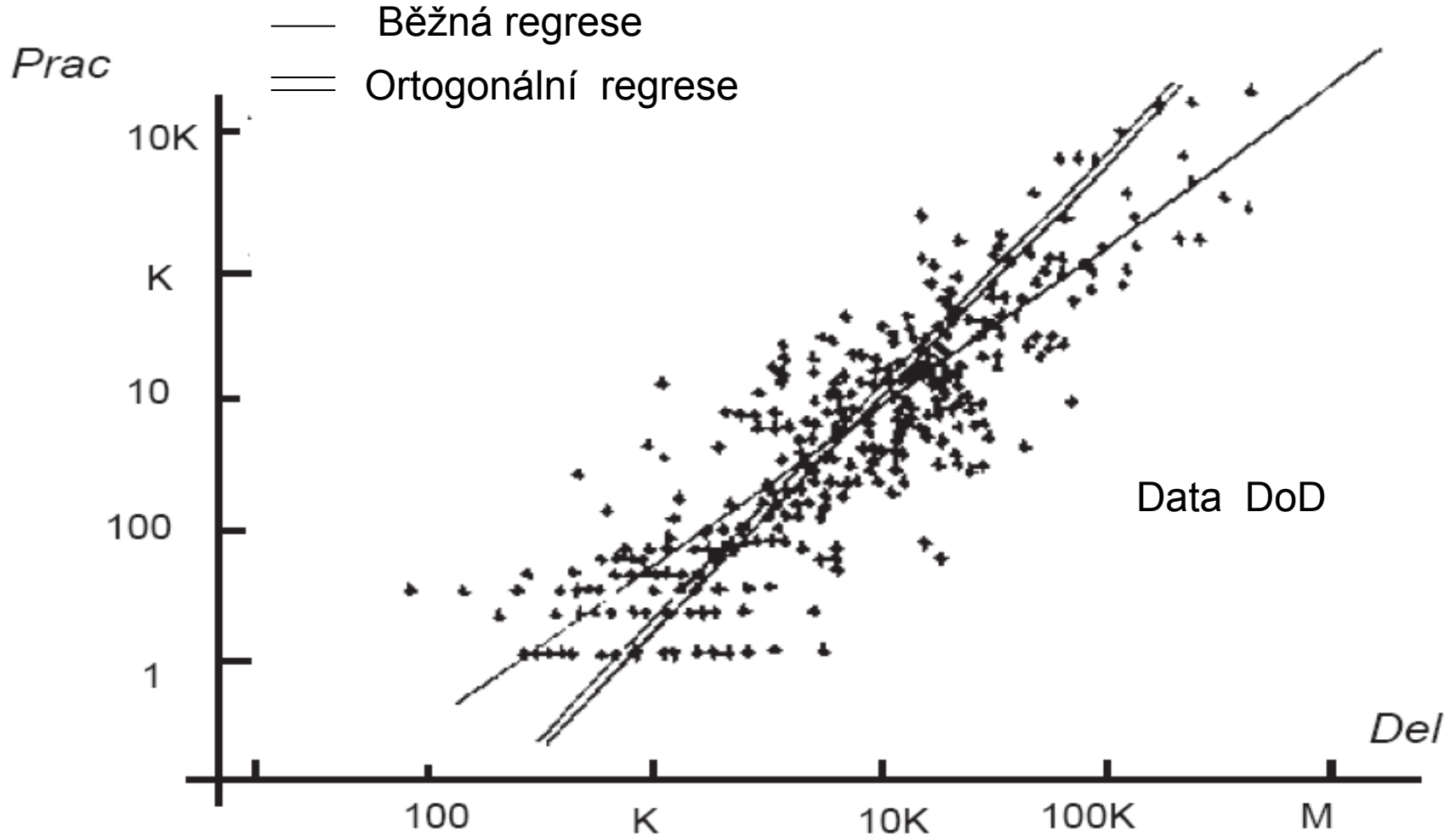
dostaneme

$$Prac = c Del^{1+b} \quad b \cong 0,25$$

Pro velké programy je hodnota  $b$  příliš vysoká. Je to důsledek toho že vztah modifikujtové technologie, jako dekompozice a znovupoužití částí.

Pokud se dekomponuje do malých komponent a systémy se liší jen počtem komponent a transport zpráv se nemusí pracně implementovat, bude  $b$  velmi malé.

