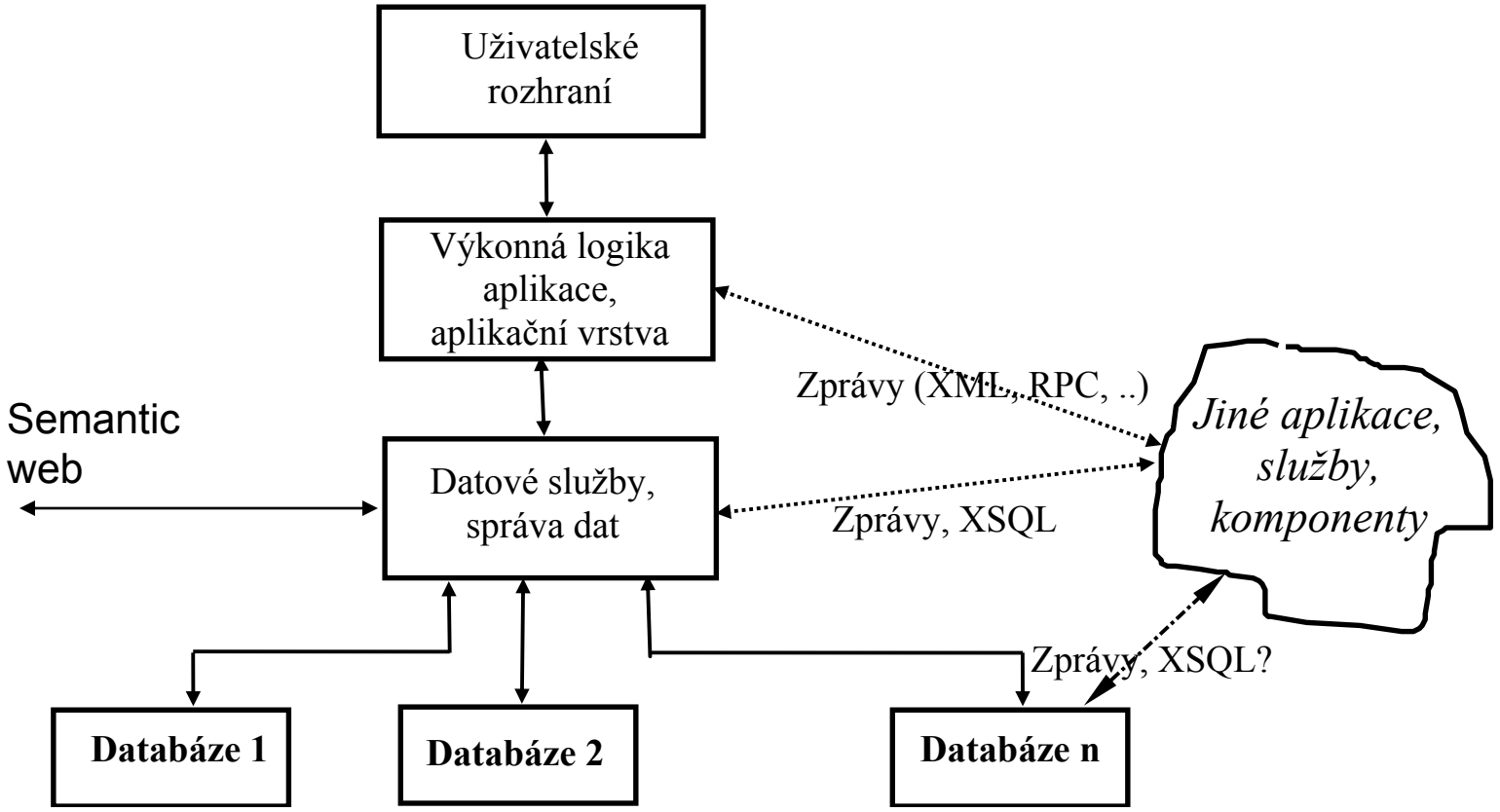
Metody návrhu systému

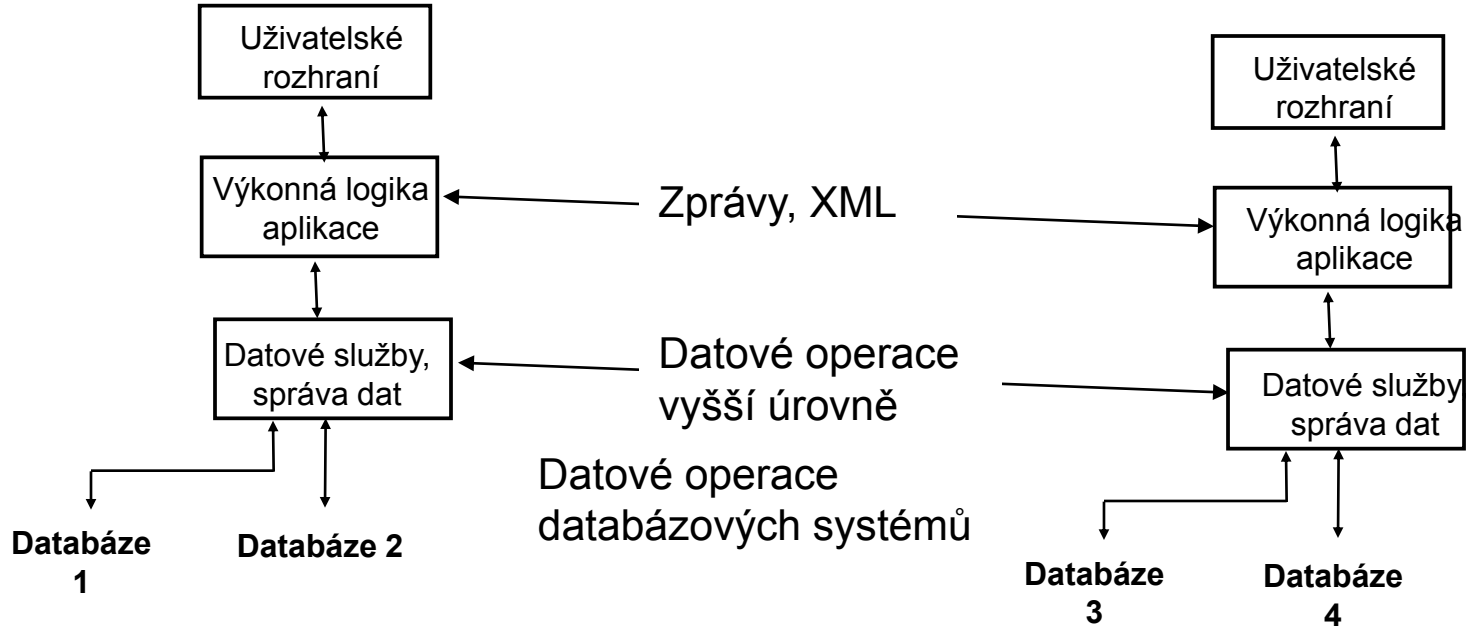
Jak pokračovat po specifikacích

Spolupráce třívrstvých komponent



Spolupráce třívrstvých komponent

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

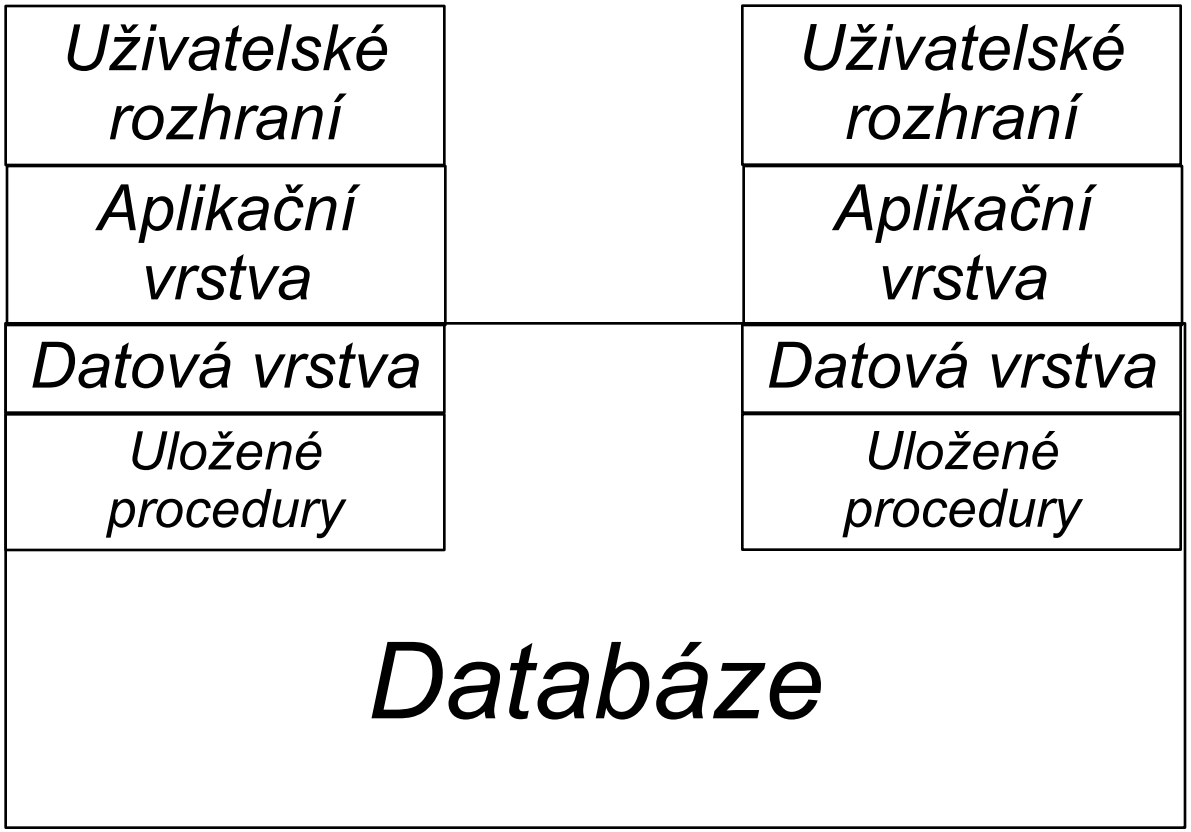


V SOA mohou být vrstvy tvořeny podsítěmi (pod SOA)

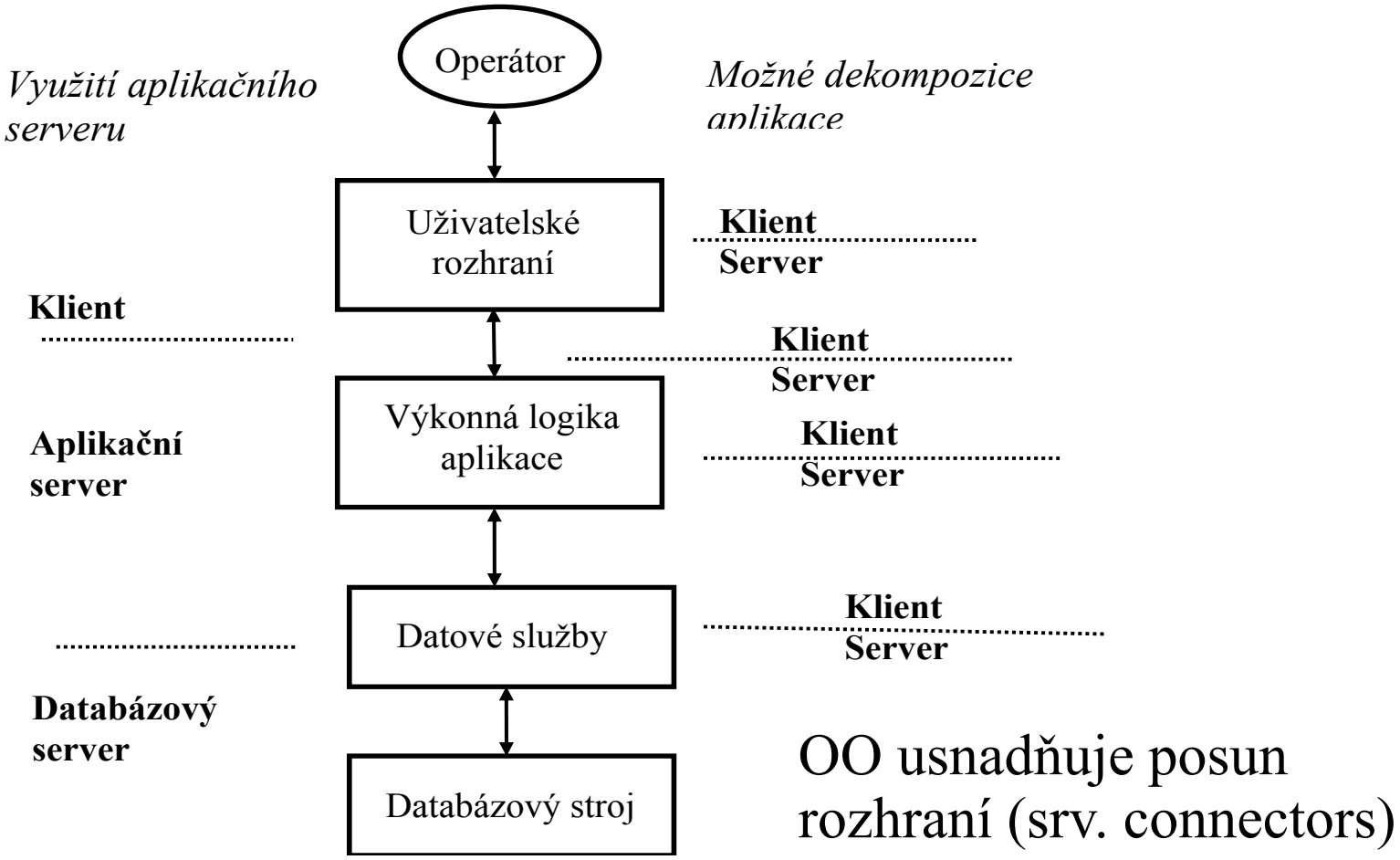
Databázově orientované systémy

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

Systémy propojené přes databázi

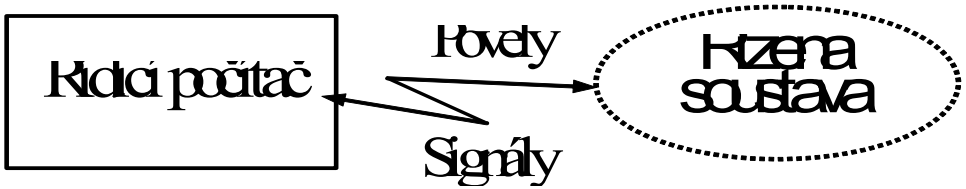


Tři vrstvy a server

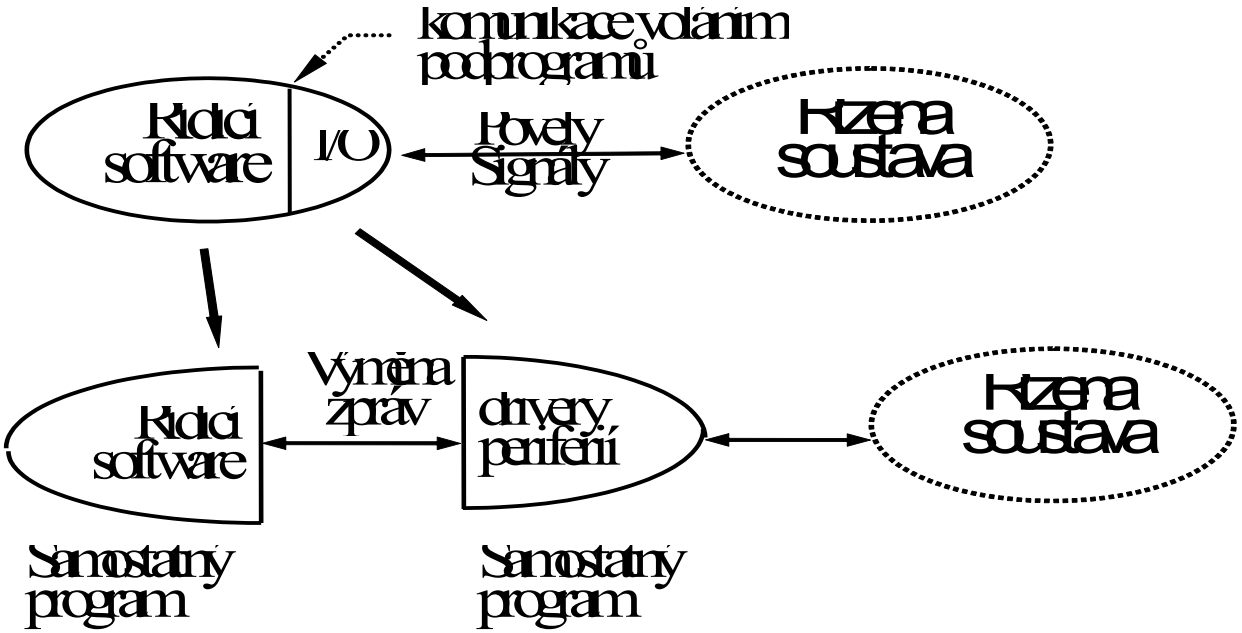


OO usnadňuje posun rozhraní (srv. connectors)