# Závislost pracnosti na délce programu

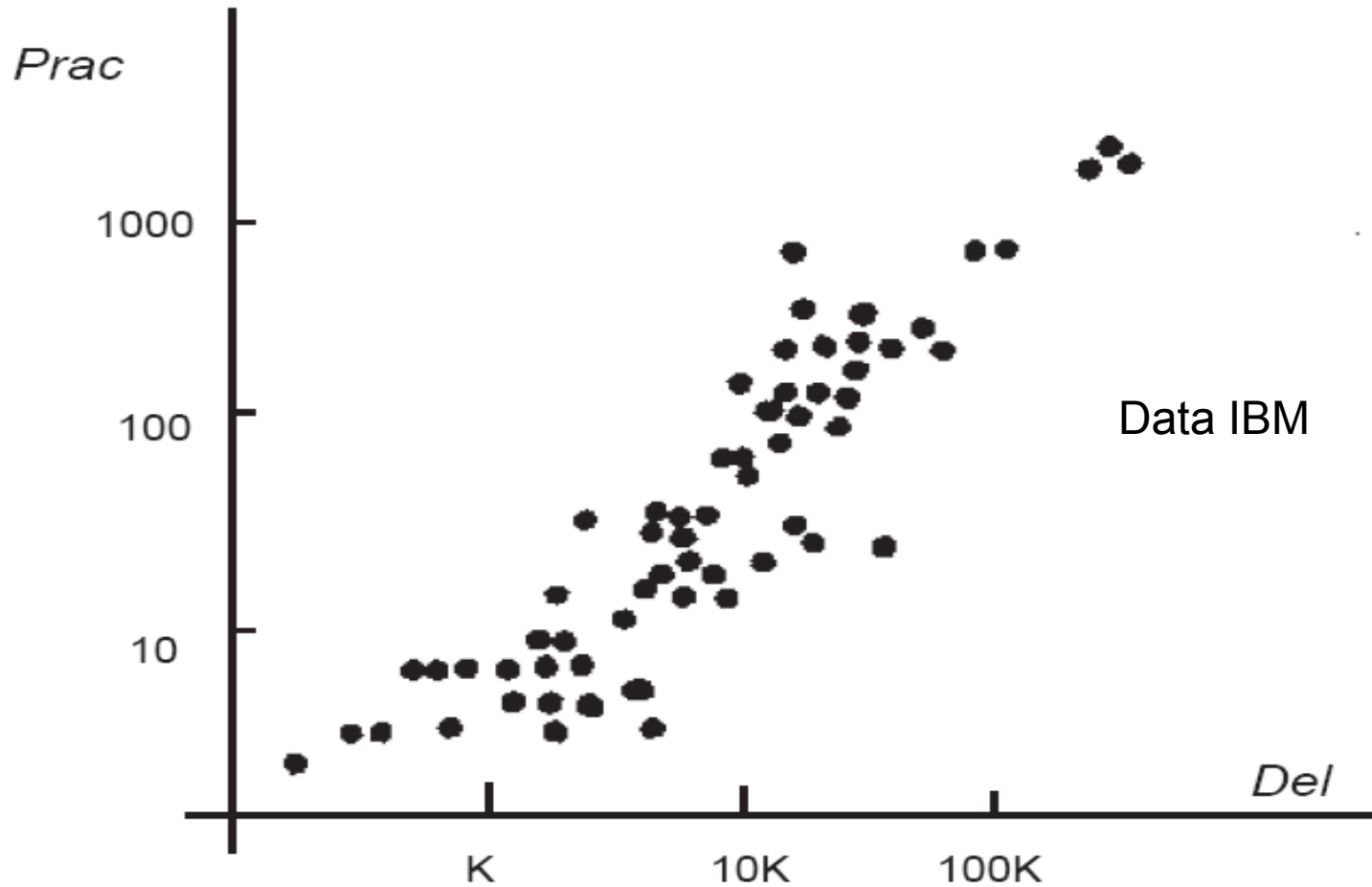


10

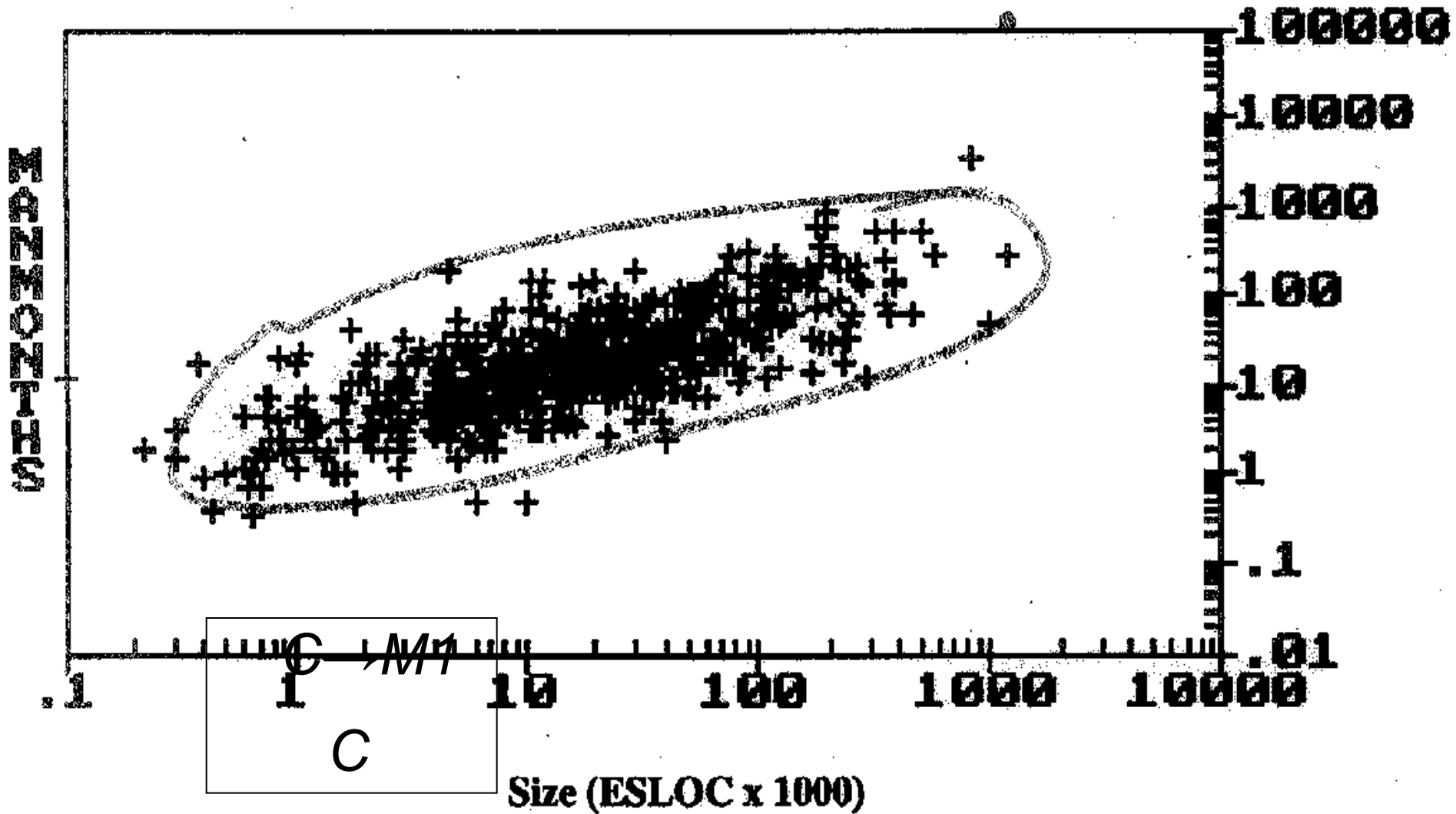
a)



# Závislost pracnosti na délce programu

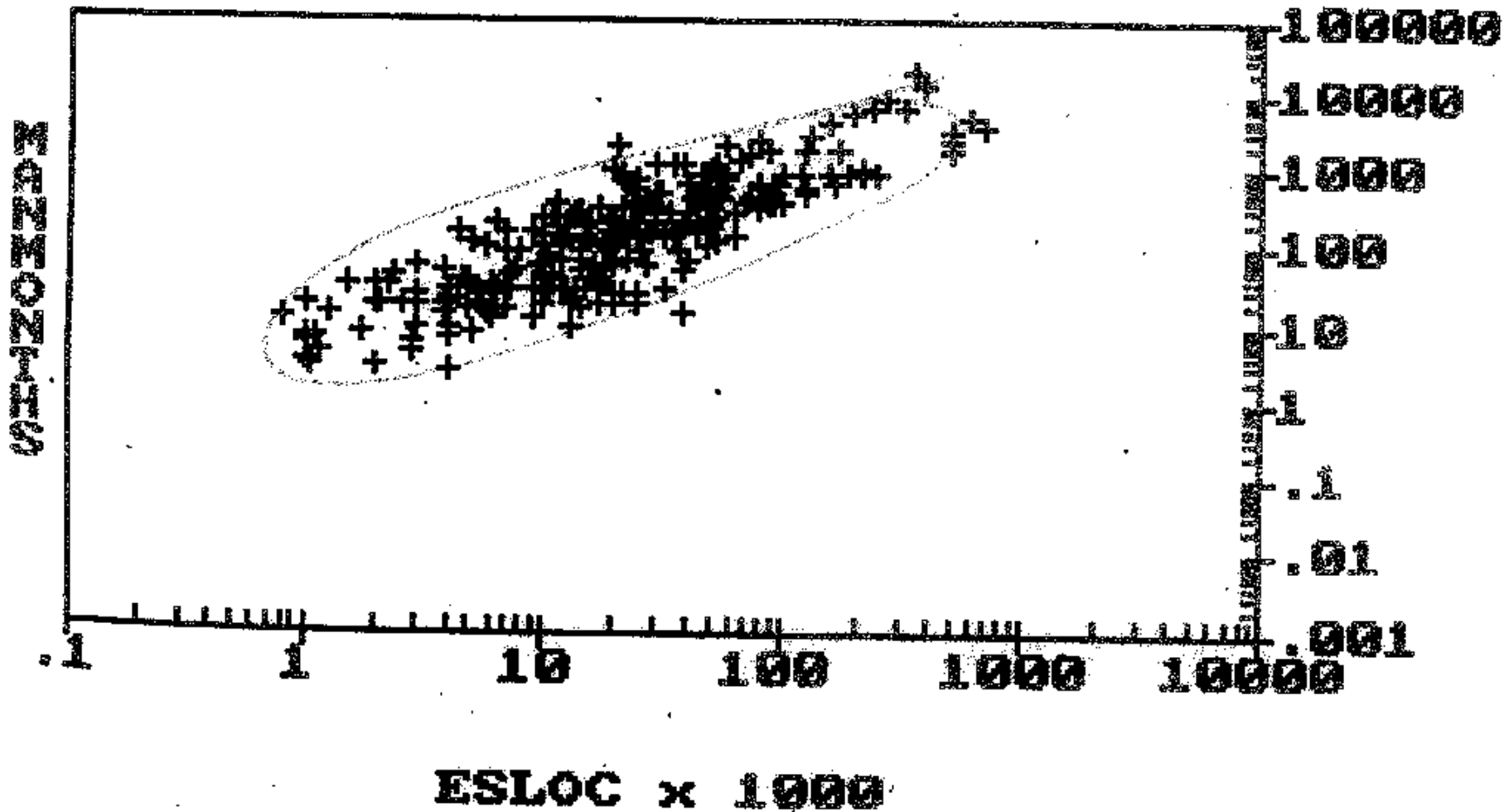


# Business Systems



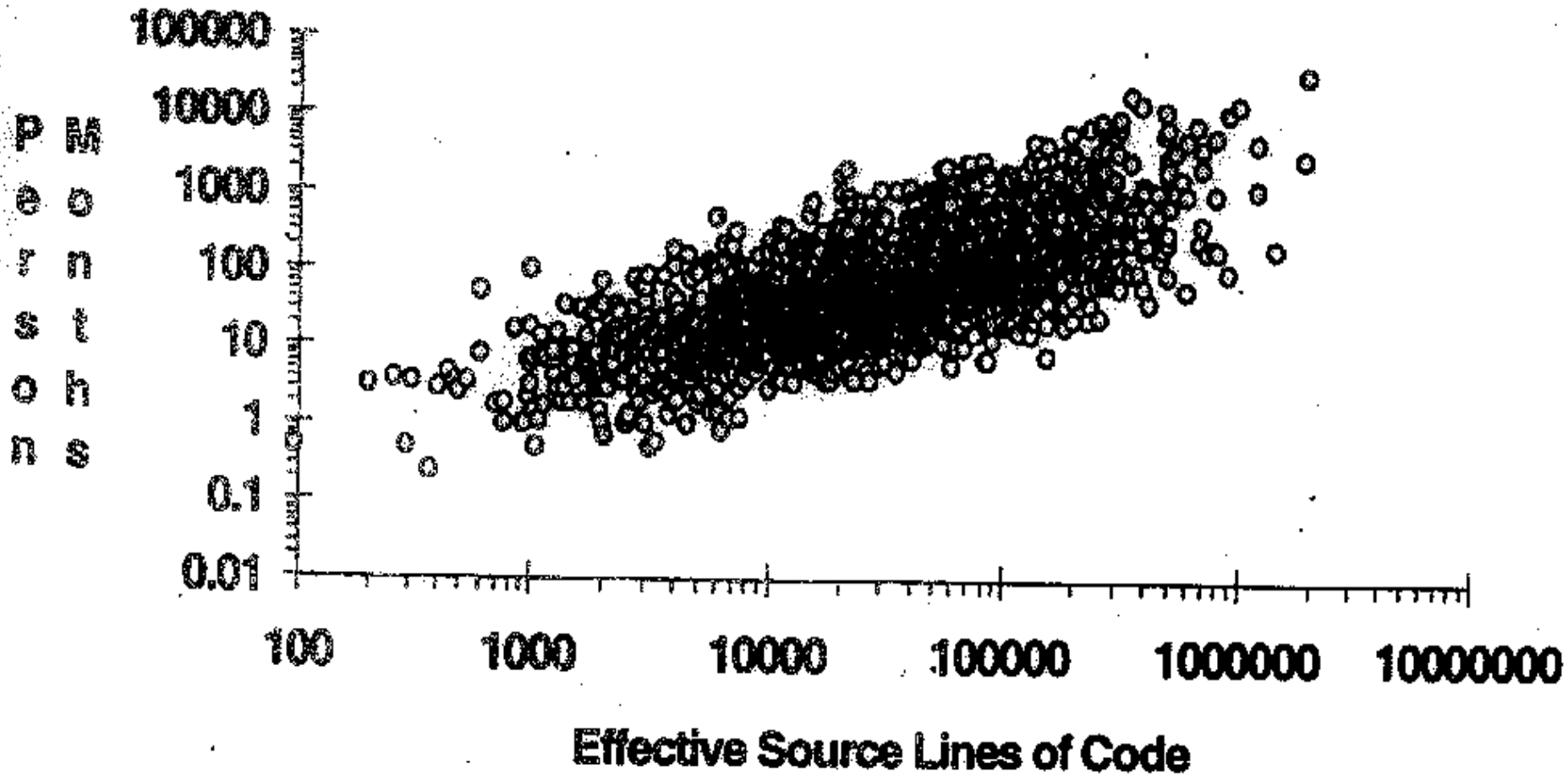
# Effort vs. Size

## Real Time Systems

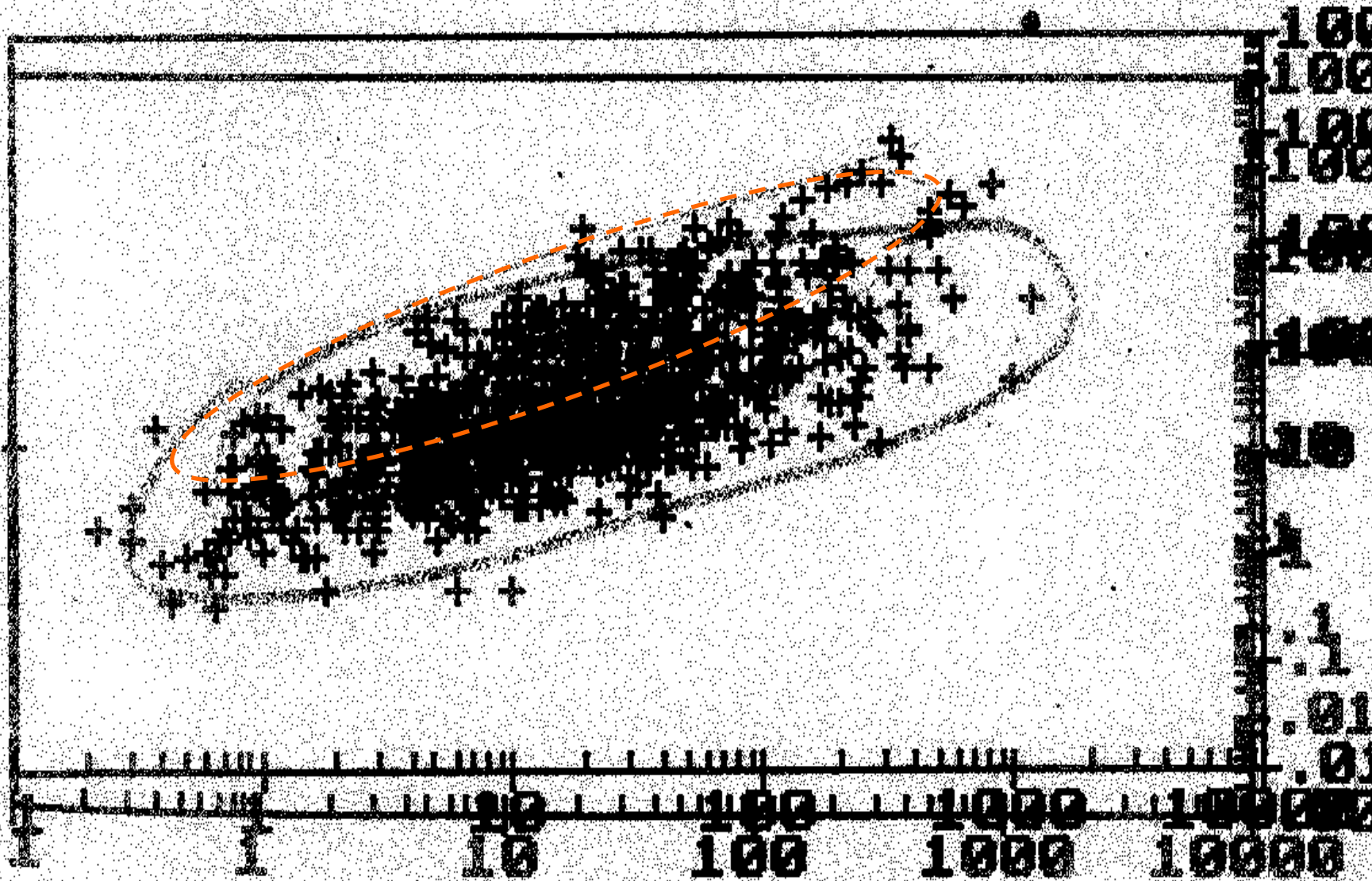


# Effort Behavior

## QSM Mixed Application Data Base



sh sa prekladá najmä s koule



Size (ESLOC x 1000)

# Pozor na statistiku

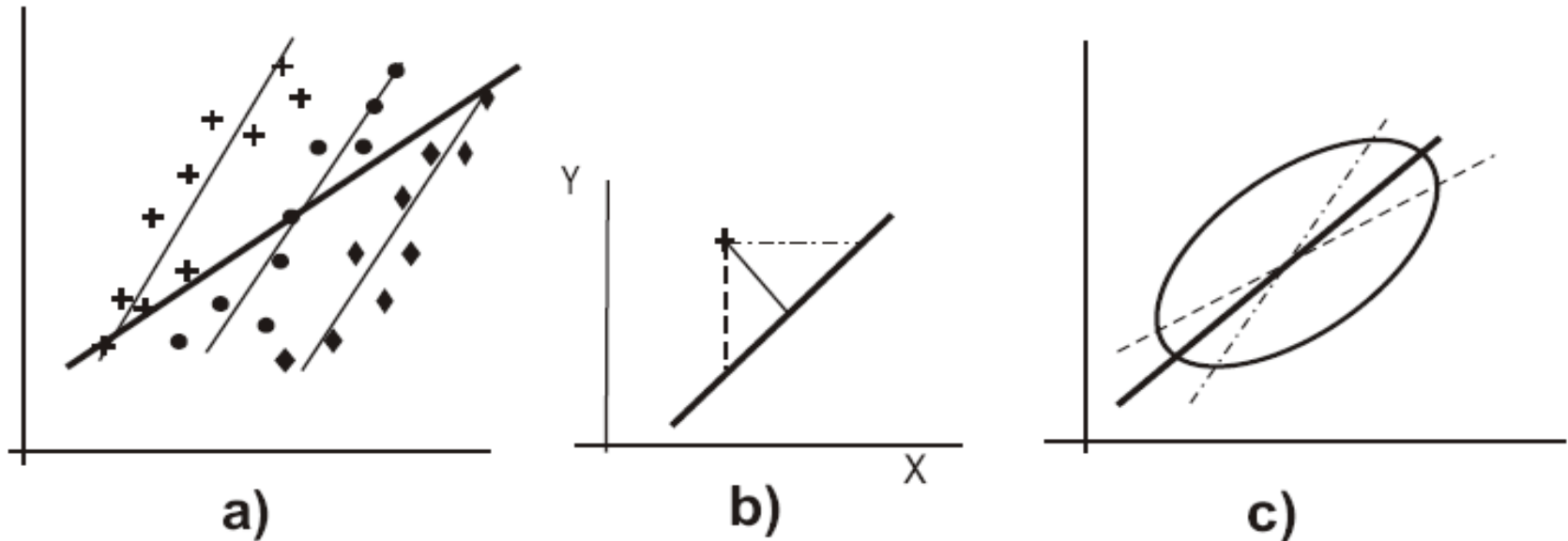
- Analýzou dat klasickou regresí dostaneme

$$Prac = c_{10} Del^d$$

Kde  $d < 1$ , což není zřejmě reálné, neboť jedna instrukce ve velkém systému dá podle zkušeností více práce (např. díky nákladům na chod týmu, než jedna instrukce v malém systému.

Je nutné použít ortogonální regresí.

# Problémy s regresí



**Vysvětlení rozporných výsledků analýzy regrese.**

a) Regresní přímky pro jednotlivé týmy (tence) mají větší sklon než regresní přímka pro data všech týmů (silná čára).

b) ——— odchylka uvažovaná při ortogonální regresi,  
..... odchylka uvažovaná při standardní regresi,  
- - - - - odchylka uvažovaná, je-li Y nezávisle proměnná.

c) Regresní přímky pro data pokrývající elipsu. Čím je elipsa širší, tím je menší sklon regresní přímky pro standardní regresi a X jako nezávislou proměnnou.

# Pro obchodní systémy ani to nevysvětluje nízký koeficient regrese

- Možnost: Velké systémy více používají různé nástroje, jako jsou generátory kódu, CASE systémy, atd. Generovaný kód je „řinčí“



# Prvá zákonitost

Pro délku a pracnost platí vztah

$$\log(Prac) = (1+a)\log(Del) + c'$$

takže

$$Prac = c Del^{1+a}$$

Ortogonalní regrese dává

$$a \cong 1/8$$

# Prvá zákonitost

Pro délku a pracnost platí vztah

$$\log(\textit{Prac}) = (1+a)\log(\textit{Del}) + c'$$

takže

$$\textit{Prac} = c \textit{Del}^{1+a}$$

Ortogonalní regrese dává

$$a \cong 1/8$$

Avšak  $c$  i  $a$  závisí na typu projektu

# Použití a důsledky

- Vztah  $Prac = c Del^{1+a}$  se využívá v odhadu COCOMO
- Pokud je pracnost budování middleware zanedbatelná klesne pracnost monolitního systému po dekompozici na  $n$  zhruba stejných dílů z  $Prac_1 = c Del^{1+a}$  na

$$Prac_n = c n(Del/n)^{1+a} = n^{-a} Prac_1$$

# Empirické výsledky

	Zdroj	Nezávisle proměnná $X$	Závisle proměnná $Y$	koeficient regrese	koeficient korelace
1.	Norden, 1978	$\log(Del)$	$\log(Prac)$	1.125	0.85
2.	Walston, Felix, 1977	$\log(Del)$	$\log(Prac)$	1.125	0.80
3.	Putnam, 1978	$\log(P/D^2)$	$\log(Prod)$	-0.66	-0.98
4.	Walston, Felix, 1977	$\log(Del)$	$\log(Doba)$	0.44	0.62
5.	Walston, Felix, 1977	$\log(Prac)$	$\log(Doba)$	0.40	0.77
6.	Walston, Felix, 1977	$\log(Prac)$	$\log(Team)$	0.62	0.82
7.	Christensen, Fitsos, 1981	$\log(Srnd)$	$\log(Soper)$	0.33	0.78
8.	Fitsos, 1980	$\log(Srnd)$	$\log(Soper)$	0.48	0.69
9.	Wolberg, 1981	$\log(Srnd)$	$\log(Soper)$	0.45	0.88
10.	Wolberg, 1981	$\log(Srnd)$	$\log(Soper)$	0.29	0.86
11.	Halstead, 1977	$Del$	$Nrnd$	1.00	0.99
12.	Halstead, 1977	$\log(Del)$	$\log(Srnd)$	0.97	0.92
14.	Halstead, 1977	$\log(Del)$	$\log(Srnd)$	0.75	0.70
15.	Halstead, 1977	$\log(Del)$	$\log(Soper)$	0.48	0.50
16.	Halstead, 1977	$\log(Del)$	$\log(Soper)$	0.42	0.30

Tab. 15.4: Tabulka výsledků ortogonální regrese pro různé soubory dat. Data řádků 9 až 1 se týkají malých podprogramů.