K SOA

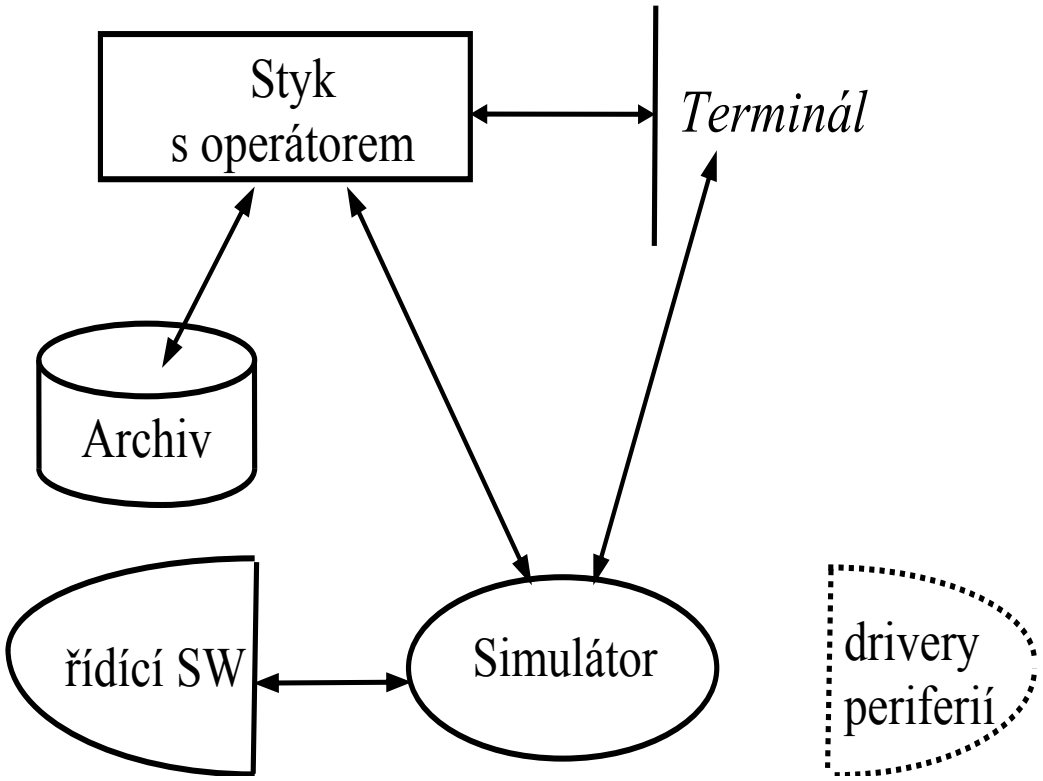


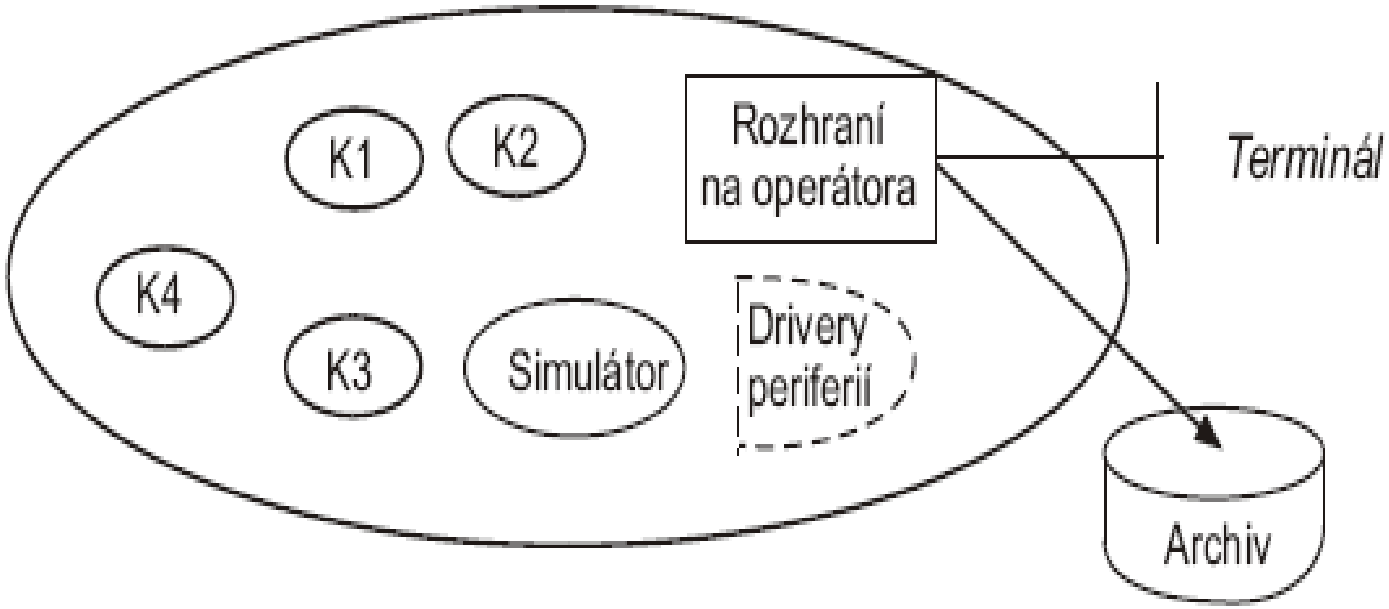
a) Soustava řízená počítačem



b) Oddělení ovladačů periferií realizujících styk s řízenou soustavou

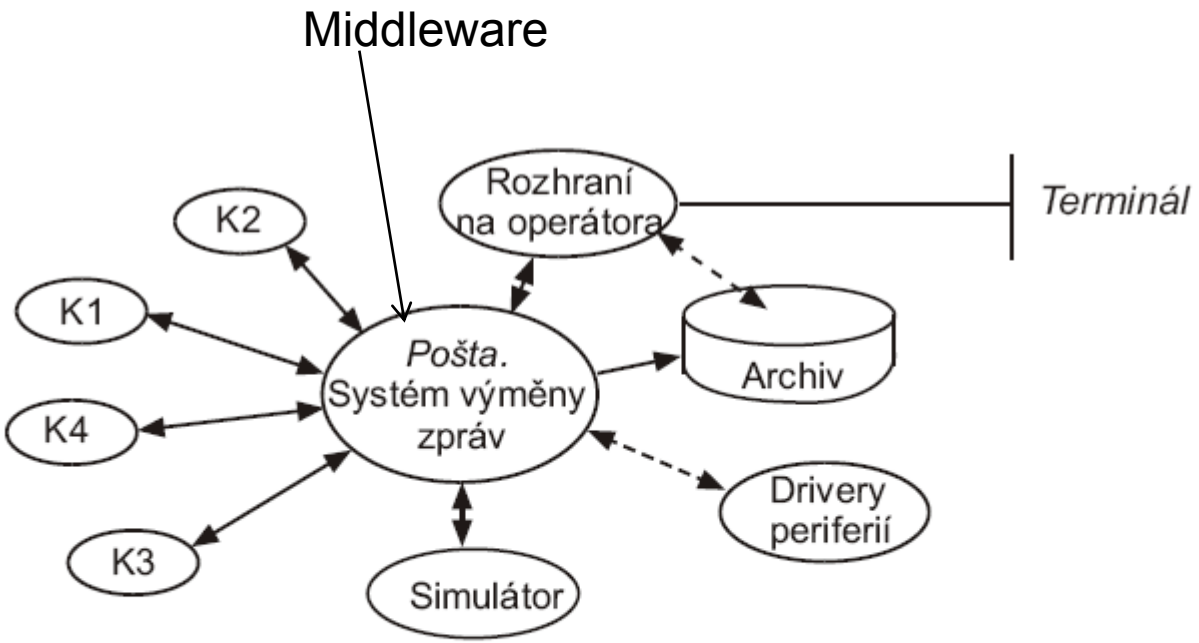
K SOA, základní sestava





Systém po dekompozici..

Každá komponenta může komunikovat s libovolnou komponentou



Dekomponovaný systém se subsystémem výměny zpráv.
Jednotlivé zprávy jsou posílámy prostřednictvím specializovaného subsystému zajiš ujícího 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 udržovatelnost
- 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 spoluopracovat se SOAP a obecně jak optimálně aplikovat to nejlepší z objektové orientace
- Obtížné přijetí filosofie SOA

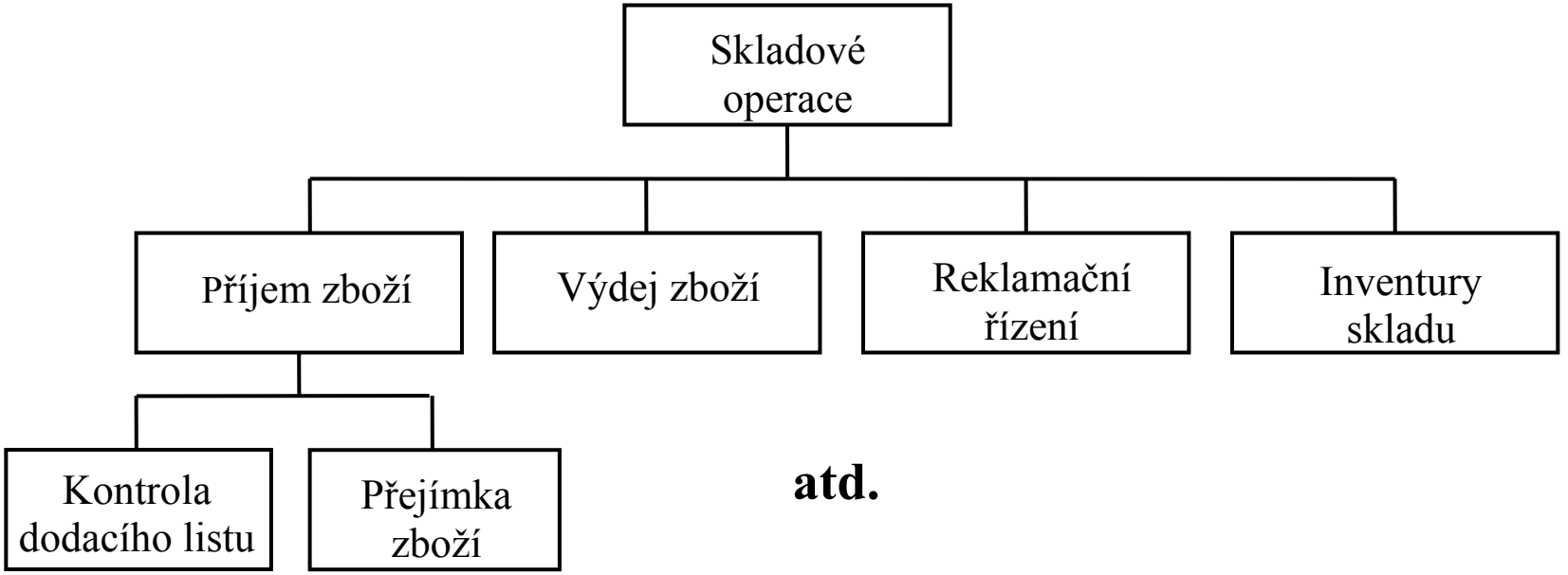
Jaksonova metoda

- 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

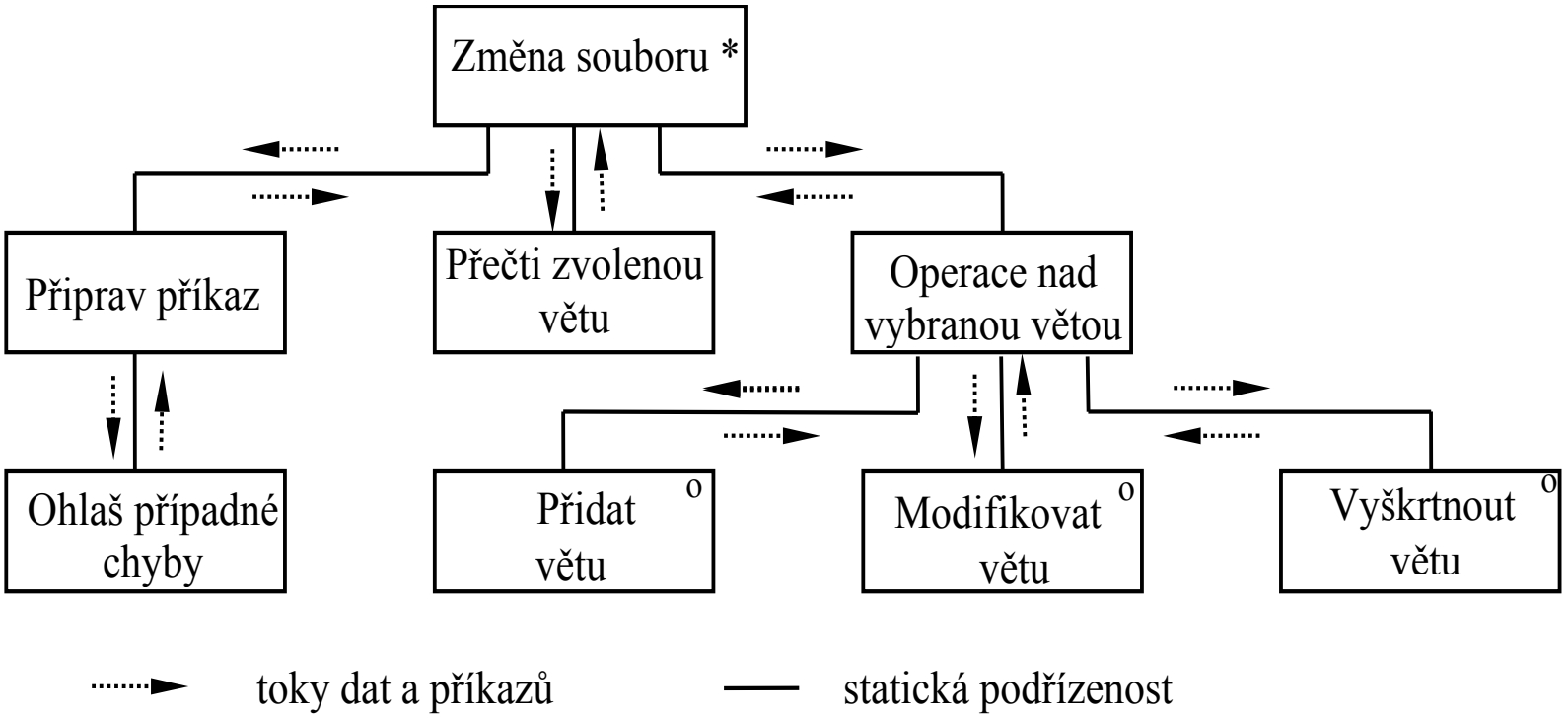
- Je vhodná pro vývoj jednotlivých procesů v diagramu toků dat
- V jistém smyslu obdoba objektové orientace pro dávkové systémy

Jaksonova metoda



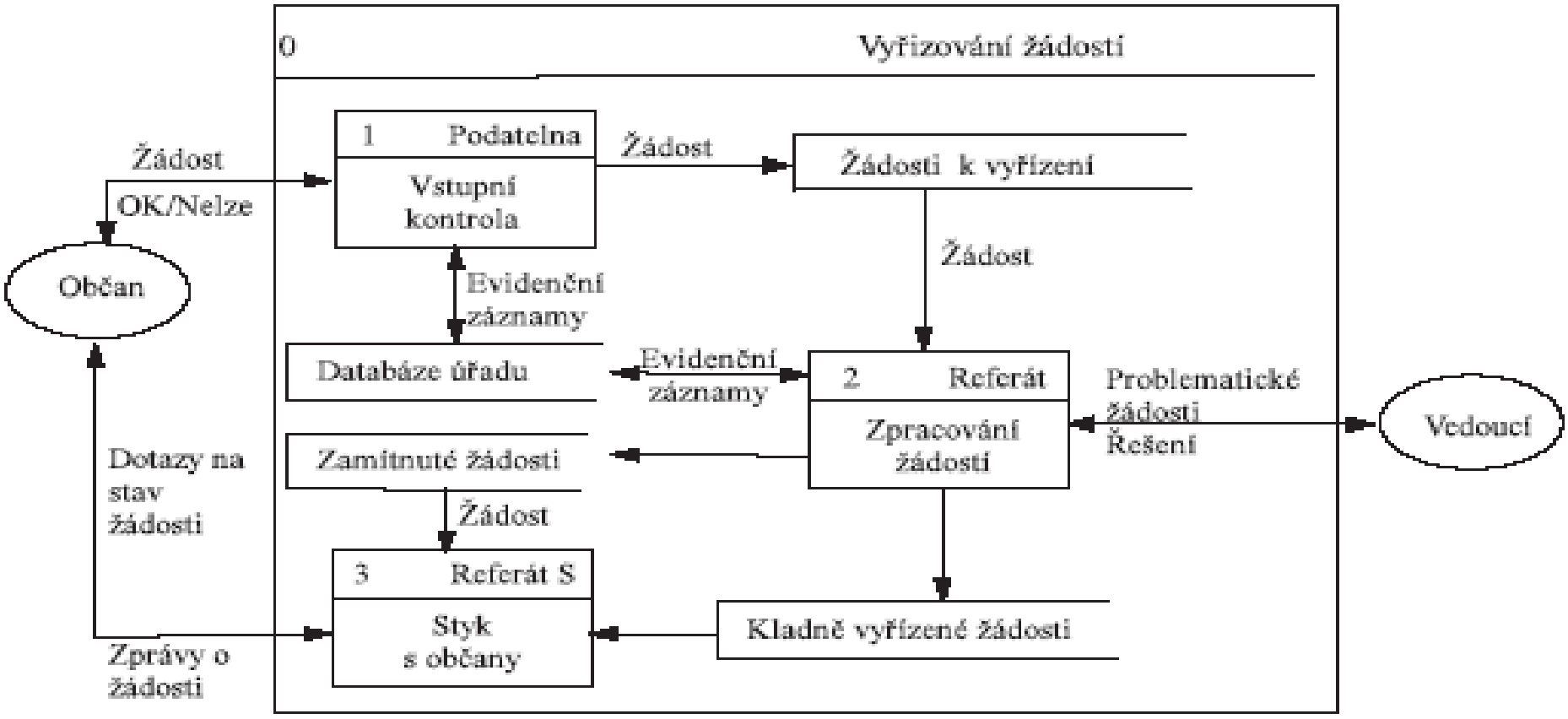
atd.