# Putnamova rovnice

Podle třetí řádky tabulky

$$\log(Prod) \hat{=} c_{25} - 0.67 \cdot \log(Prac/Doba^2)$$

čili

$$Prod \hat{=} c_{26} Prac^{-2/3} Doba^{4/3}$$

Z definice  $Del = Prac Doba$ , takže

$$Del \hat{=} c_{26} Prac^{1/3} Doba^{4/3}$$

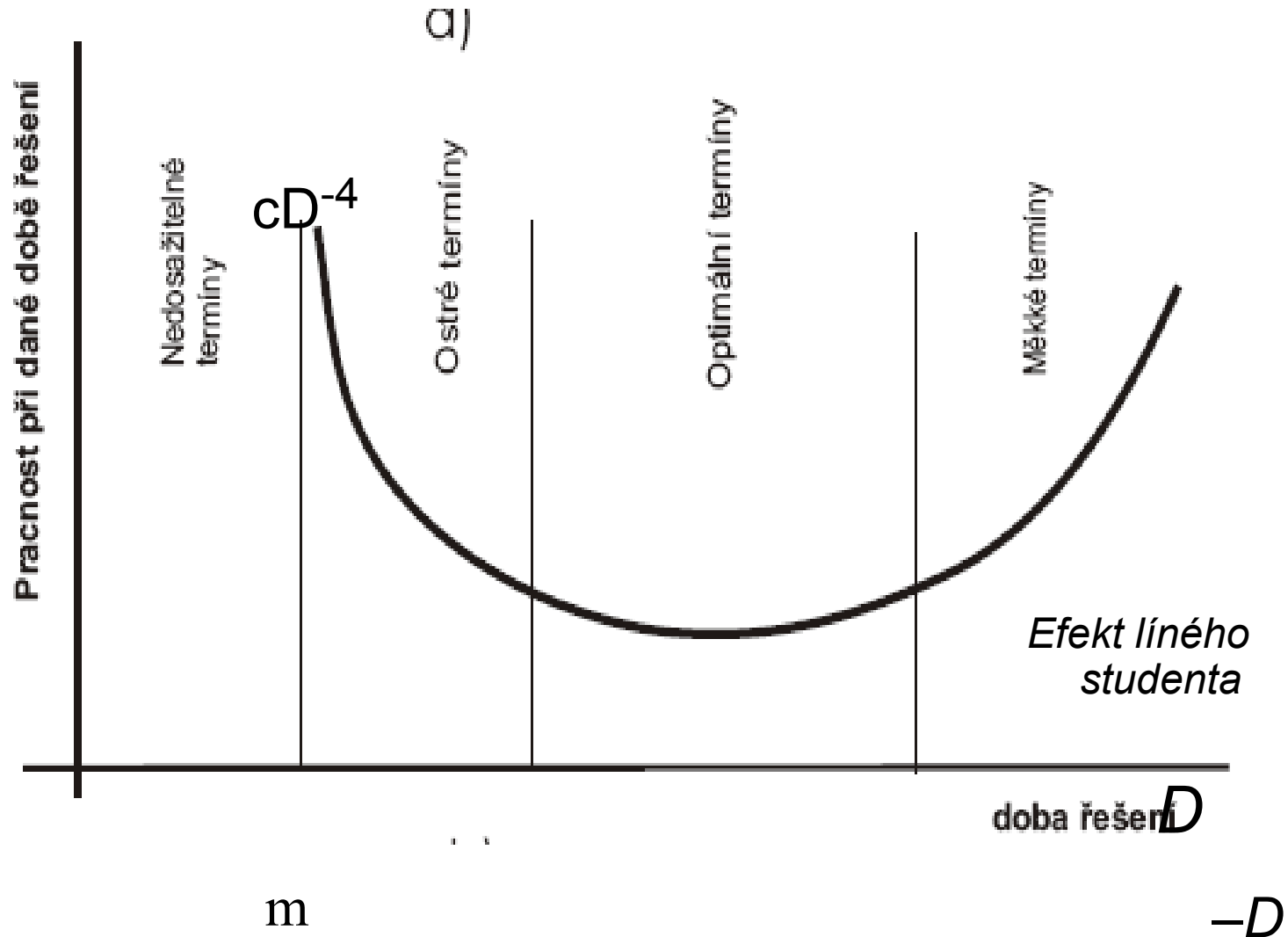
To je Putnamova rovnice

# Vliv napjatých termínů

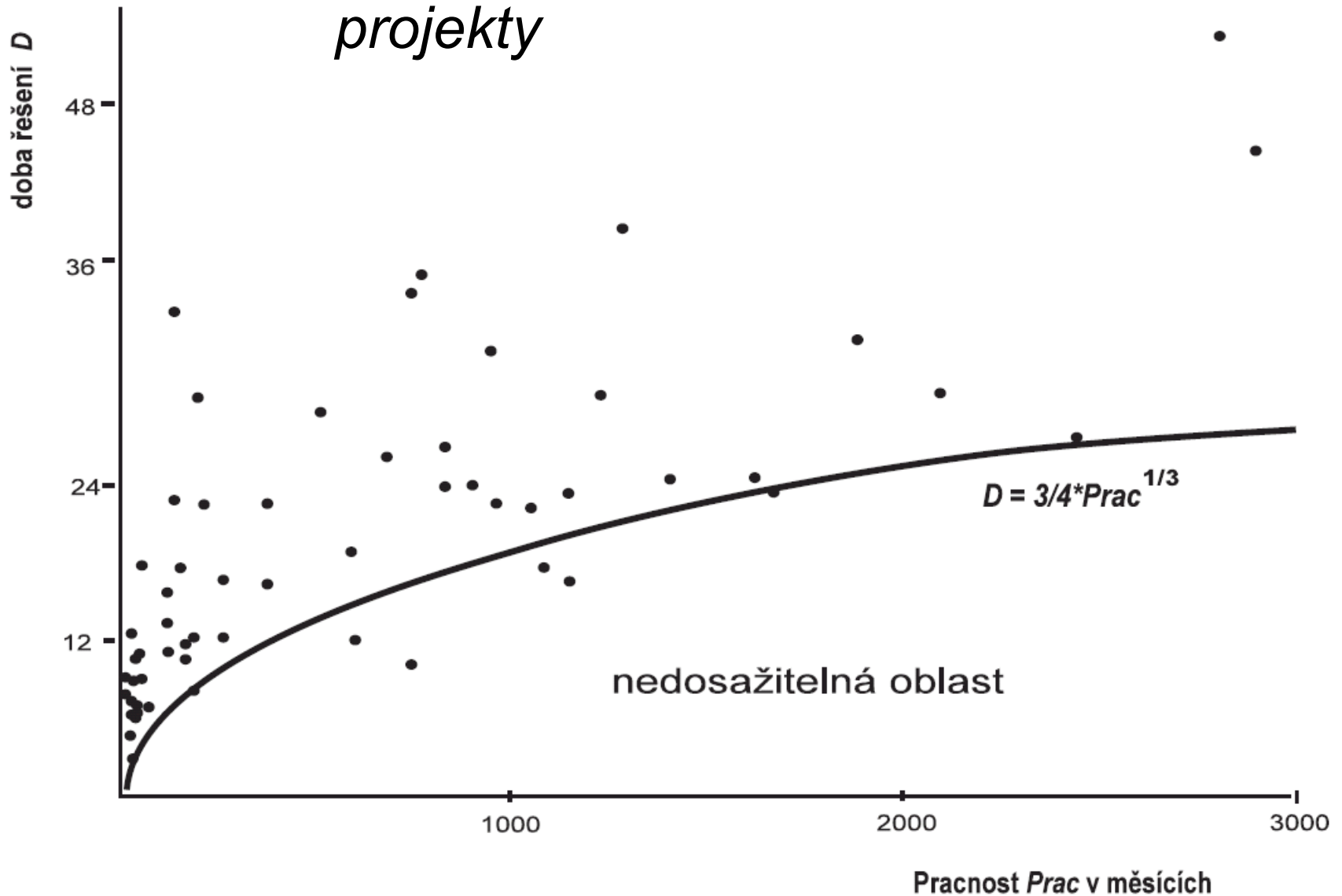
- Dobu řešení nelze v praxi libovolně zkracovat,
- Pro každý projekt existuje tedy mez  $m$ , pod níž se nelze prakticky dostat. Interval  $\langle 0, m \rangle$  se nazývá nedosažitelná oblast pro daný projekt.  $m$  je funkcí  $Prac$
- Pro více projektů je nedosažitelná oblast oblast roviny  $(Prac, Doba)$ , kde

$$Doba < \frac{3}{4} Prac^{1/3}$$

# Pracnost u nedosažitelné oblasti



# Některé starší výsledky pro SW projekty





# Chování pracnosti u nedosažitelné oblasti

Uvažujme dvě realizace  $A, B$  stejného projektu s atributy

$$Del_A, Doba_A, Prac_A$$

$$Del_B, Doba_B, Prac_B.$$

Vydělením Putnamových rovnic pro obě realizace dostaneme

$$Del_A/Del_B = (Prac_A/Prac_B)^{1/3} \cdot (Doba_A/Doba_B)^{4/3}$$

# Chování pracnosti u nedosažitelné oblasti

Nechť  $Doba_A > Doba_B$ . Poněvadž programy psané ve spěchu bývají delší je možné předpokládat  $Del_A \leq Del_B$  tj.  $Del_A / Del_B \leq 1$