Jaksonova metoda



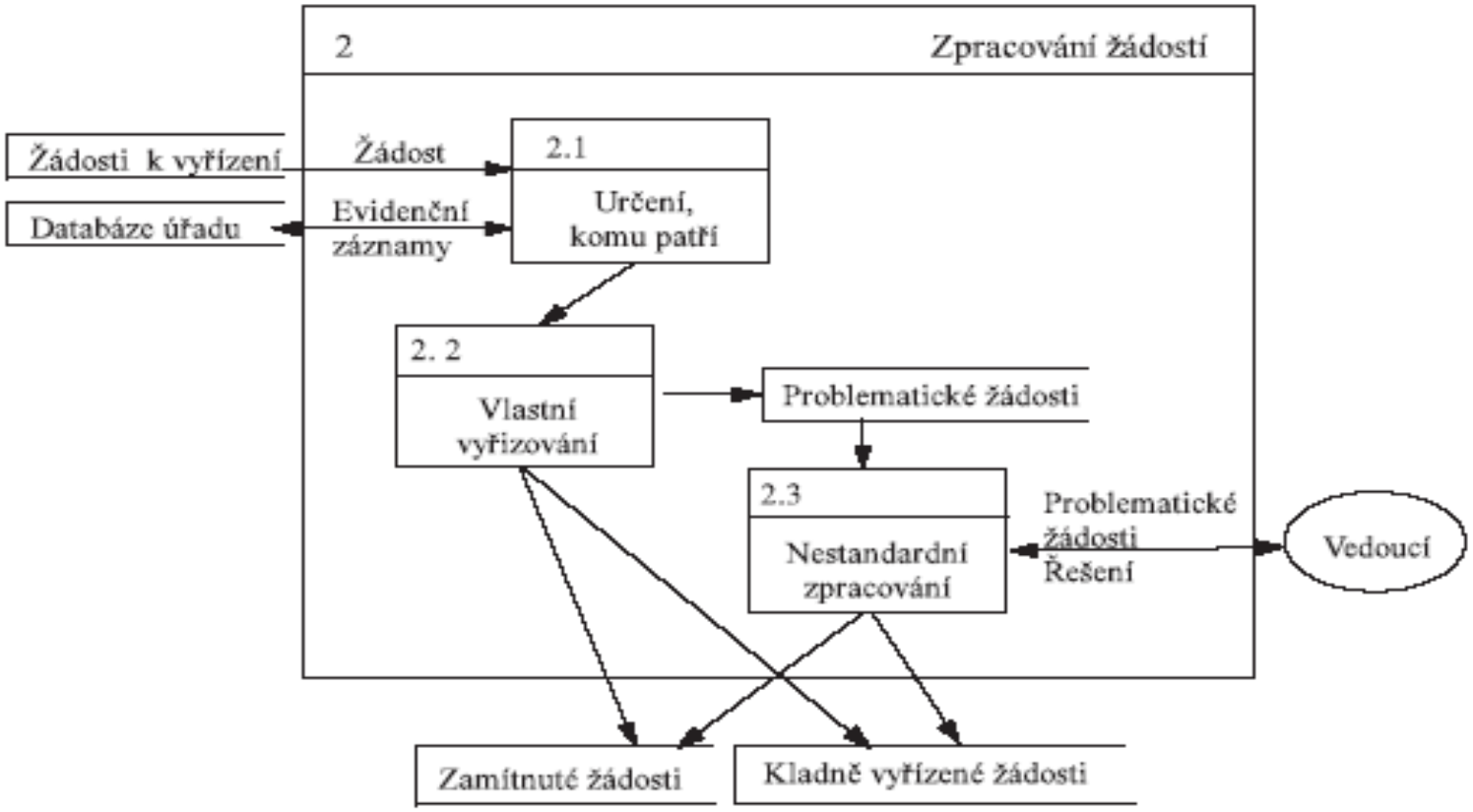
Často lze strukturu programu odvodit z dekompozice činností, vhodné pro dávkové zpracování

Dataflow



Obr. 12.9: Diagram toků dat systému Vyřizování žádostí.

Dataflow, funkcionální dekompozice



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

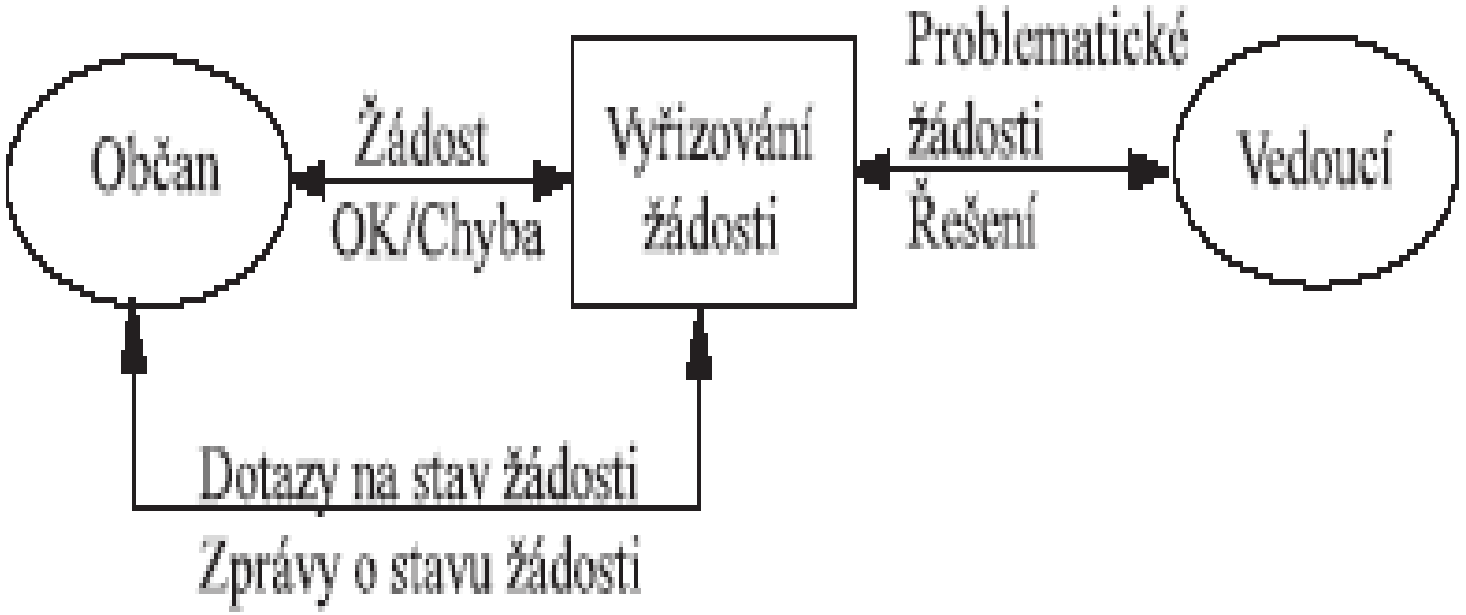
Prvky DFD v SOA a cloudu

- 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 žádostí.

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				

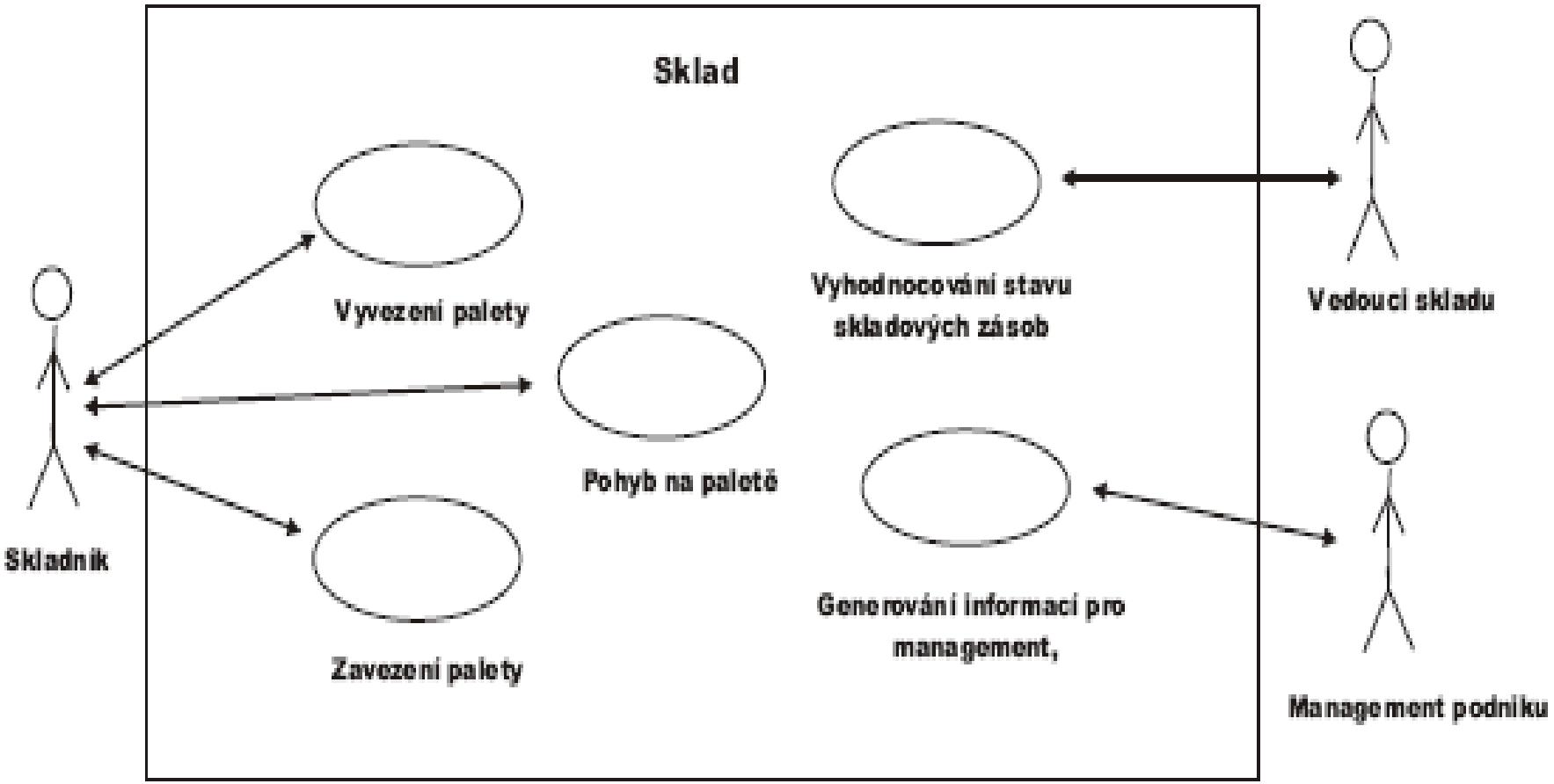
Rozhodovací tabulky

- 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
- Případ použití

Případy použití



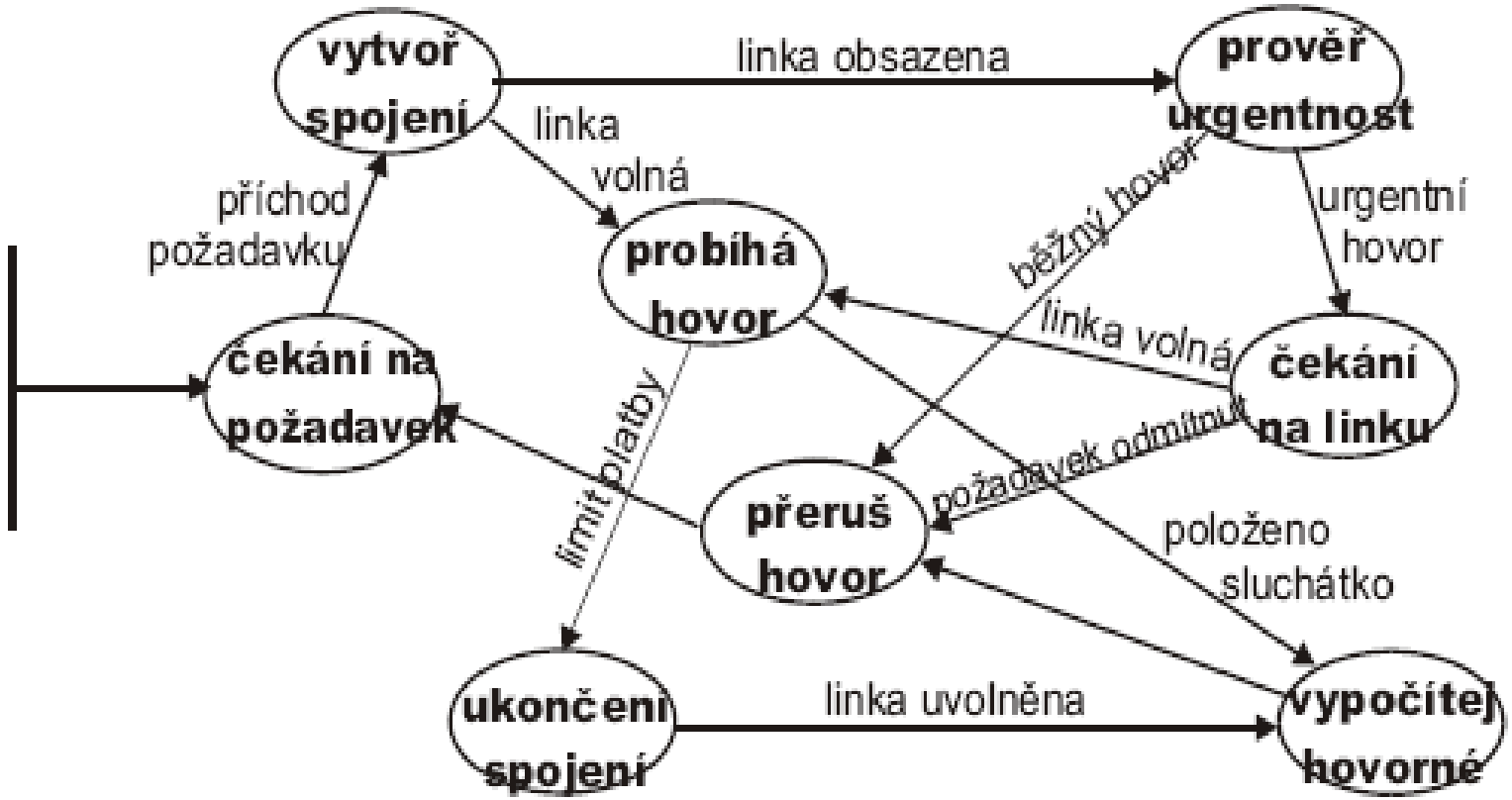
Případy použití

- 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)

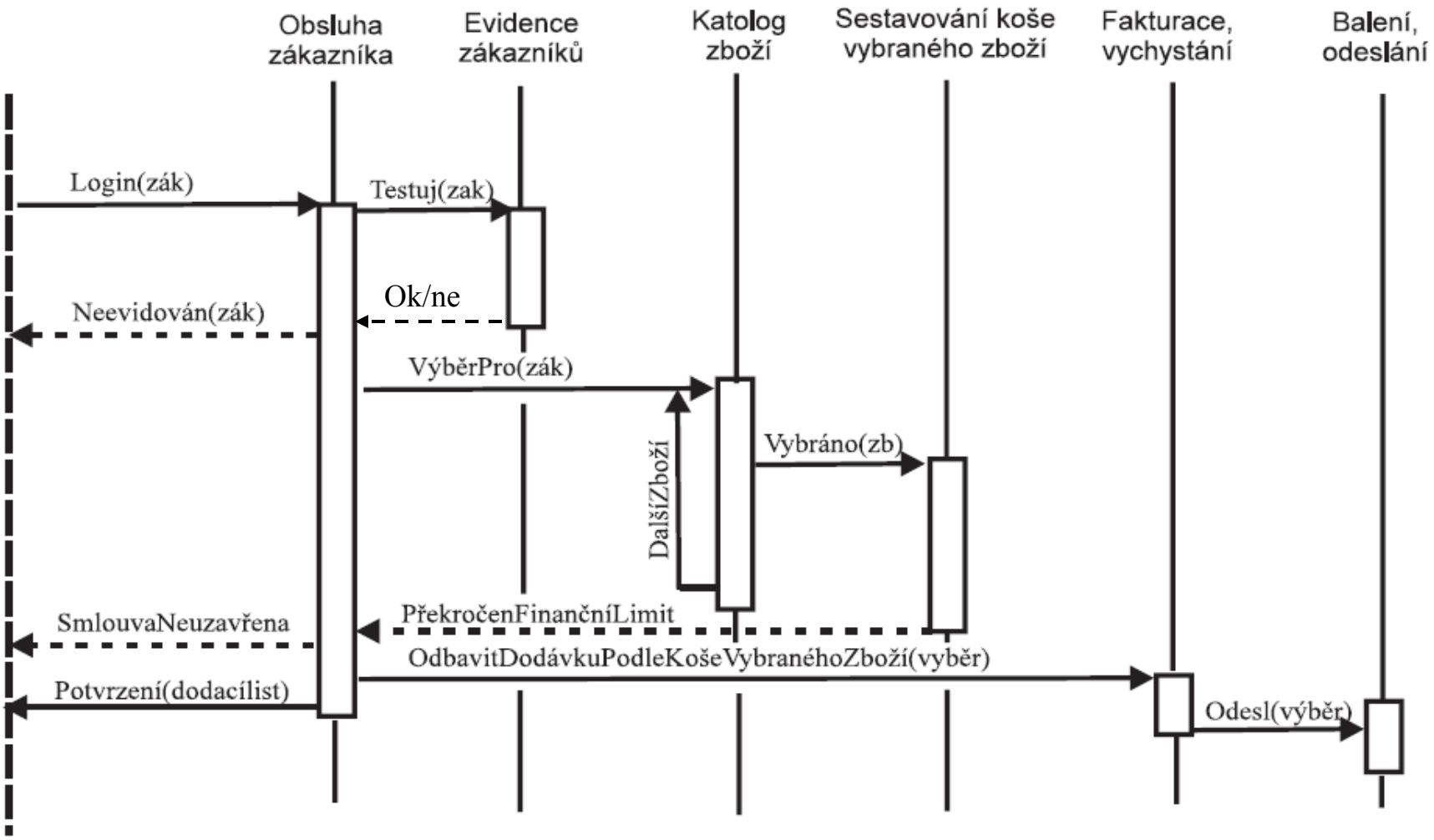
Pracovní tok

Pracovní tok, scénář

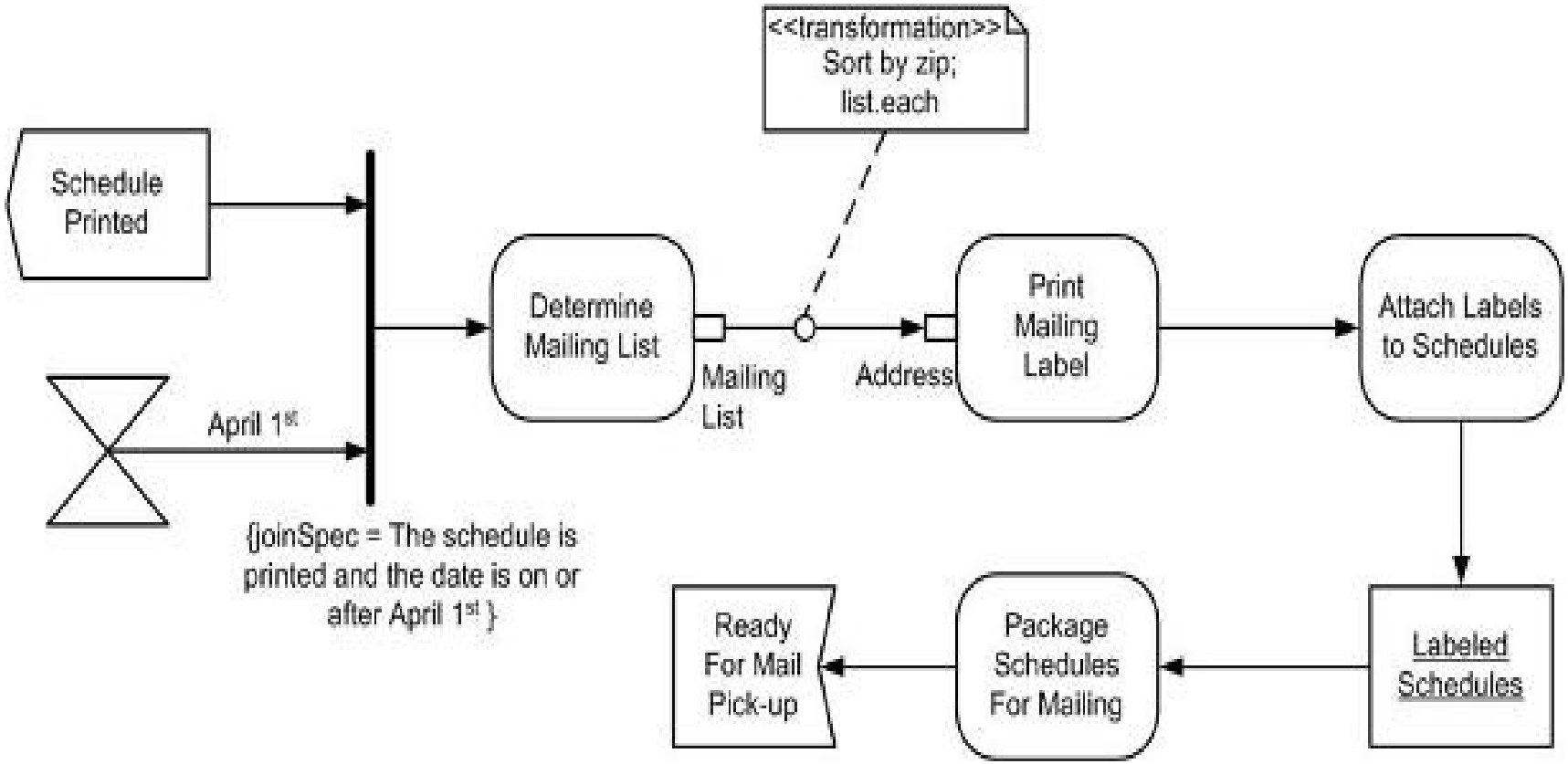
Pracovní tok, spolupráce komponent

- 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 mýsto komponent používá jednotlivé třídy

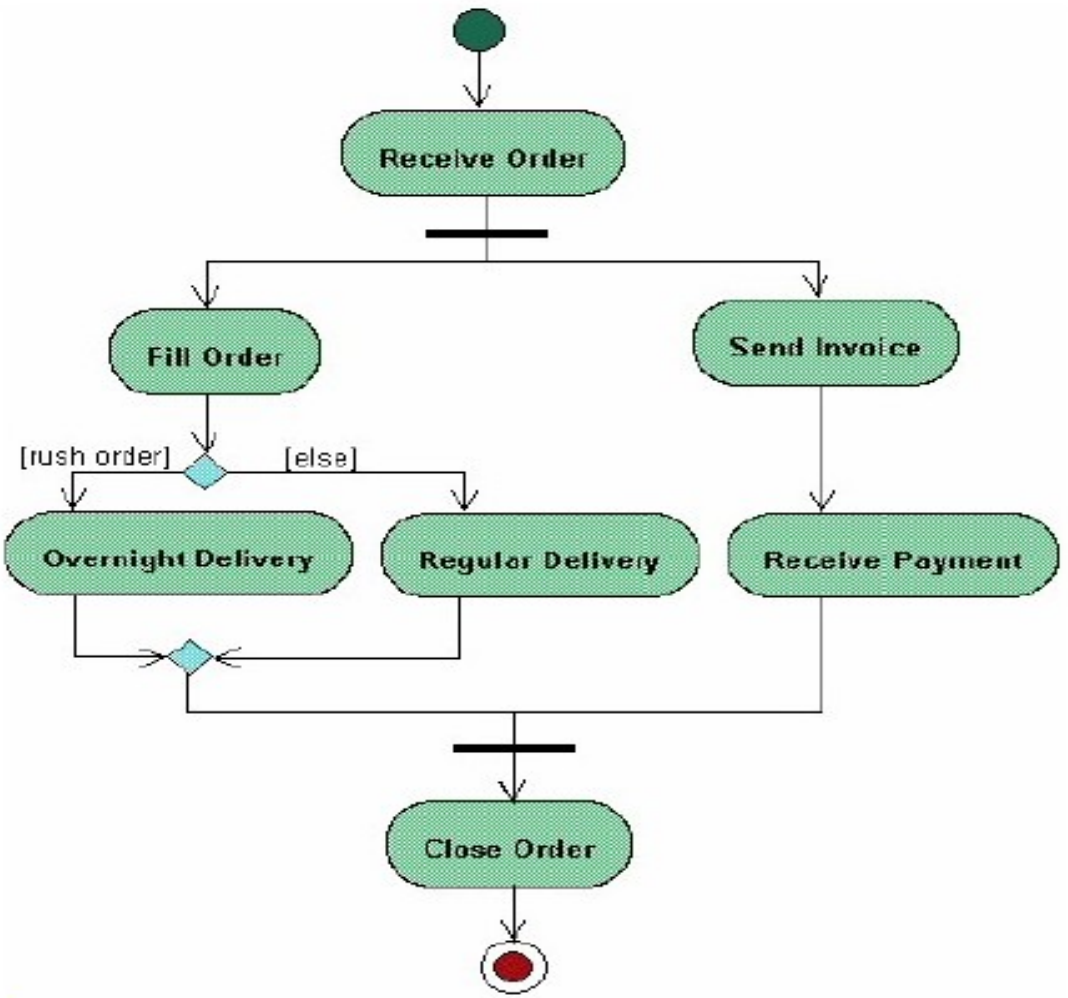
Pracovní tok, UML



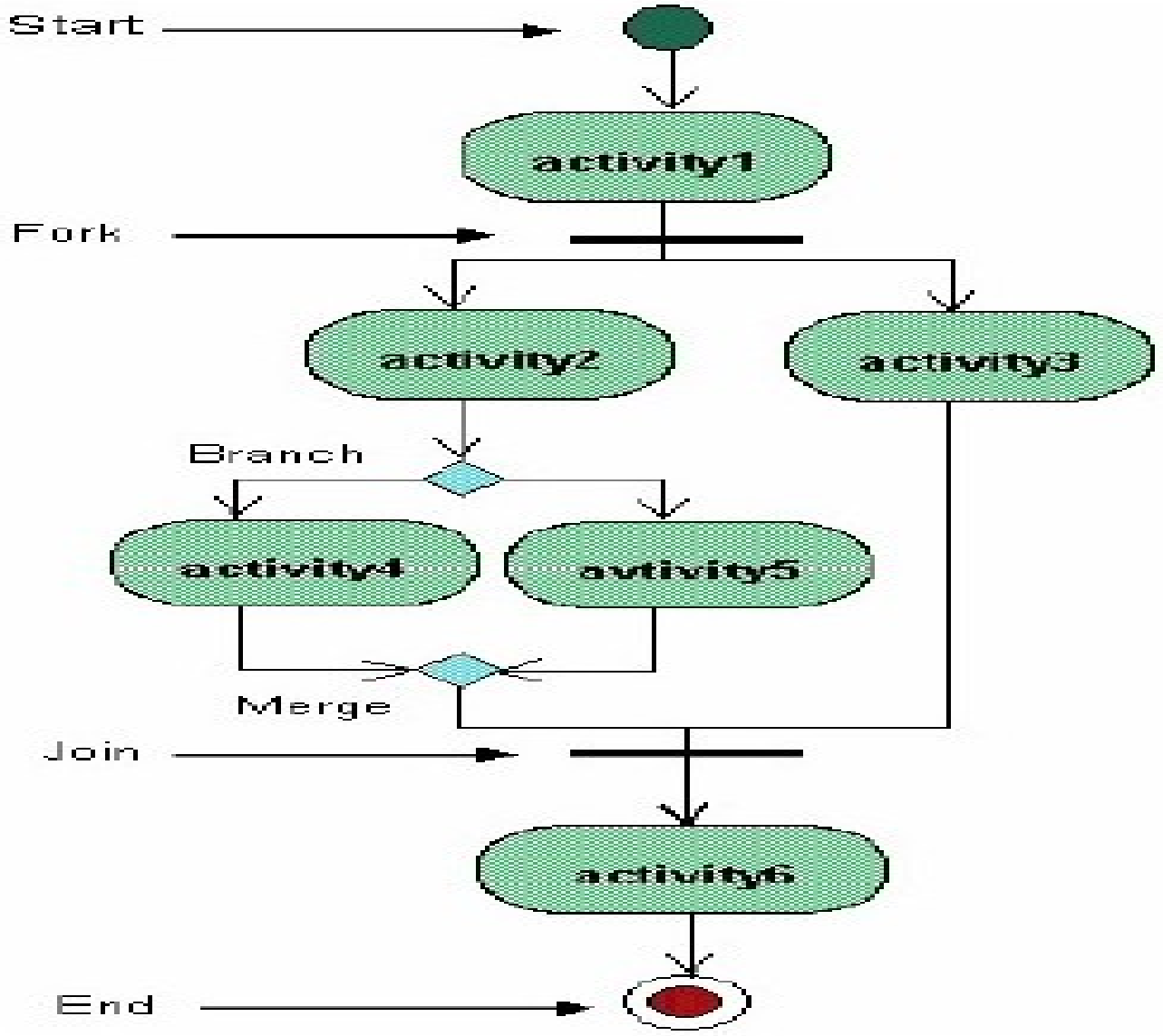
Příklad diagramu aktivit

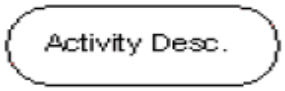
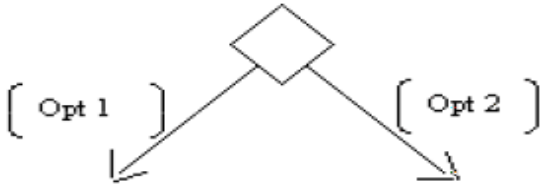
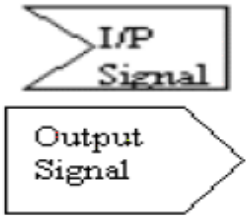
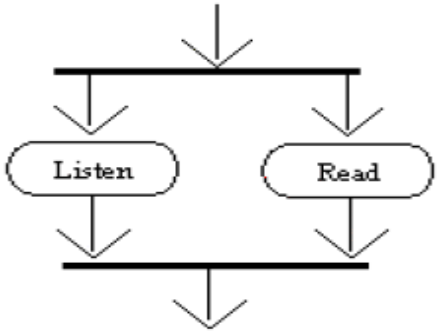