Po úpravách dostaneme z poslední rovnice

$$1 \geq Del_A / Del_B = (Prac_A / Prac_B)^{1/3} (Doba_A / Doba_B)^{4/3}$$

Z toho po úpravách, považujeme-li hodnoty s indexem  $A$  za konstantní dostaneme obdobu Stefan-Botzmannova zákona

$$Prac_B \geq c_{30} Doba_B^{-4}$$

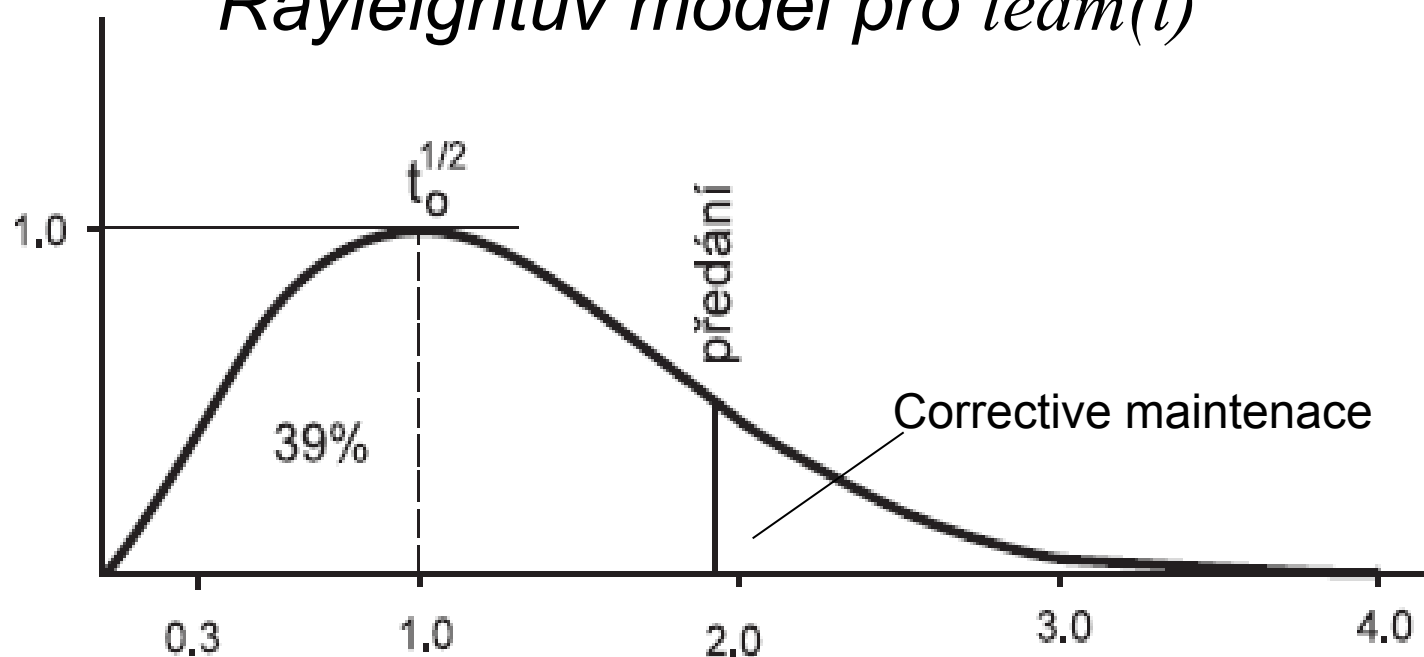
# Použití ve function points

Zkrácení termínu o šestinu prodlouží pracnost dvakrát

$$Prac_A \doteq Prac_B \cdot (6/5)^4 \doteq 2 \cdot Prac_B.$$

Ale zkrácení o polovinu zvýší pracnost šestnáctkrát. Metodika Function points nemá opravu na zkracování termínů tak drastickou, zkrácení na polovinu ale nepovažuje za možné.

## Rayleightův model pro $team(t)$



Obr. 15.12: Rayleightova křivka. Část plochy pod křivkou po předání jsou práce, které budou provedeny v rámci údržby (corrective maintenance). To, co se do okamžiku předání nestihne udělat, přechází do údržby, kde se projevuje jako neodstraněné chyby.

$$team(t) \hat{=} K \cdot t/t_0 \cdot \exp\left(-\frac{t^2}{2t_0}\right)$$

# Kritika Rayleightova modelu

- Příliš rychle klesá v nekonečnu, potom by ale byl SW bez chyb možný a to se nepozoruje
- Množství práce do maxima je příliš veliké, neodpovídá datům o corrective maintenance (40% pracnosti vývoje)
- Nevysvětluje, proč platí pro SW Stefan-Boltzmannův zákon
- Má málo parametrů a změna parametrů nemění příliš tvar
- Proto si půjčíme Planckův zákon

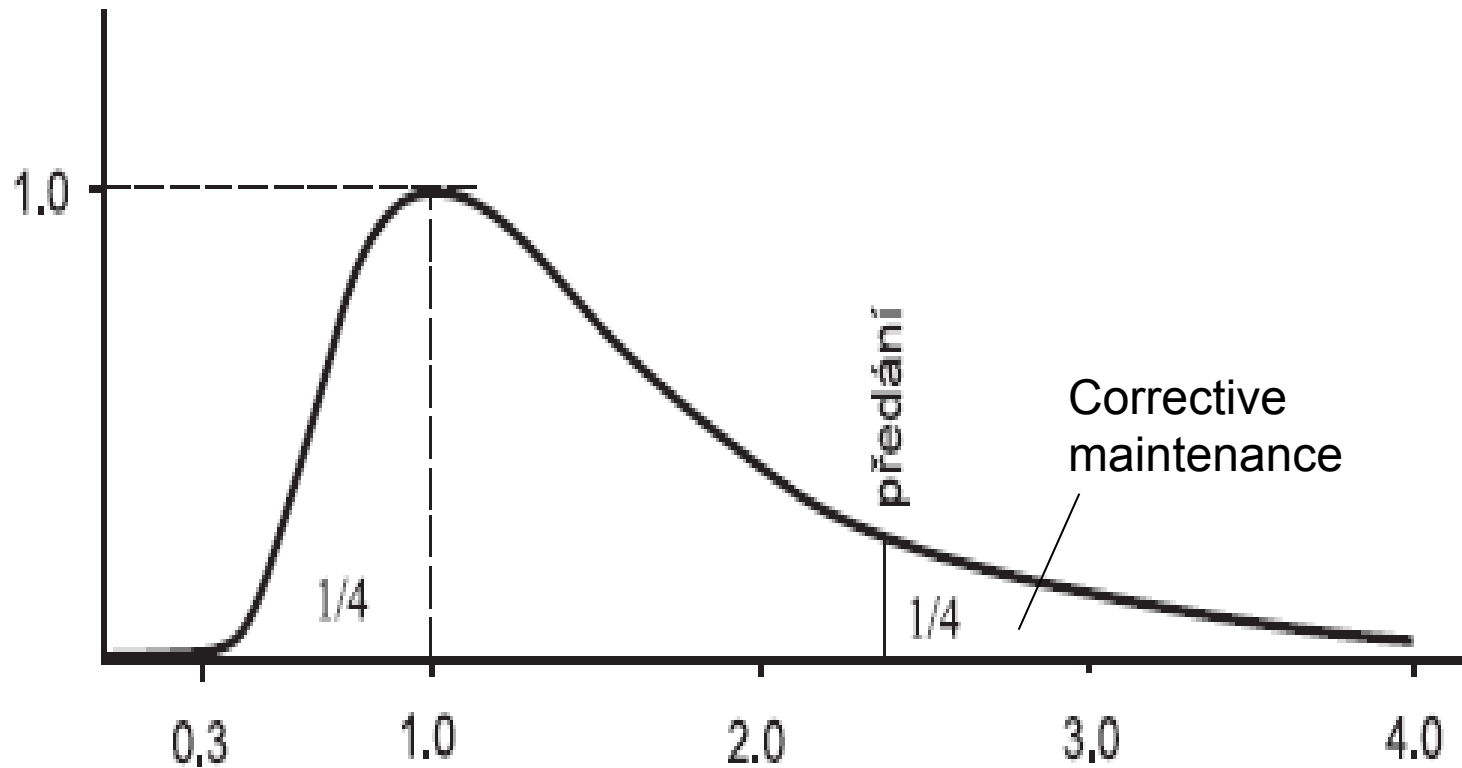
# Planckův model

- Standardní tvar zákona záření absolutně černého tělesa

$$Team(t) = \frac{Ct^5}{exp(D/t)-1}$$

D určuje tvar křivky a polohu maxima

## Planckův model



Obr. 15.13: Normalizovaný tvar Planckovy křivky  $Planck(t) = 142.32 \cdot t^{-5} / (\exp(4.9651/t) - 1)$ .

# Kritika Planckova modelu

- Začíná růst až od  $1/3$  (Existují názory, že je to OK, pokud do  $team(t)$  nezahrnujeme práce na specifikacích).
- Má také málo parametrů
  - A. Zavedeme lineární transformaci nezávisle proměnné
  - B. Změníme exponent 5 v polynomiální části



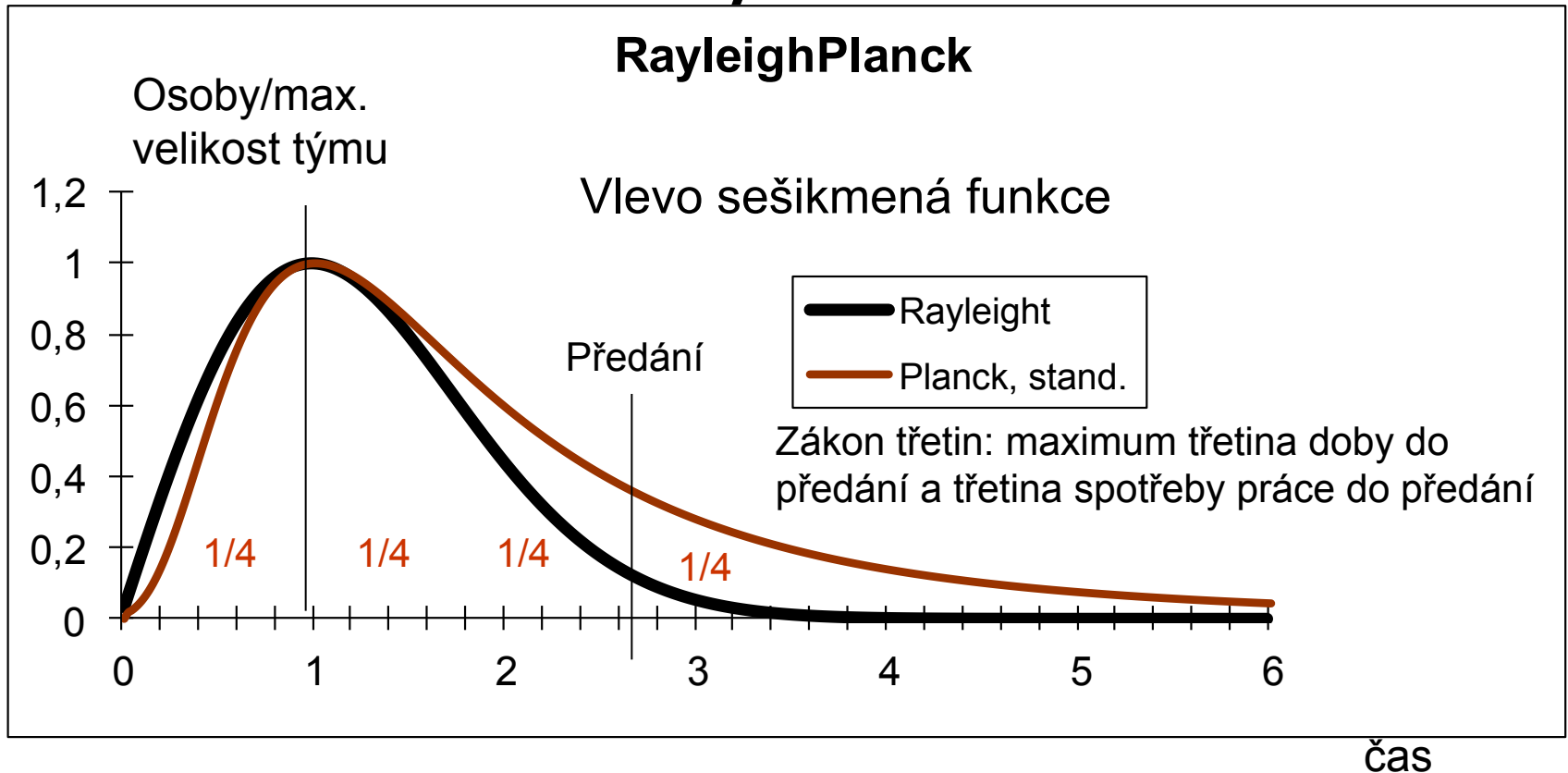
# Modifikace Planckova modelu

A. Transformace nezávislé proměnné

$$team(T) = \frac{c \cdot d^5 (T + k \cdot d)^{-5}}{\exp\left(\frac{D \cdot d}{T + k \cdot d}\right) - 1}$$

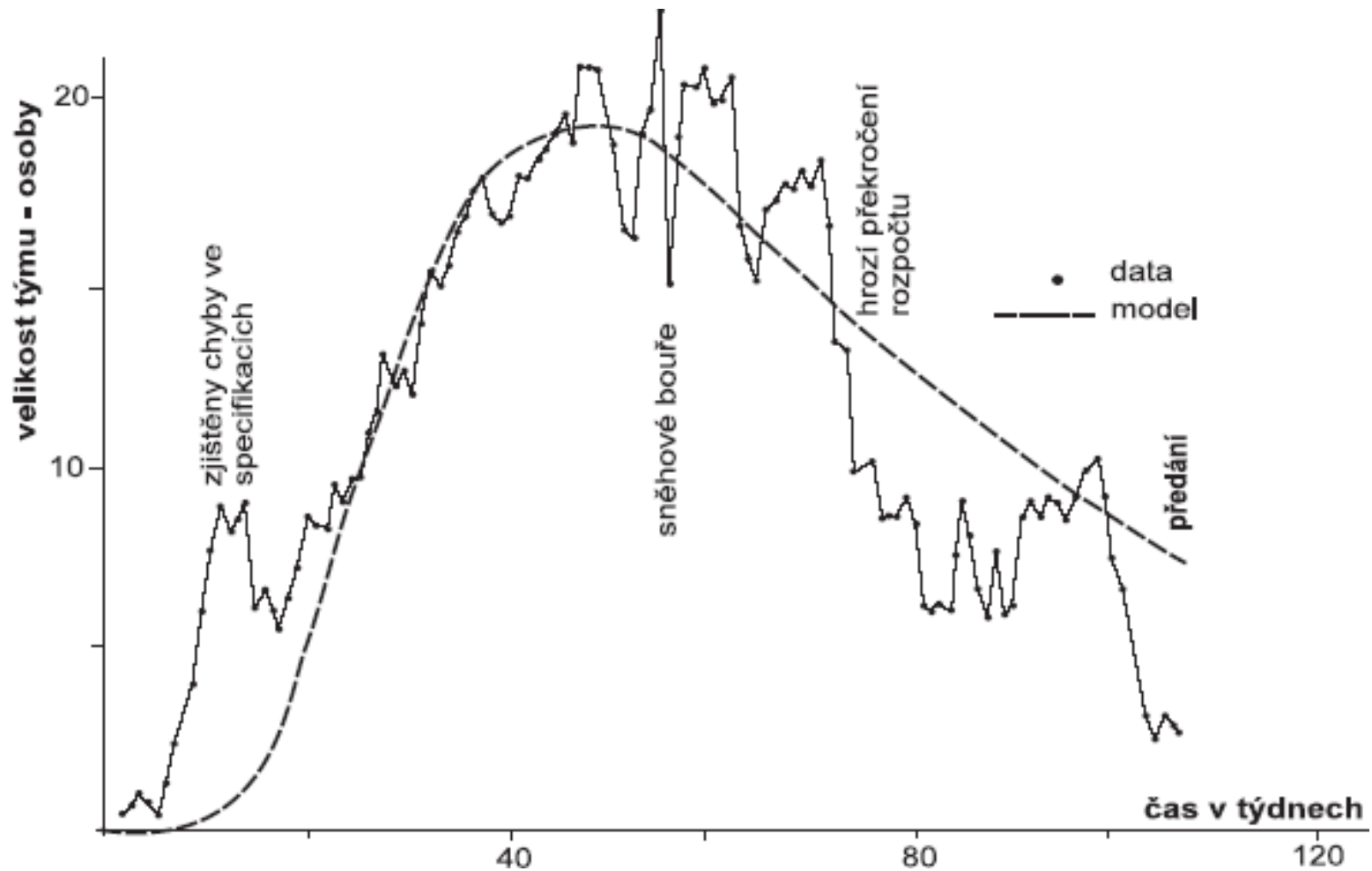
Dává dobré výsledky

# Najímaný tým, vrchol a odhad doby řešení

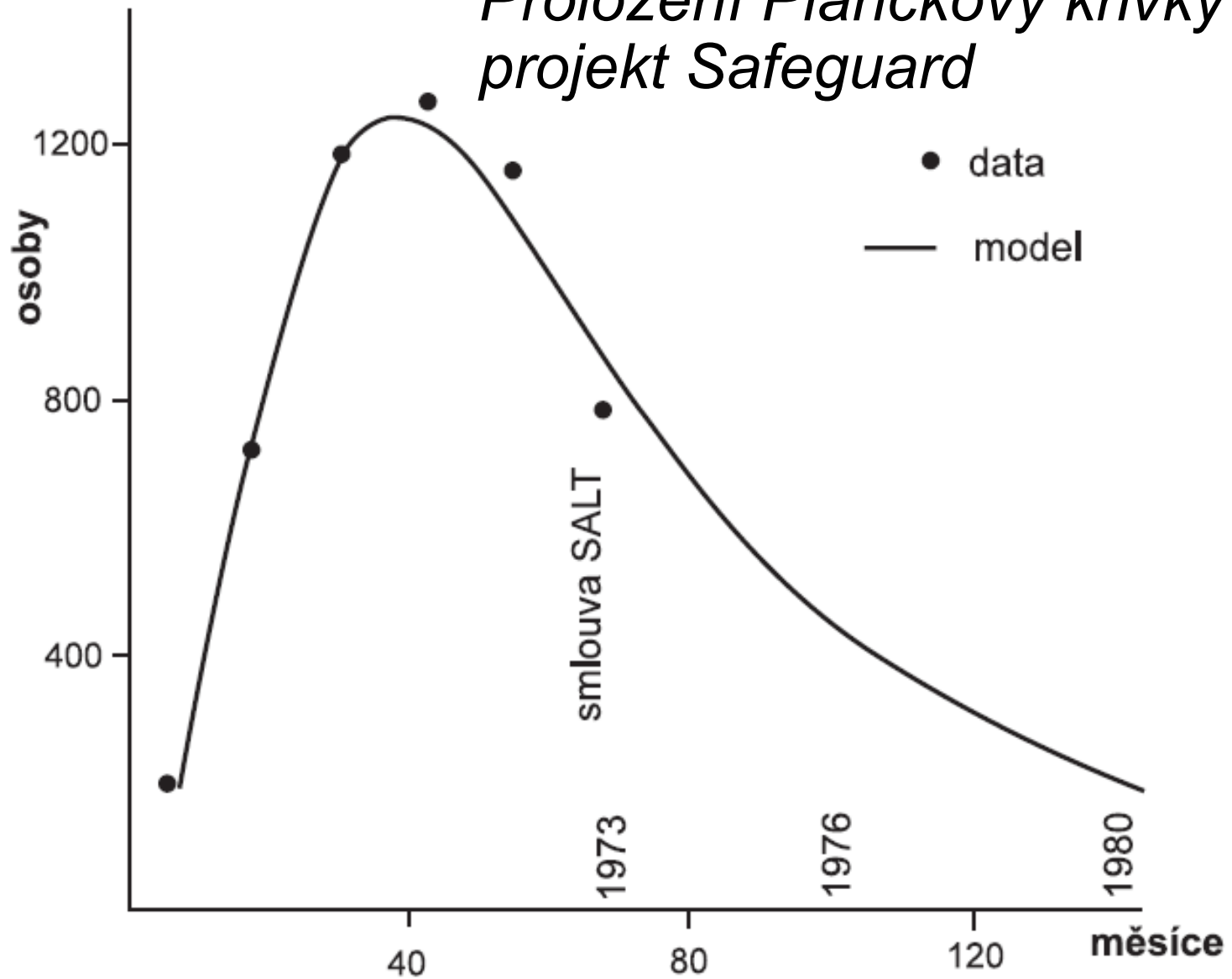


Transformace proměnných tak, aby max bylo v bodě 1 a mělo hodnotu 1 a v nule byla hodnota funkce prakticky nula. U pevného týmu odpovídá intenzitě práce