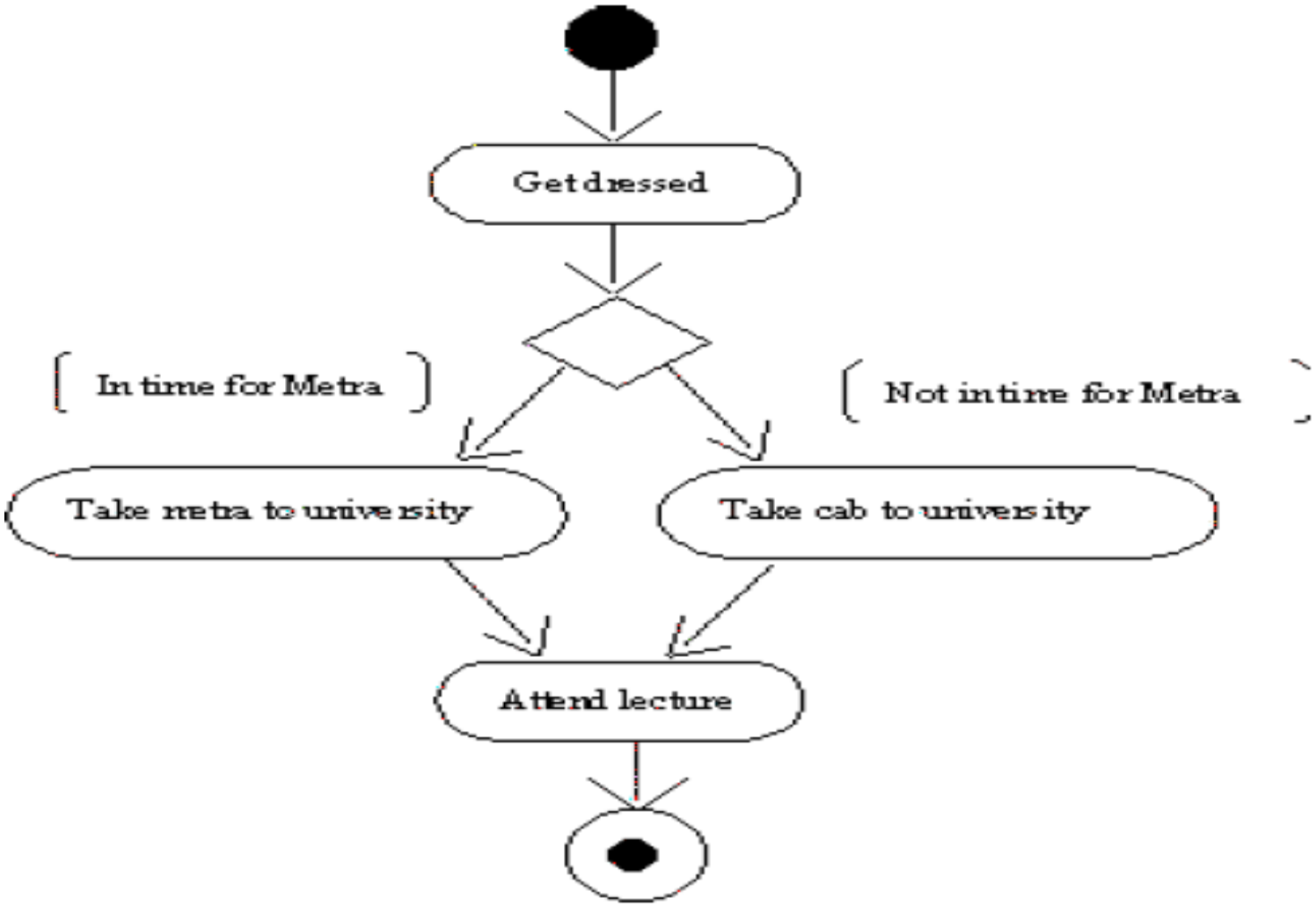
Diagramy aktivit

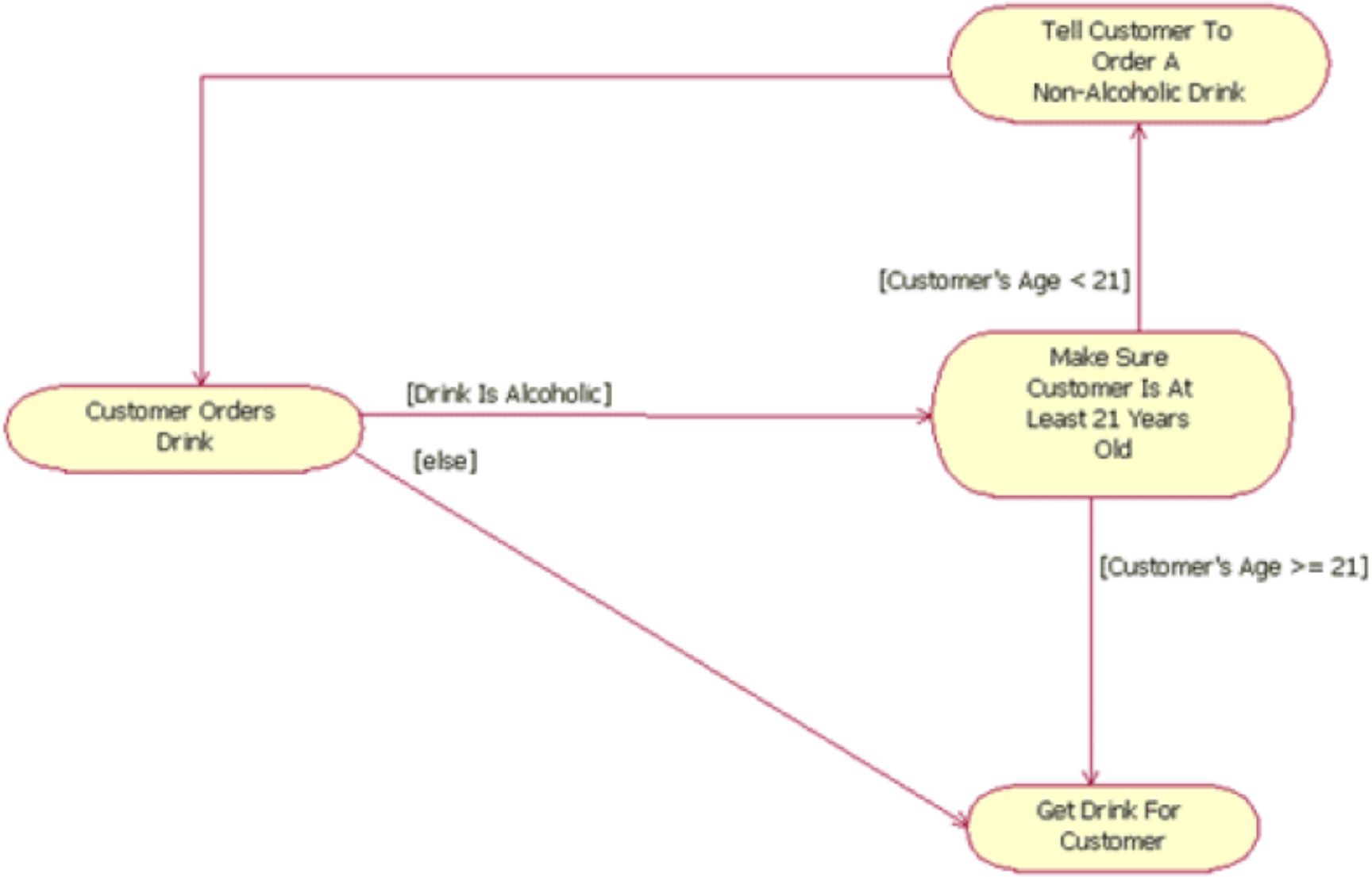


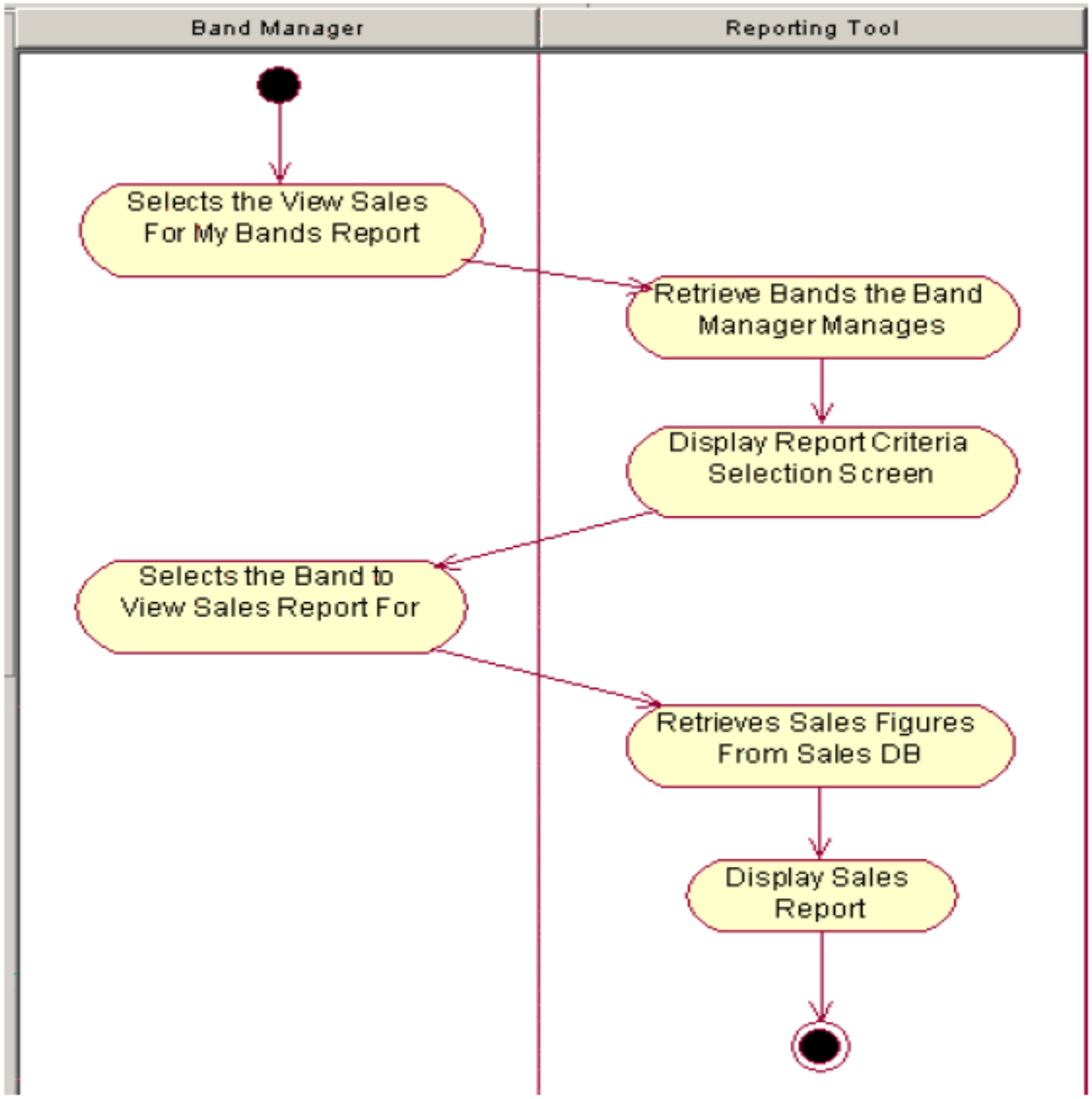
Diagramy aktivit



Element and its description	Symbol
Initial Activity: 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.	
Activity: Represented by a rectangle with rounded (almost oval) edges.	
Decisions: 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.	
Signal: 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.	
Concurrent Activities: 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.	
Final Activity: The end of the Activity diagram is shown by a bull's eye symbol, also called as a final activity.	


Get dressed
[In time for Metra]
[Not in time for Metra]
Take metra to university
Take cab to university
Attend lecture


Tell Customer To Order A Non-Alcoholic Drink
[Customer's Age < 21]
Make Sure Customer Is At Least 21 Years Old
Customer Orders Drink
[Drink Is Alcoholic]
[else]
[Customer's Age >= 21]
Get Drink For Customer


Band Manager
Reporting Tool
Selects the View Sales For My Bands Report
Retrieve Bands the Band Manager Manages
Display Report Criteria Selection Screen
Selects the Band to View Sales Report For
Retrieves Sales Figures From Sales DB
Display Sales Report

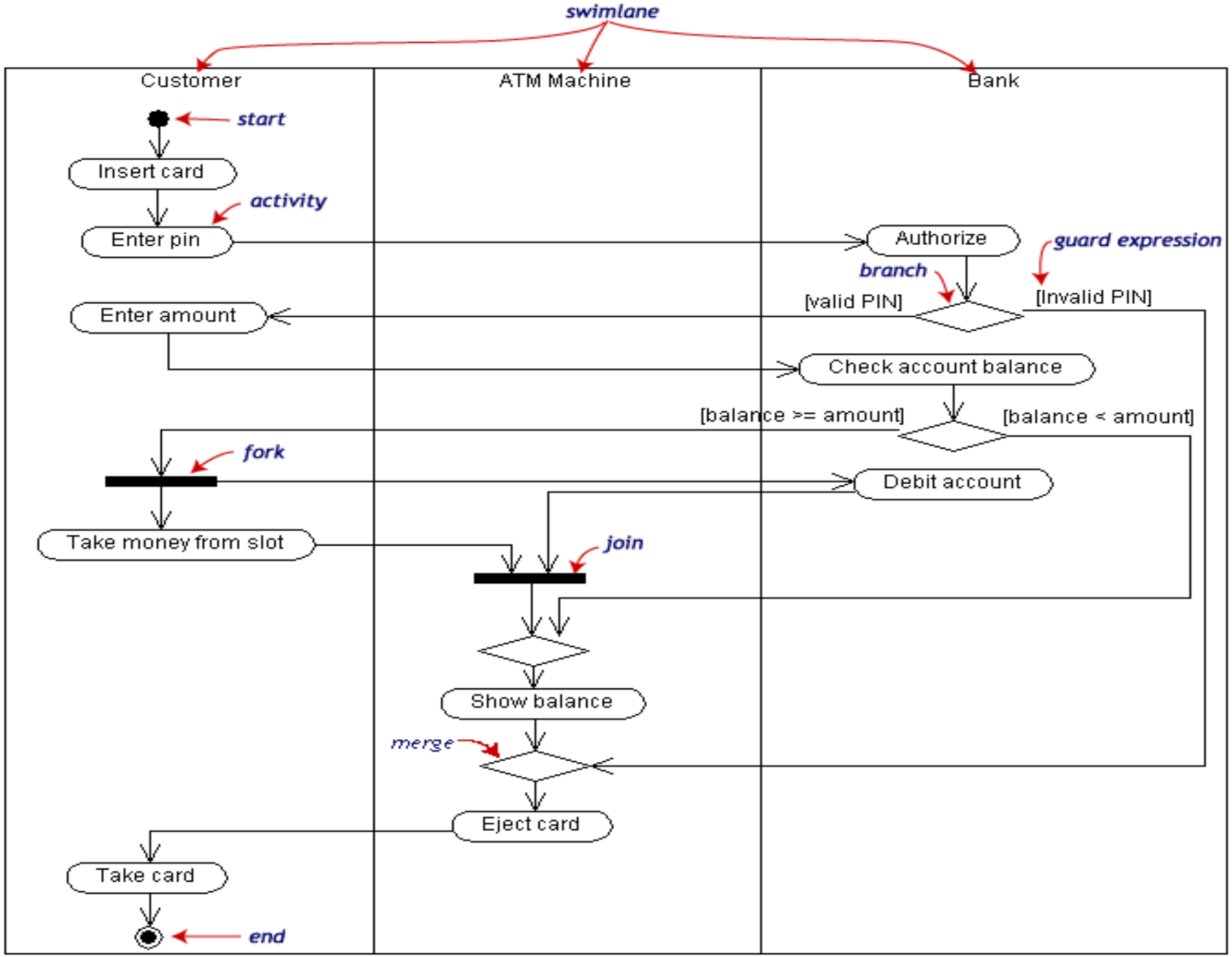
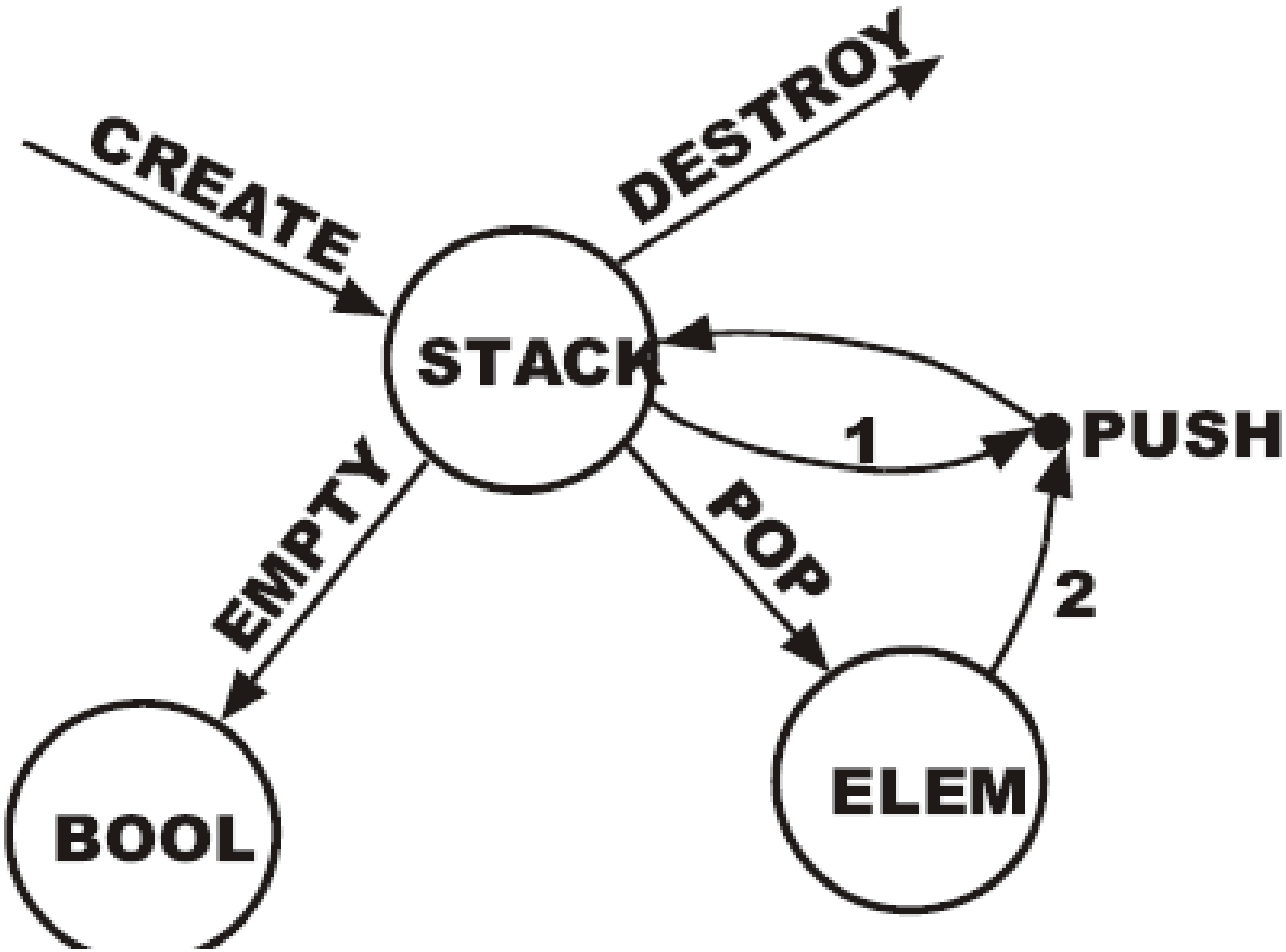

swimlane
Customer
ATM Machine
Bank
start
Insert card
activity
Enter pin
Authorize
guard expression
branch
[valid PIN]
[Invalid PIN]
Enter amount
Check account balance
[balance >= amount]
[balance < amount]
fork
Debit account
Take money from slot
join
Show balance
merge
Eject card
Take card
end

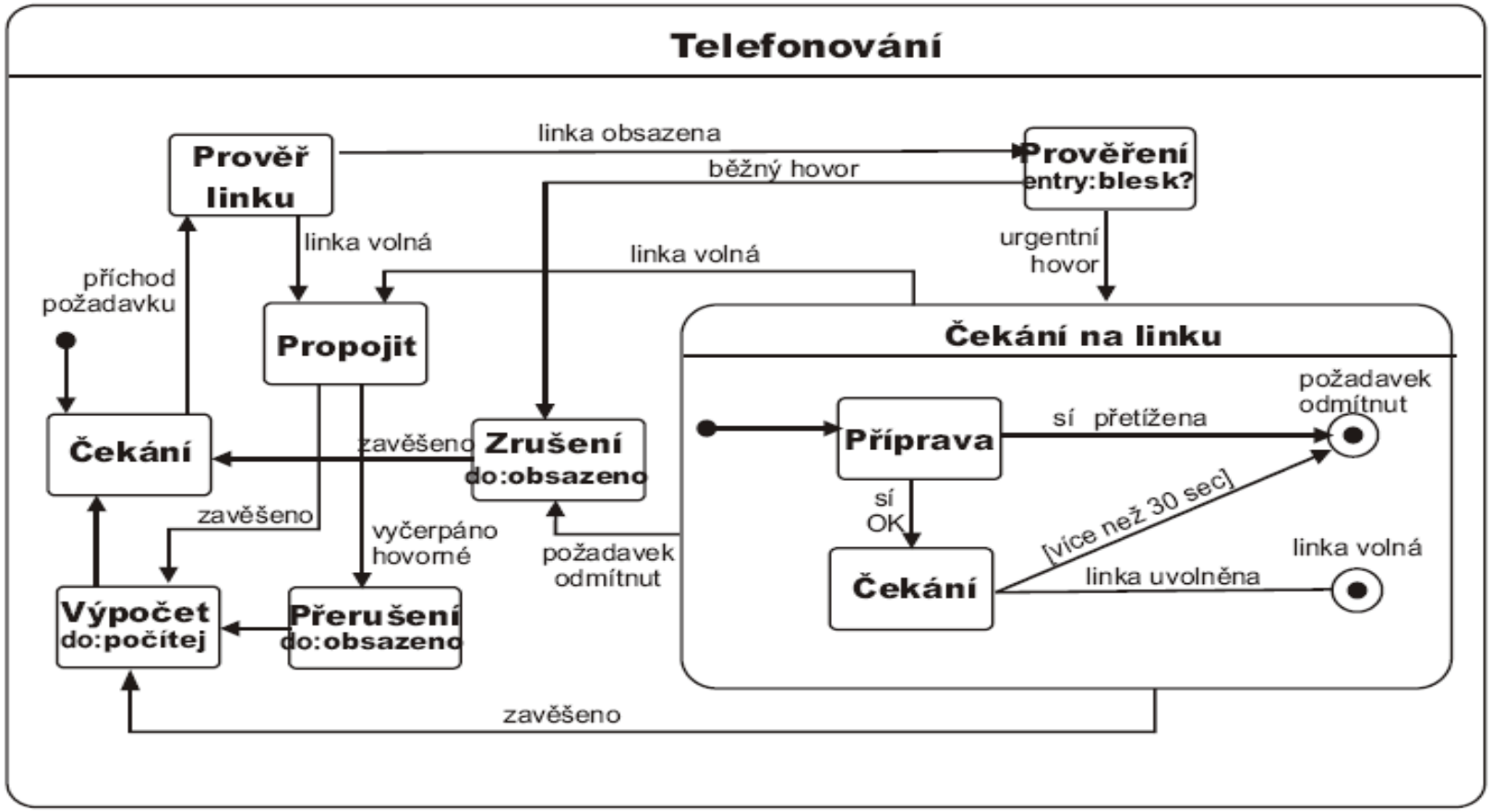
Diagram aktivit

- 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

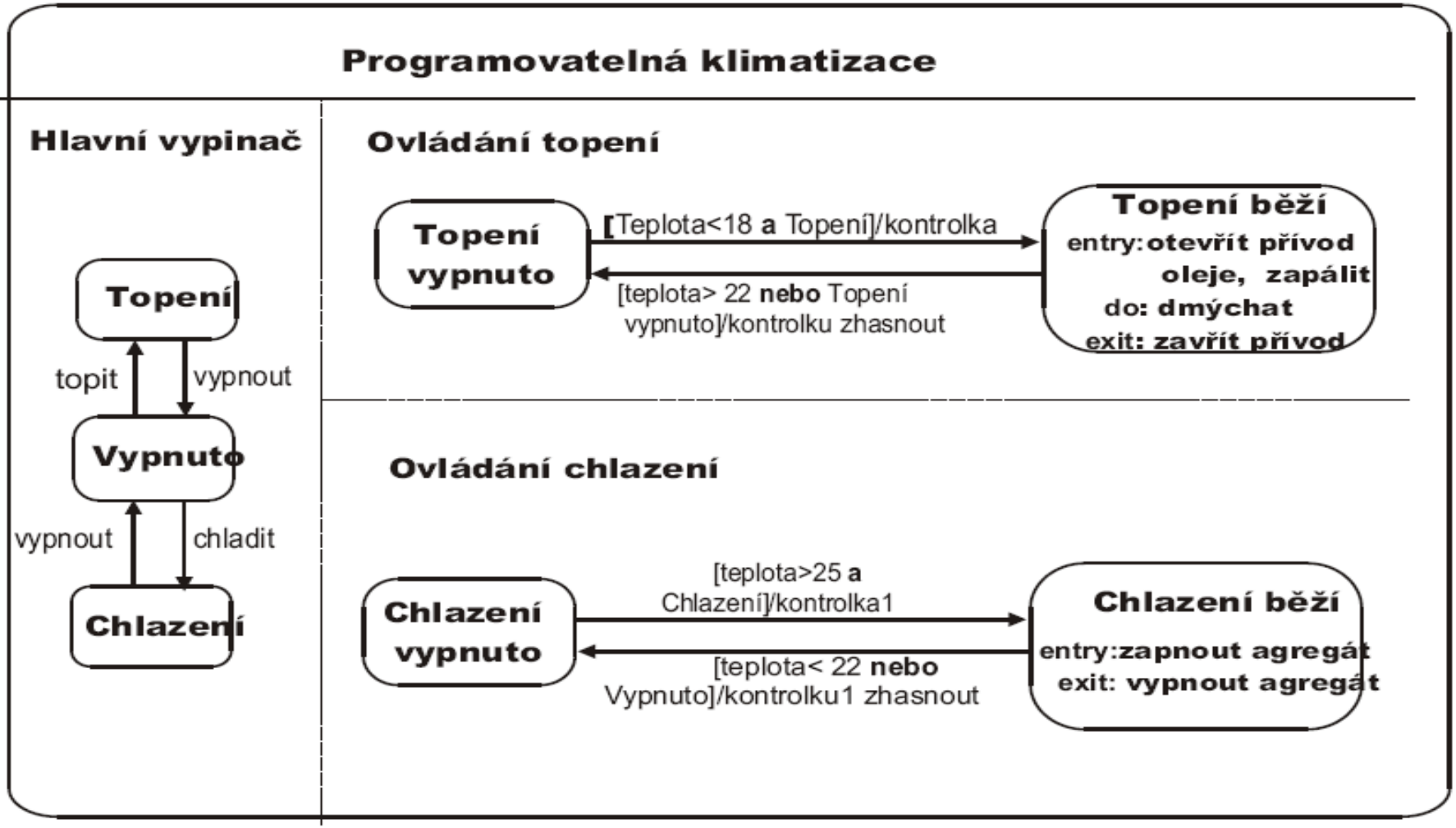
Signatury



Koordinace činností (UML)



Koordinace činností (UML)

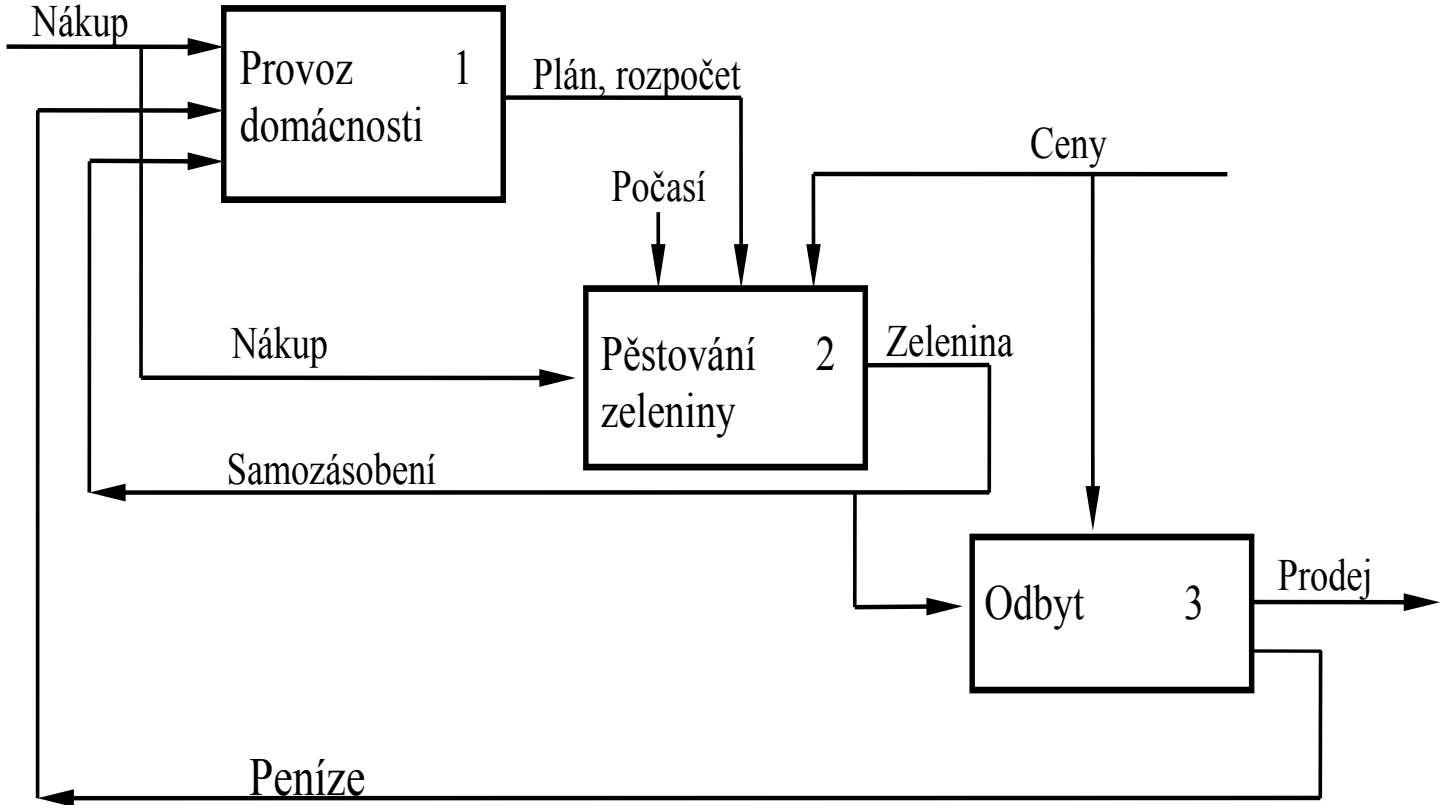


SADT
Structured Analysis and Design Technique

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

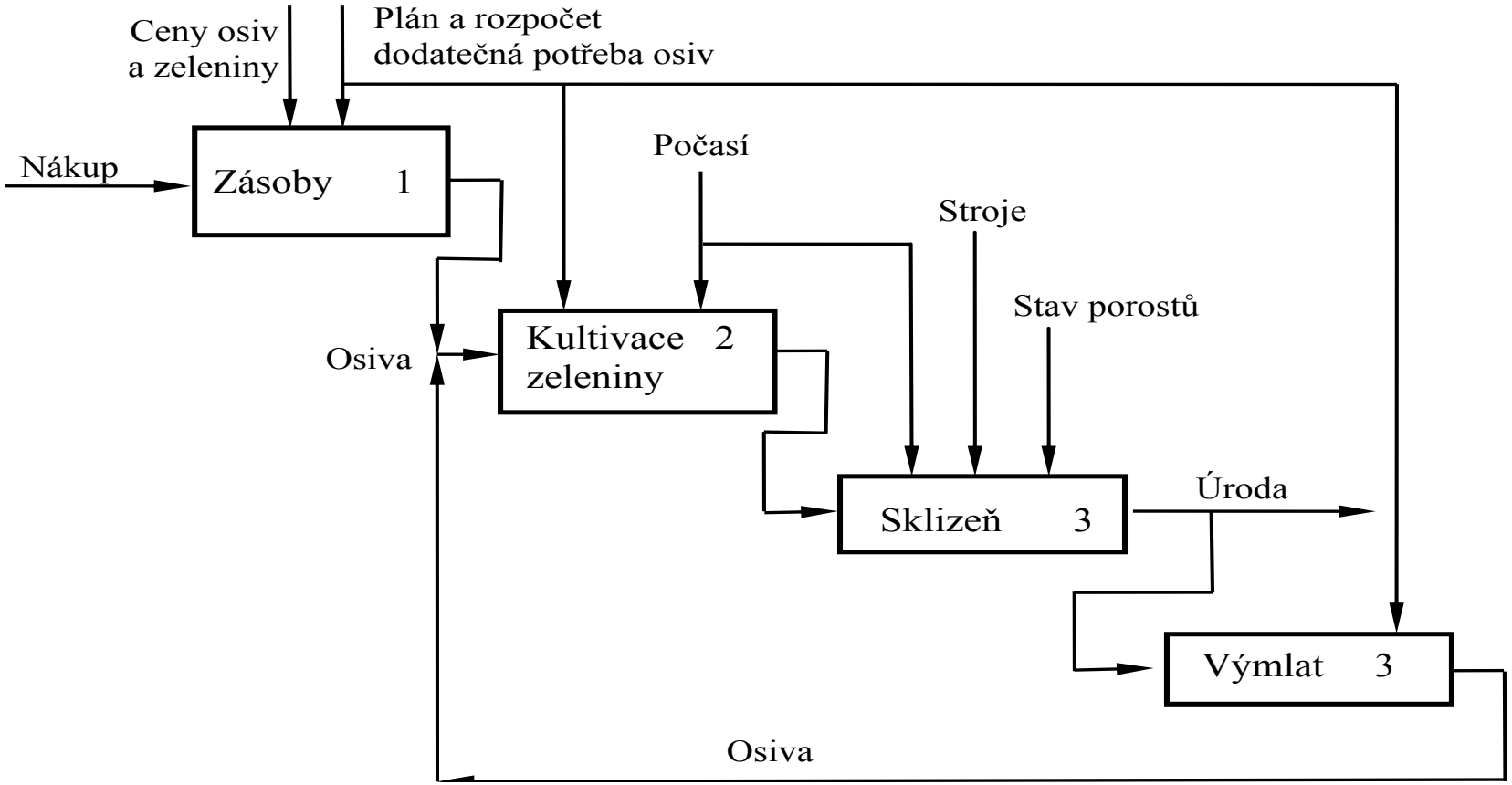
SADT (IDEF0 až IDEF3)

DIAGRAM AKCÍ A0: *Zahradnictví*.

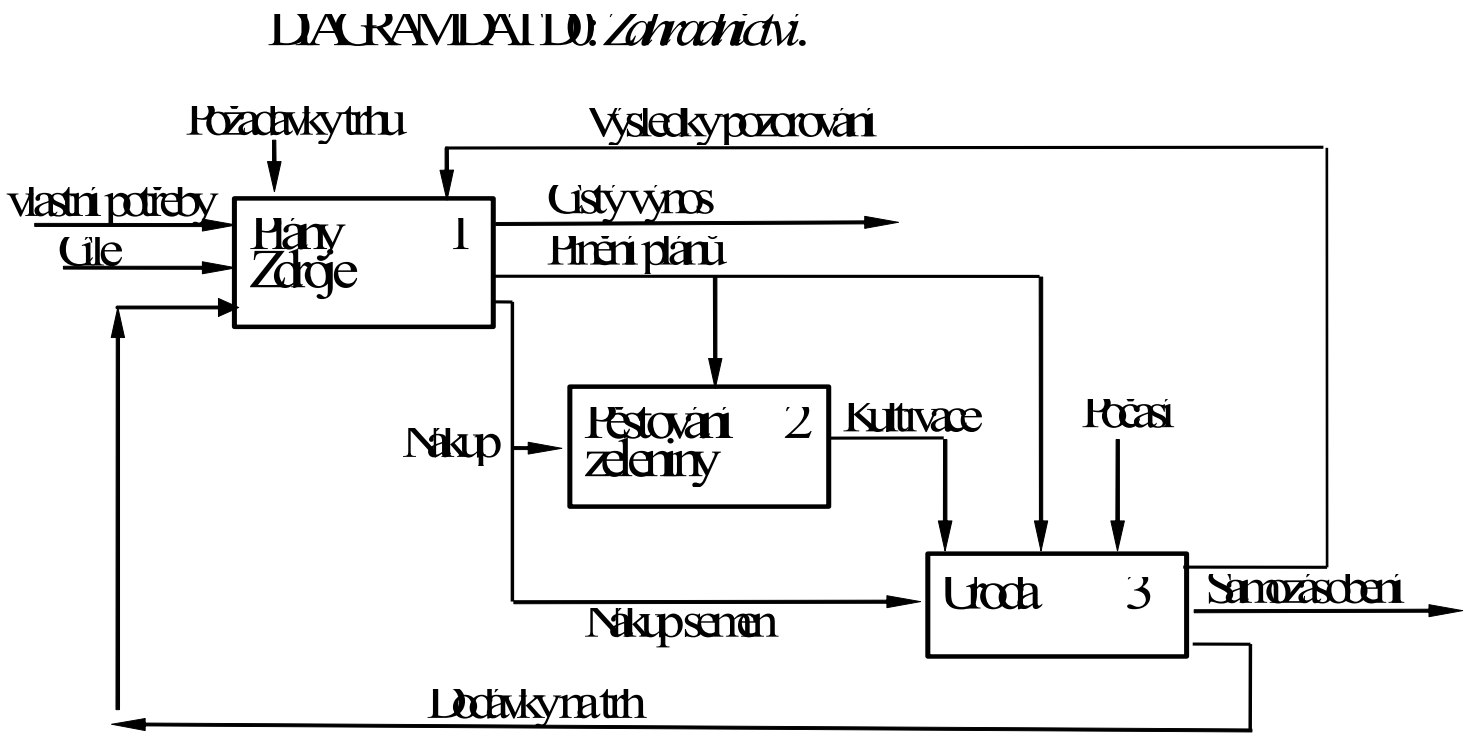


SADT (IDEF0 až IDEF3)

DIAGRAM AKCÍ A0.1 *Pěstování zeleniny*.