# Proložení Planckovy křivky pro SW řízení ponorek



# Proložení Planckovy křivky pro projekt Safeguard



Obr. 15.14: Planckův model velikosti týmů pro projekt Safeguard. 188

Bylo jasné, že projekt nelze v rozumné době dokončit i proto, že nebylo možno včas vytvořit SW. A také by to stálo majlant

# Zobecnění Planckova modelu

- Planckův model lze zobecnit

$$team(T) = \frac{C(T+kd)^{-q}d^q}{exp(D/(T+kd))-1}$$

Parametry jsou  $C$ ,  $D$ ,  $k$ ,  $d$   $q$ . Stefan-Boltzmannův zákon pak dostane tvar

$$Prac_B \geq c_{30} Doba_B^{-q+1}$$

# Zobecnění Planckova modelu

- Zobecněný Planckův model dostaneme výše uvedeným postupem, má-li Putnamův zákon tvar

- $Del \cong cPrac^{1/p}Doba^{q/p}$

pro vhodné kladné  $p$

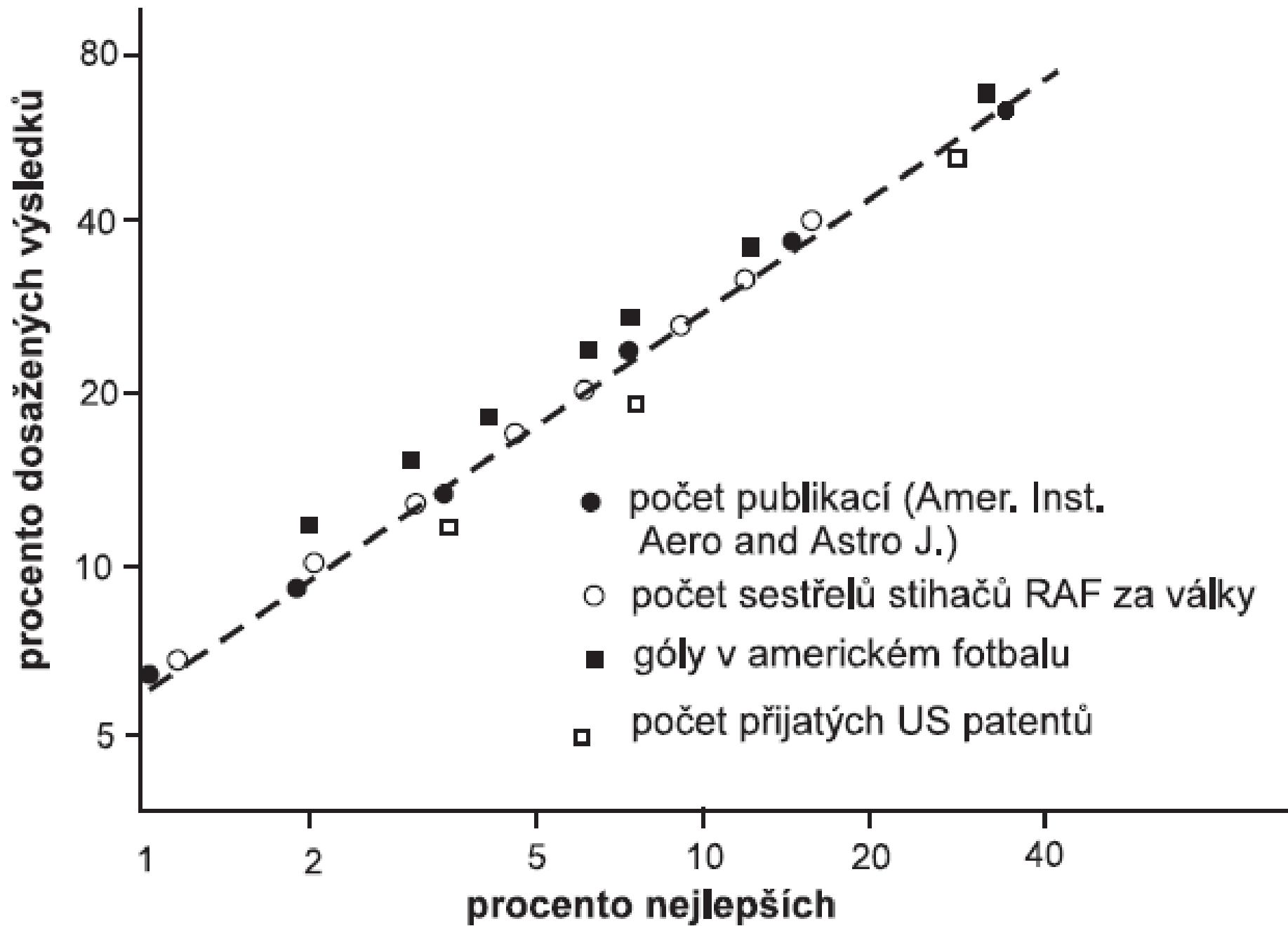
# Použití Rayleigtova a Planckova modelu

- Uvažujeme-li fakt, že část práce se po předání e přesune do údržby, platí následující pravidla
- **Rayleigt - zákon polovin.** V době dosažení maxima velikosti týmu jsme za polovinou doby řešení a spotřebovali jsme přes polovinu práce
- **Planck – zákon třetin,** jsme asi v třetině doby řešení a spotřebovali jsme třetinu práce (a tedy budeme muset dvojnásobek dosud spotřebované práce ještě vynaložit)



# Rozložení výkonnosti

Budiž  $a(p)$  procento výsledků, které dosáhlo  $p$  procent nejlepších (např.  $a(1)$  je procento branek, které nastřílelo jedno procento nejlepších. Uvidíme, že  $a(1)=7$ )  
Vyneseme-li graf  $a(p)$  v dvojité logaritmickém měřítku dostaneme následující obrázek.



# Výsledky nejlepších

- Z grafu vyplývá, že

$$a(p) = cp^{1/2}$$

kde  $c$  nepříliš silně závisí na typu činnosti

Procento nejlepších udělá 7,5% výsledků, 20% nejlepších udělá polovinu práce, 40% nejhorších neudělá skoro nic

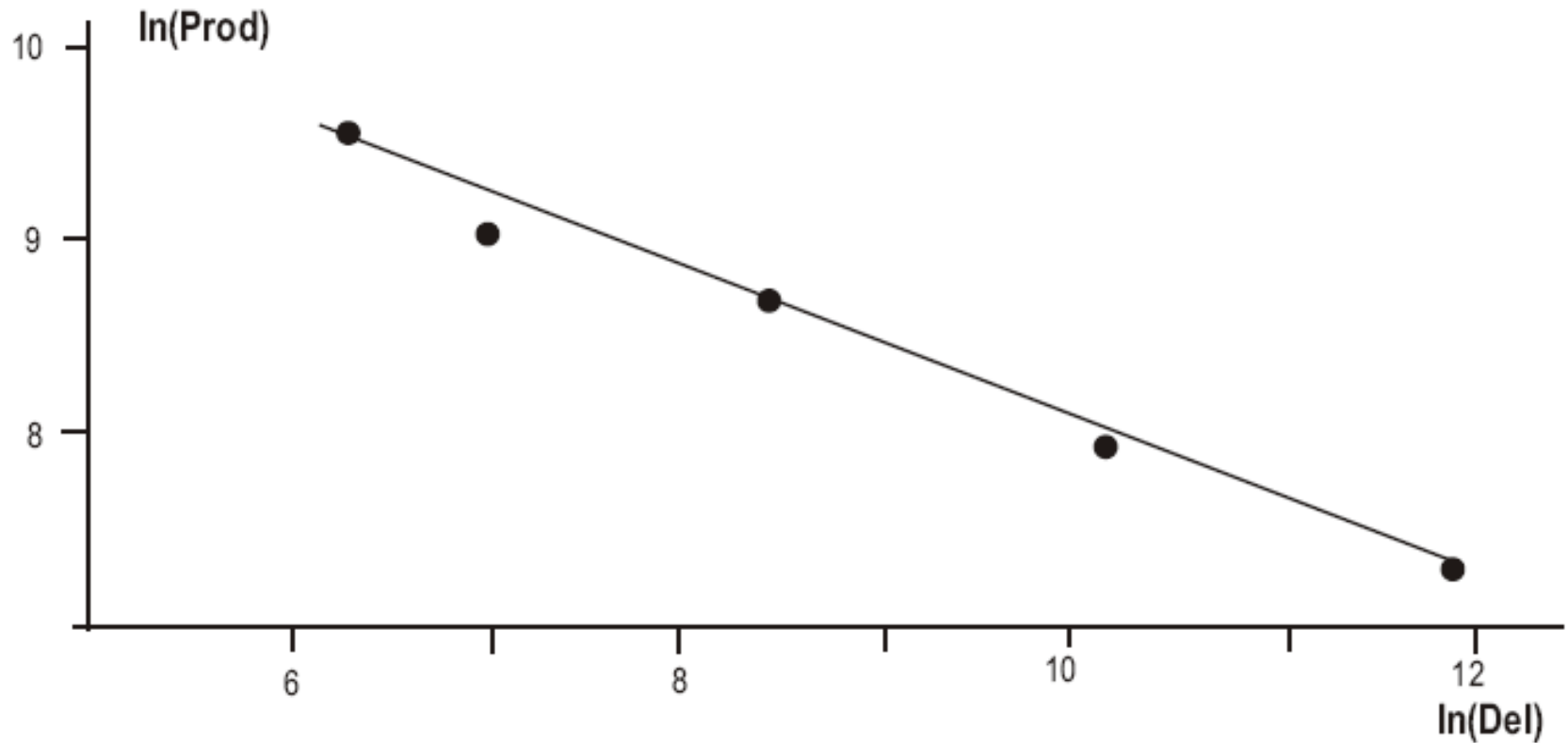
Seřadíme-li pracovníky podle výkonnosti, a budeme-li vynášet, kolik udělal každý jednotlivec, bude mít příslušná funkce tvar