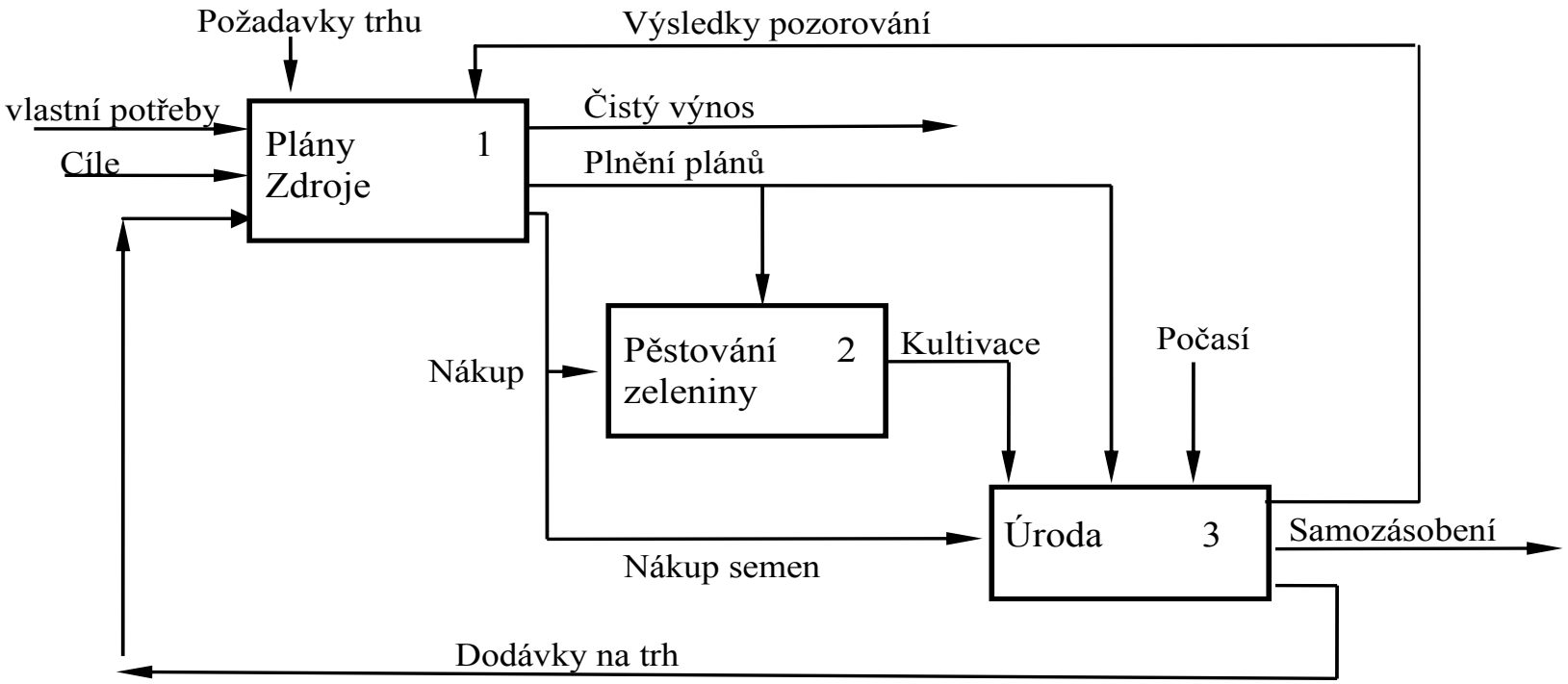
SADT (IDEF0 až IDEF3)



Obr. 12.7.3. 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

Himaláj a Stolová hora

- Kolik investovat do lidí a prostředků (a vlastně i do specifikací). Mám prostředky na n člověkoměsíců programování, prostředky na m člověkoměsíců investuji do podmínek práce, tím zvýším produktivitu $f(m)$-krát.

$f(m)$ by měla růst, musí být $f(0)=1$, tj. žádná investice, žádná změna

Výkon tedy bude

$$Q_n(m) = (n-m)f(m)$$

Pak Q nabývá maxima v bodě, kde

$$0 = -f(m) + (n-m)f'(m)$$

Zvolme $f(m)= 1+cm/n$.

Je to dosti konzervativní odhad, f bývá superlineární.

Přínos bývá i v jiných projektech, investice do specifikací mají podobné efekty

Himaláj a Stolová hora

Po dosazení do vzorce pro derivaci Q dostaneme podmínku pro maximum

$$0 = -(1+cm/n) + (n-m)c/n$$

$$0 = -1 - cm/n + c - cm/n$$

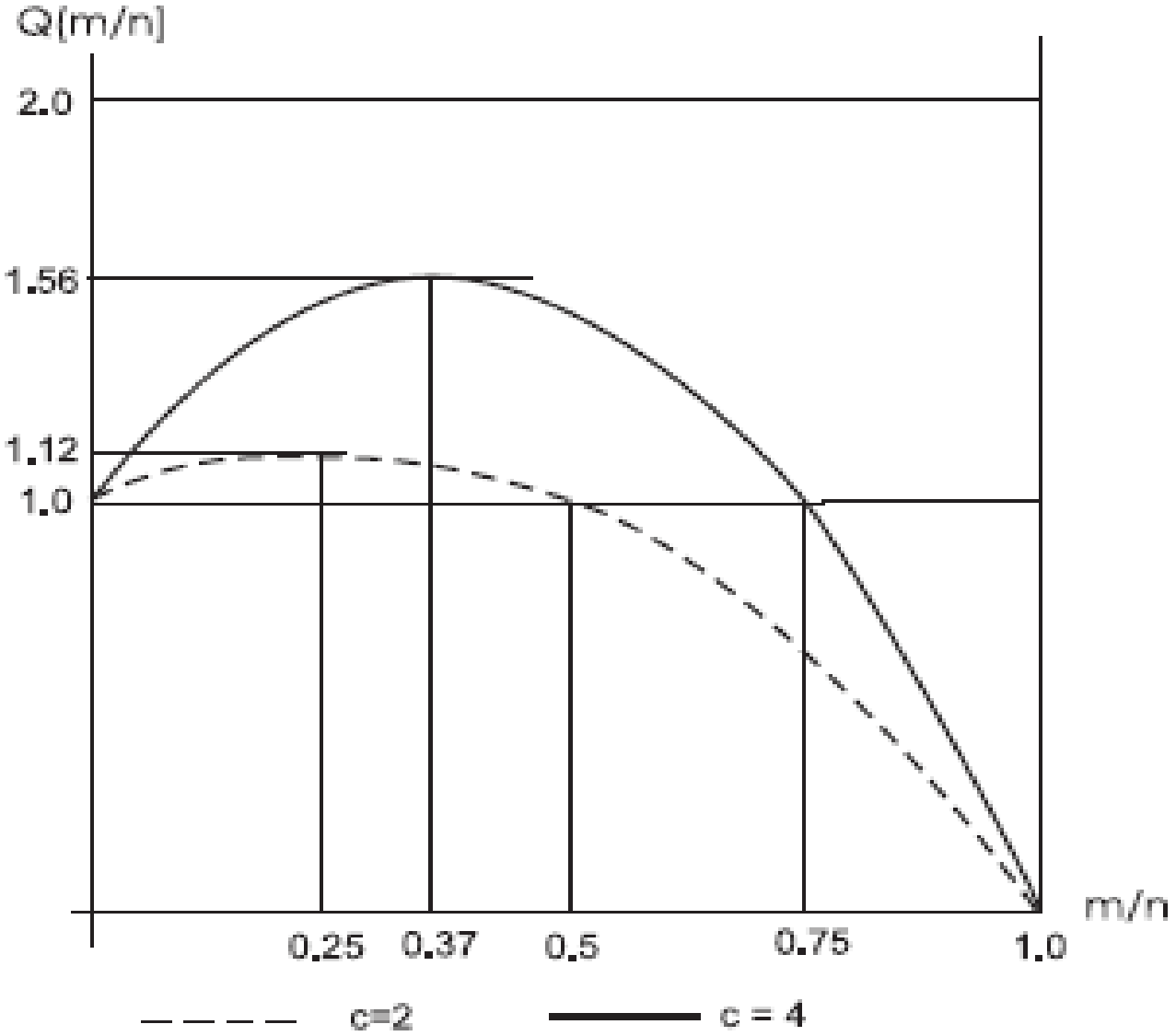
Převedeme-li m/n na levou stranu, dostaneme

$$2cm/n = c-1$$

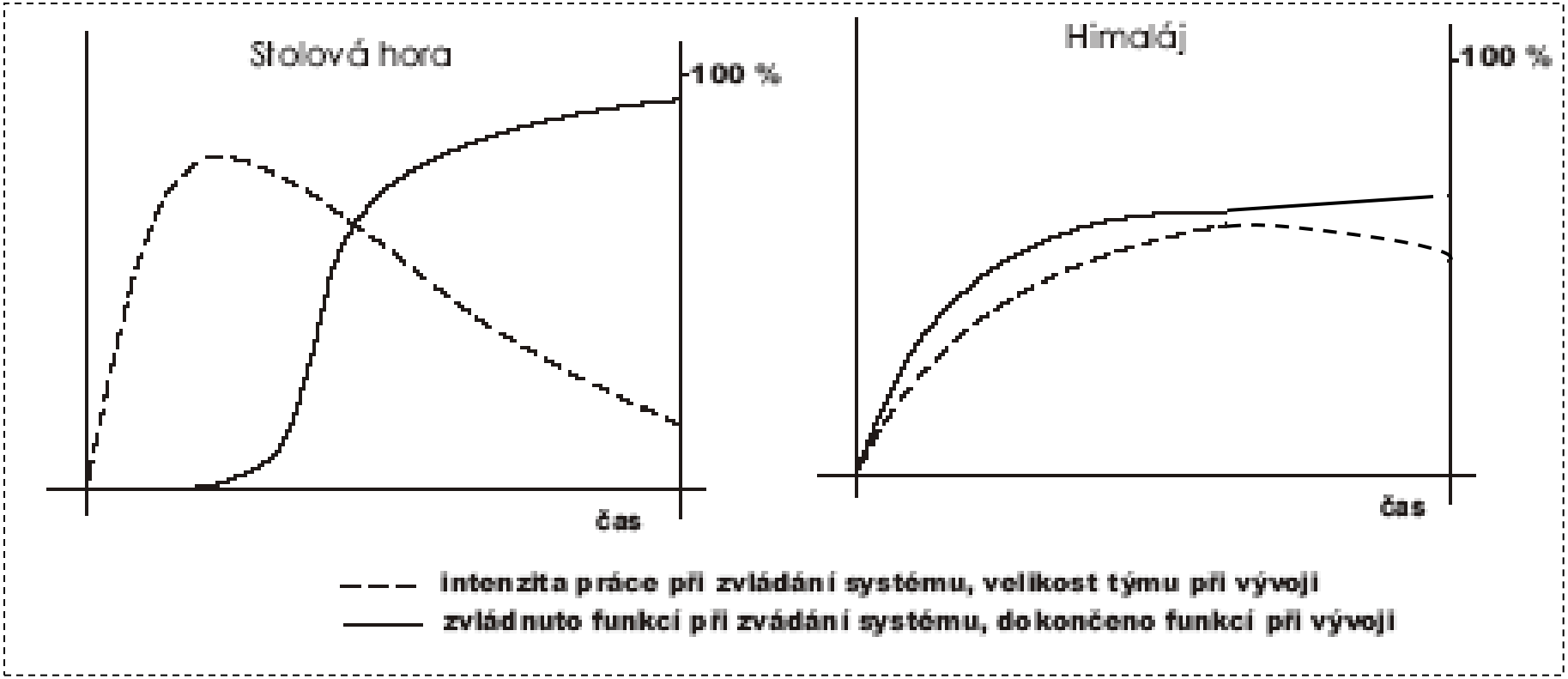
Čili

$$m/n = 1/2 - 1/(2c)$$

Himaláj a Stolová hora



Himaláj a Stolová hora



Stolová hora – dlouho nic, pak prudký výstup na vrchol
Himasláj – začnu brzy stoupat, vrchol stále v nedohlednu

Himaláj a Stolová hora

Tabulka bodů maxima

c	1/2-1/(2c)	$\max(Q(m)/n)$	zvýšení %
2	0.25	9/8	12.5
3	0.33	4/3	33.3
4	0.37	25/16	56.2
6	0.42	49/27	104.2
8	0.44	81/32	153.2

Tab. 12.1: Efekty zvýšení výkonu při použití nástrojů.

Himaláj a Stolová hora

3. Přínos nového nástroje se ovšem většinou neomezuje pouze na daný projekt. Přínosy v dalších projektech mohou být značné. Mnoho se ušetří na údržbě. Příkladem správnosti této úvahy je jazyk C při vývoji prvé verse Unixu.
   - Je tedy třeba dodržovat pravidlo 1/3 na vedlejší výdaje prakticky vždy, kdy je v dosahu nástroj přinášející dostatečné efekty.
4. Doba zvládnutí kupovaného nebo doba vývoje nového nástroje je dána vlastnostmi nástroje samotného. To znamená, že m/n nebude přesně vyhovovat podmínce maxima, takže skutečný efekt bude pak o něco menší, než je uvedeno v následující tabulce.

Himaláj a Stolová hora

Máme-li více nástrojů, pak můžeme postupovat tak, že postupně přidáváme nástroje s náklady m_1, m_2, m_3 atd. Prvý nástroj dá zvýšení produktivity c_1, prvé dva c_2, atd. Implementujeme nebo vyvíjíme tolik a takové nástroje, aby

$$\Sigma_1^n m_i \leq 1/2 - 1/(2c_n)$$

a c_n bylo co největší

Himaláj a Stolová hora

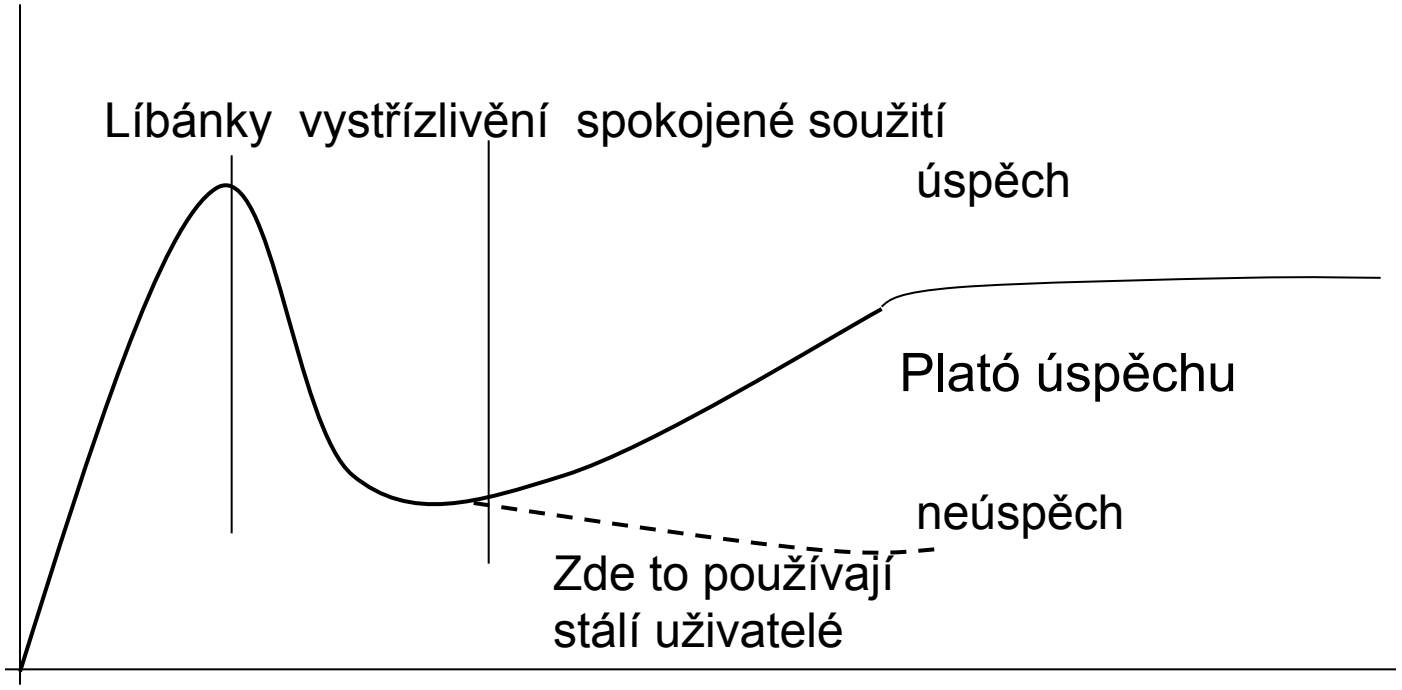
- Funkci $f(m)$ jsme zvolili poněkud spekulativně. K obdobným výsledkům ale dospějeme i pro jiné volby tvaru funkce f.
- Pro větší n si mohu dovolit vývoj složitějších a tedy účinnějších nástrojů, tam lze při správné strategii docílit vynikající výsledky
- Neuvažujeme, že hlavní přínos může být v úspoře nákladů na údržbu (přehlednost, snadnost oprav)
- Nástroje mohou zlepšovat logiku a kromě toho i umožňovat znovupoužitelnost a efektivitu
- Je třeba rozvíjet prostředí a nástroje (vývoj, nákup), musím ale na to mít schopné lidi, *stejný efekt ale může mít investice do znalostí lidí*

Modernost metodiky

- Nový nástroj se zprvu používá tam, kde se staré přístupy neosvědčily, proto jsou výsledky zprvu skvělé. Další důvod neúspěchu je, že ho používají inovátoři a ne běžní uživatelé (podobné efekty existují i ve školství)
- Pak se narazí na meze a inovátory to navíc již nebaví a přijde rozčarování, někdy neoprávněné
- Nakonec upadne nástroj buď do zapomnění, nebo se osvědčí a je rutinně používán – plató úspěchu. Někdy to trvá řadu let (u objektové orientace to bylo více než deset let)
- Je třeba být u novinek zdravě ale ne přehnaně skeptický