$$b(p) = 1/2cp^{-1/2}$$

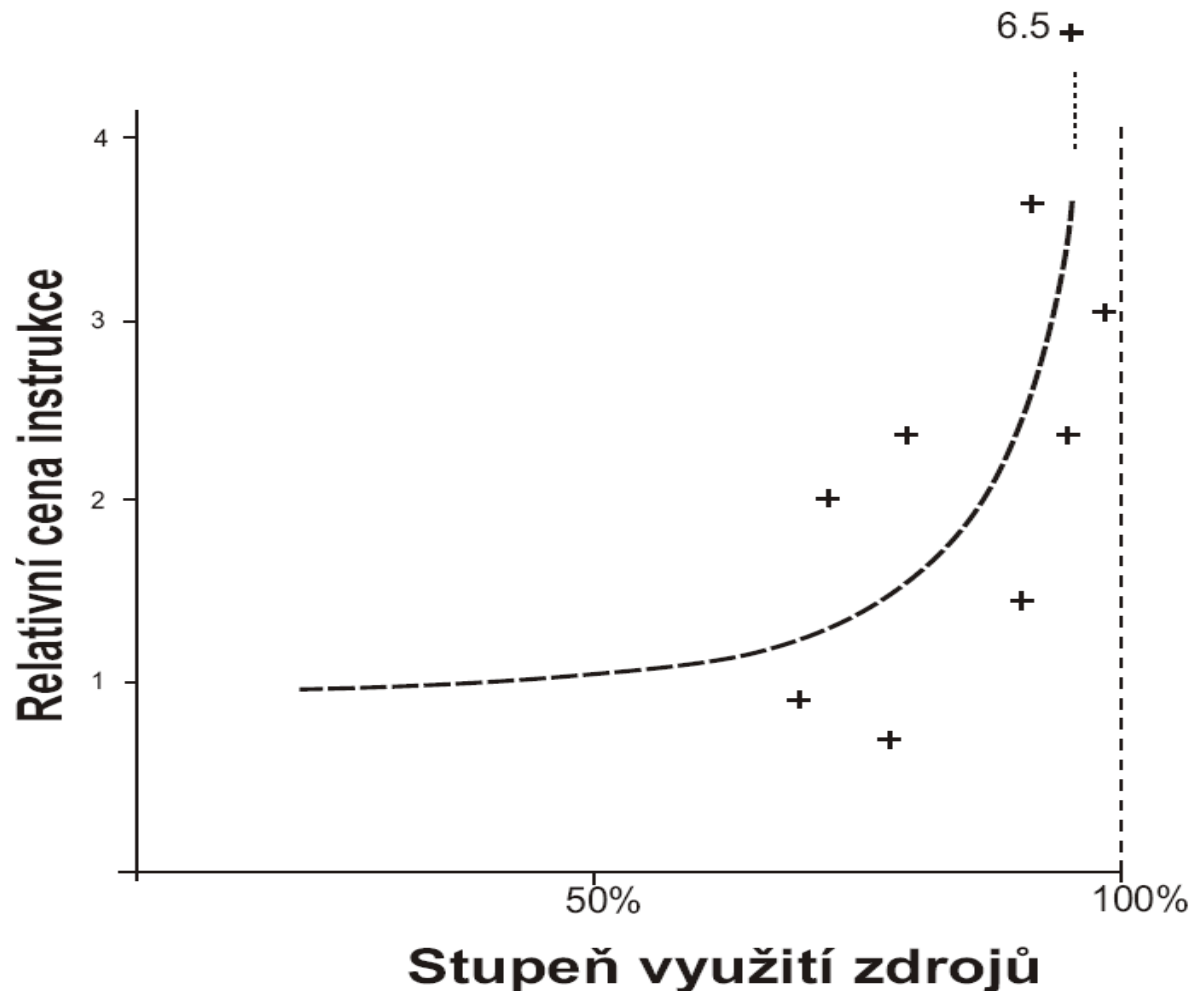
# Použití

- Vědomí důležitosti talentu a kvality lidí
  - Nejlepší udělají nejen nejvíce výsledků na hlavu, ale také udělají i nejlepší výsledky
- Pokud jsou ve větší skupině problémy na straně kvalitních lidí, je nutné zkoumat, čím to je a zda to vadí (např. zda se nejedná o rutinní práce, kde se mohou kvalitní lidé cítit nevytížení)

Závislost produktivity na délce programu pro malé programy, směrnice =  $-1/4$ .



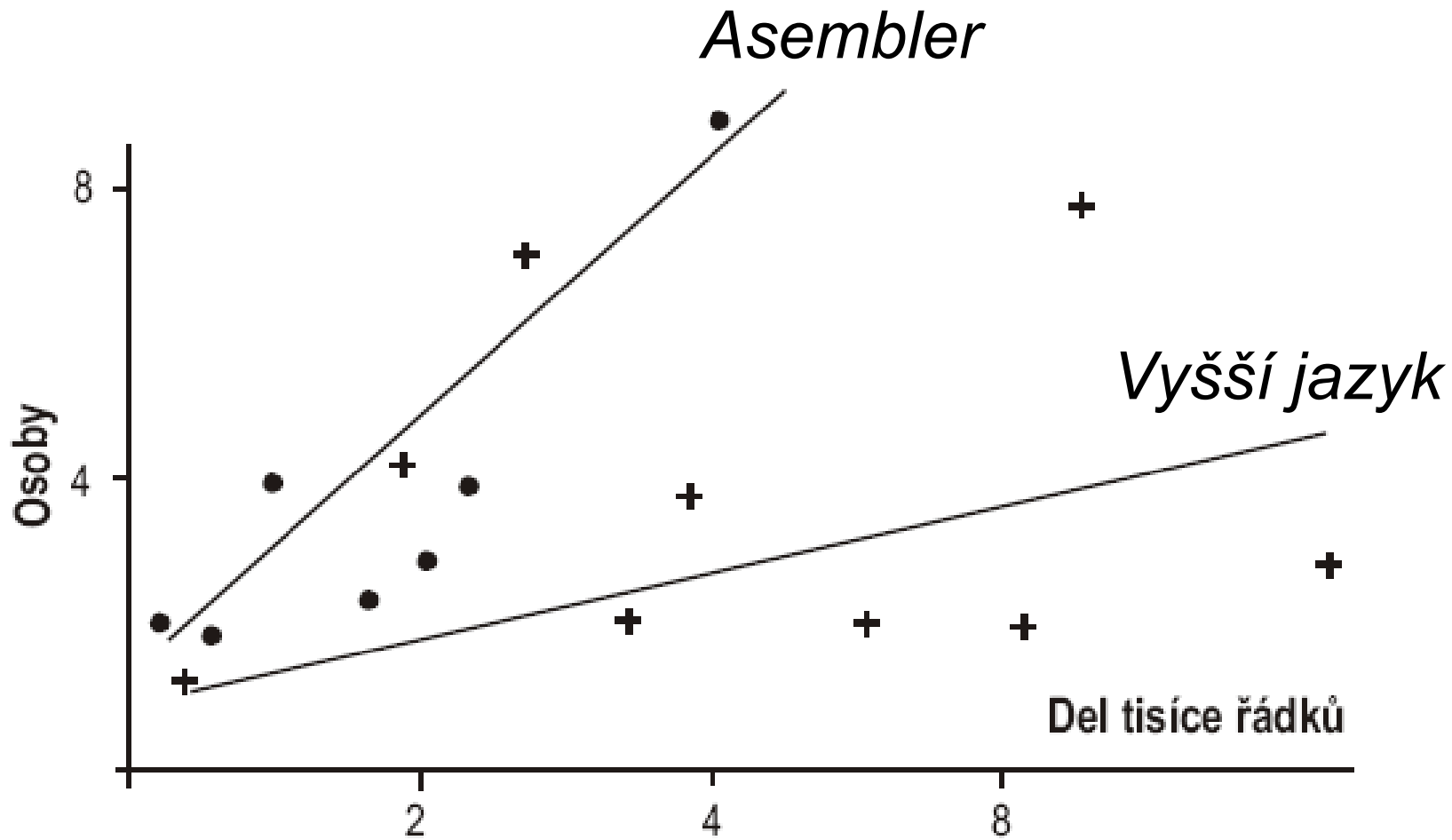
# Vliv vytíženost HW



# Využití

- U produktů, které nemají charakter masové spotřeby
  - Instalovat systém s 50-60% rezervou aby byl prostor na úpravy
  - Zvětšit rezervu na alespoň 50%, klesne-reserva pod 40%
  - Příklad použití: Letecký dispečink
  - Často se zbytečně šetří

# Náročnost údržby





# Využití

- Asembler je na údržbu (měřeno počtem osob udržujících systém na tisíc řádků) asi pětikrát pracnějšší než klasický programovací jazyk. U moderních systémů může být rozdíl ještě markantnějšší
- Poněvadž jeden řádek vyššího jazyka nahradí i několik řádek assembleru, je rozdíl produktivity alespoň 1:10
- Vyhýbat se assembleru, je-li to možné

# Databáze metrik