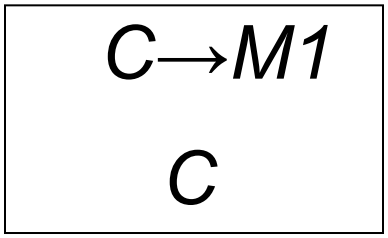
Modernost nástroje

- Funkce množství positivních ohlasů



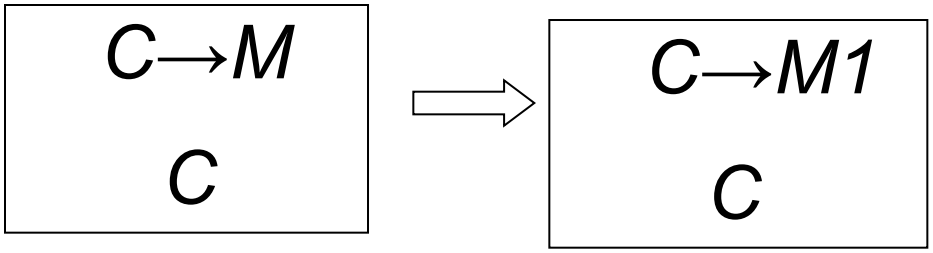
Dobrý nástroj je vždy výhodou

Kompilátor

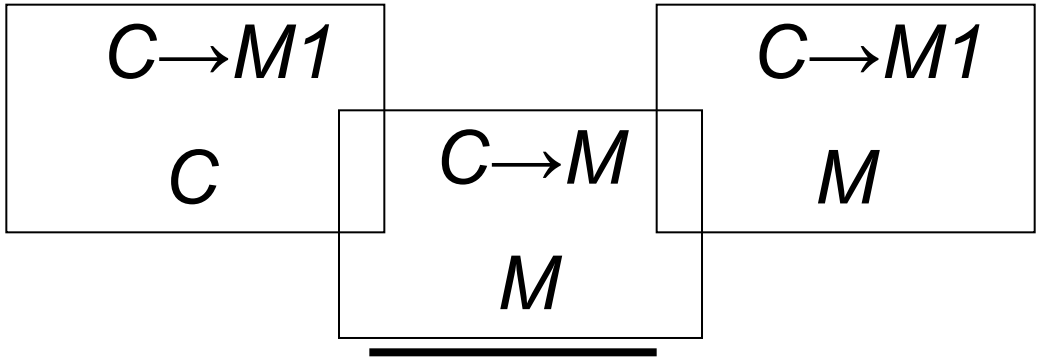


Kompilátor
z C zapsaný v C

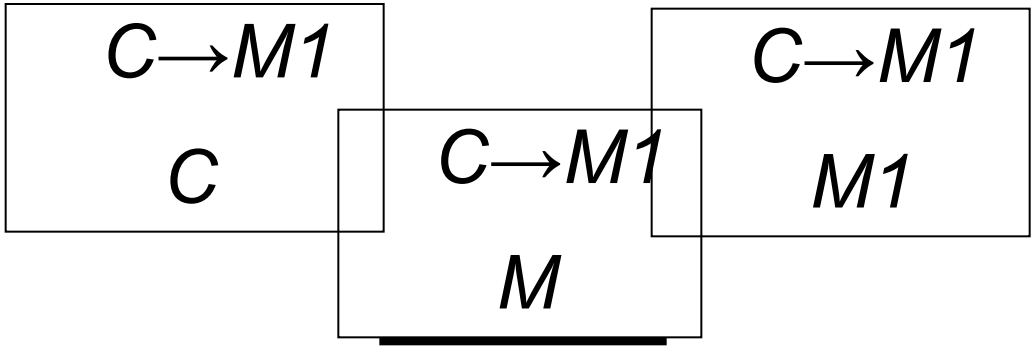
Přenos kompilátoru



Přepíše se generátor kódu jen zčásti automatizovaně

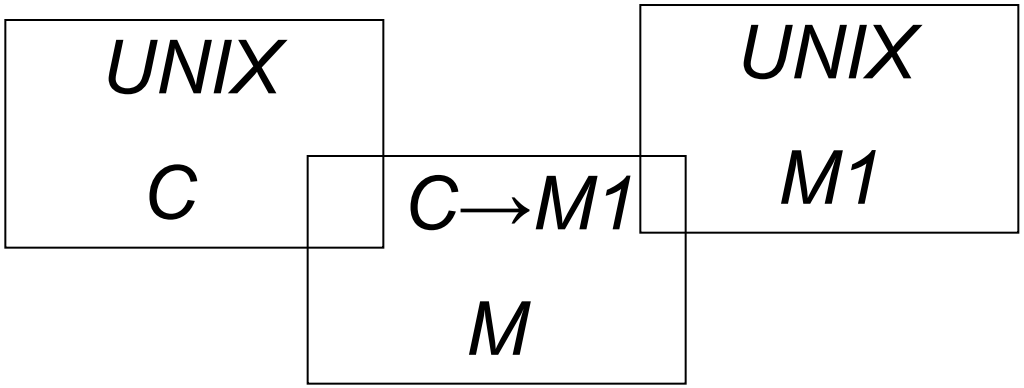


Kompilátor v C přeložen C kompilátorem běžícím na M, získán Křížový kompilátor běžící na M překládající pro M1



Cílový kompilátor

Přenos UNIXU

- 

Je ale nutné přepsat drivery,
cca 10% nákladů

Etapy zvládání 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 zvyknouPřibývá 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

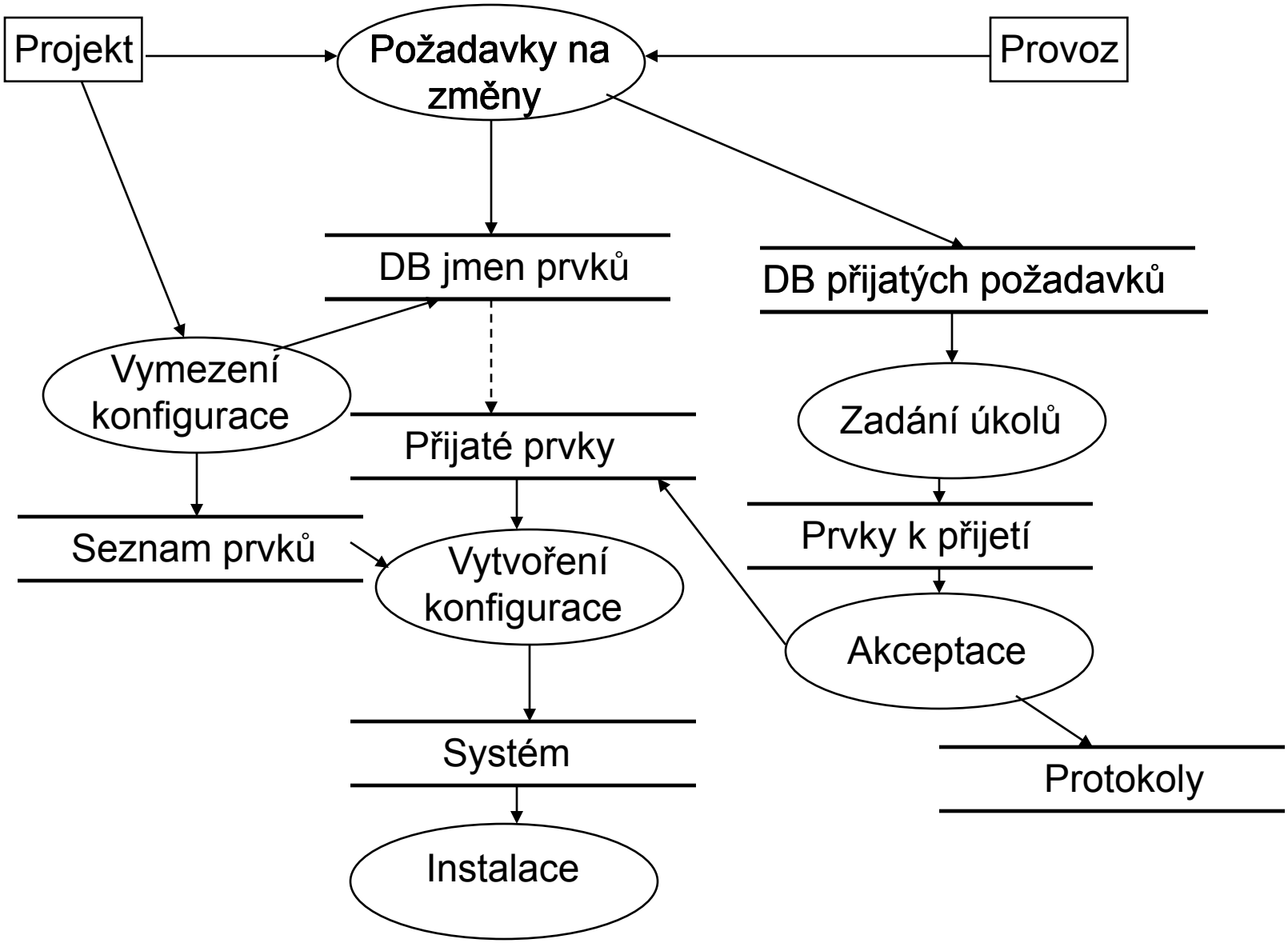
ISO10007 – obecně řízení konfigurace

ISO 12207, ISO 90003, ISO 15846, ISO 20000 - software

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


Projekt
Požadavky na změny
Provoz
DB jmen prvků
DB přijatých požadavků
Vymezení konfigurace
Přijaté prvky
Zadání úkolů
Seznam prvků
Vytvoření konfigurace
Prvky k přijetí
Akceptace
Systém
Protokoly
Instalace

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 standardizoat
  - 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 ale díky jazyce Simula, jazyka pro programování simulací, Simula byla jazyk prvého modelu, simulátor se používal při testech správnosti nastavení kardiostimulátorů)
- Nejen vlastnosti jazyka, ale také vývojového prostředí
- Důležité, jak se jazyk uplatní v podpoře SW architektur (COBOL a dataflow diagramy)
- 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, blokované 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 aplikaci.

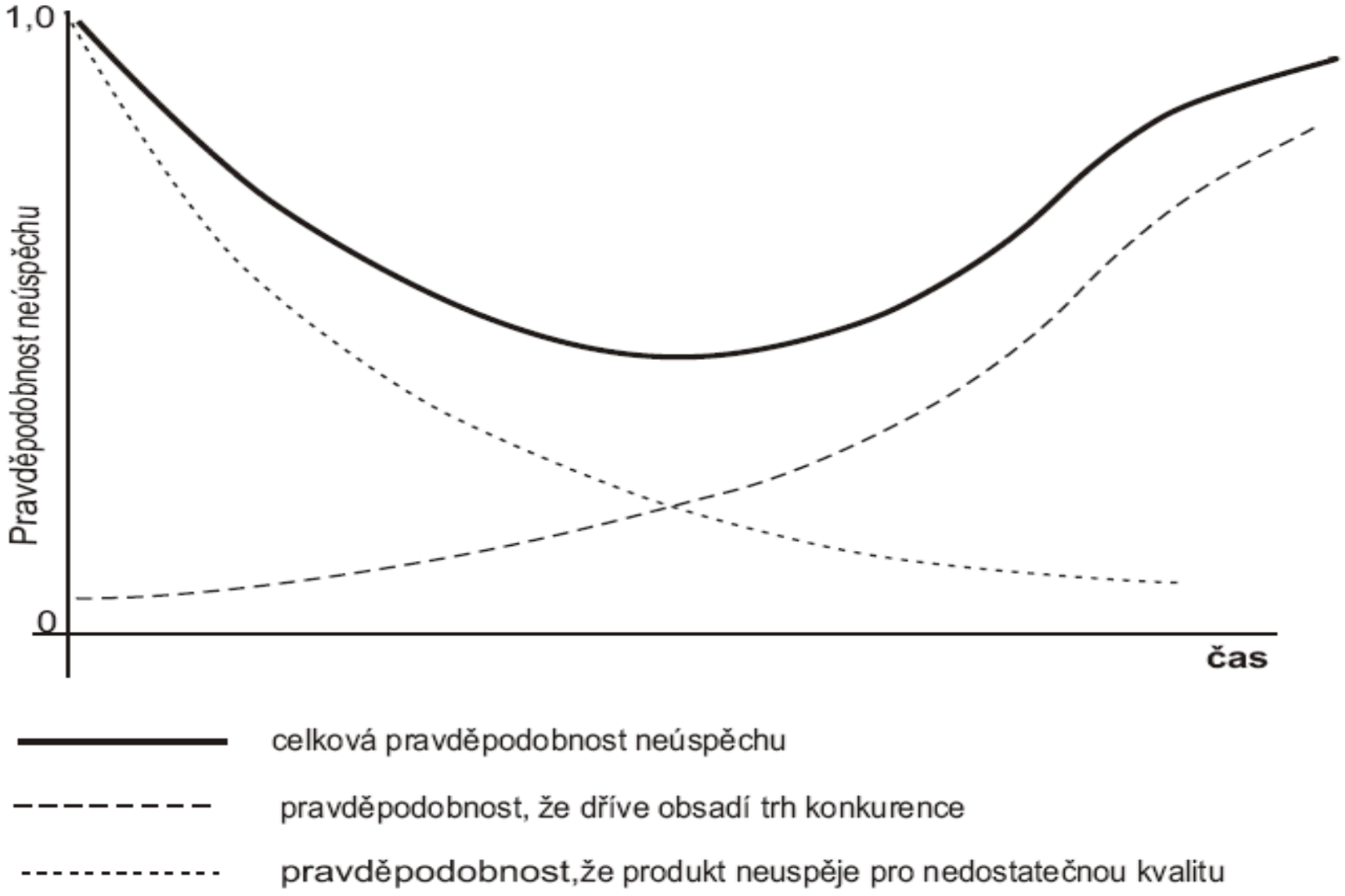
Důvody: Churchova téze, co nejde algoritmizovat na Turingově stroji, nelze vůbec, složitost reálného světa – ten není počítačem, je náhodný v principu (kvantová mechanika)

- 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

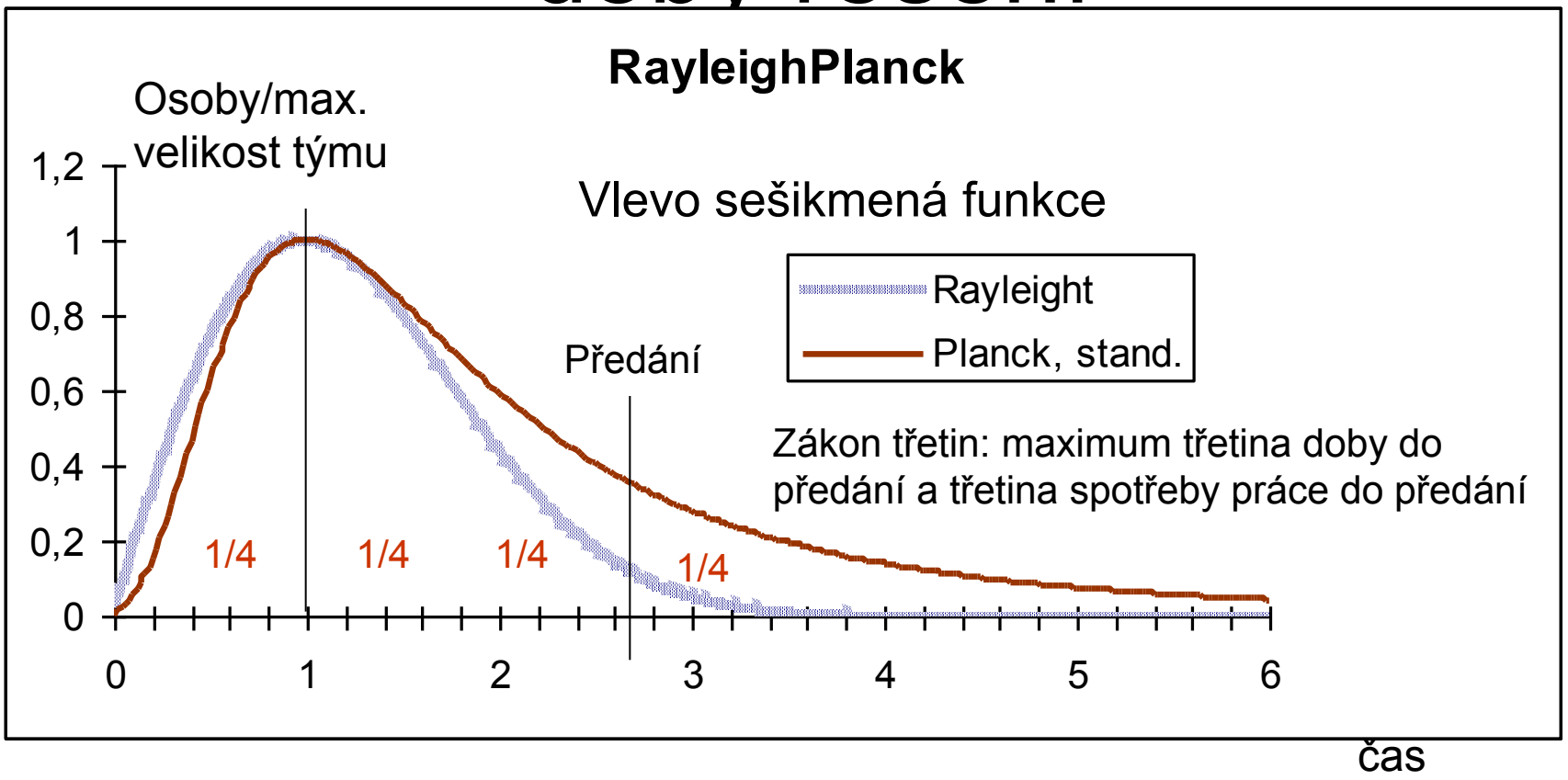
Testování

- 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



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 intensitě práce

Nelze zero defect software

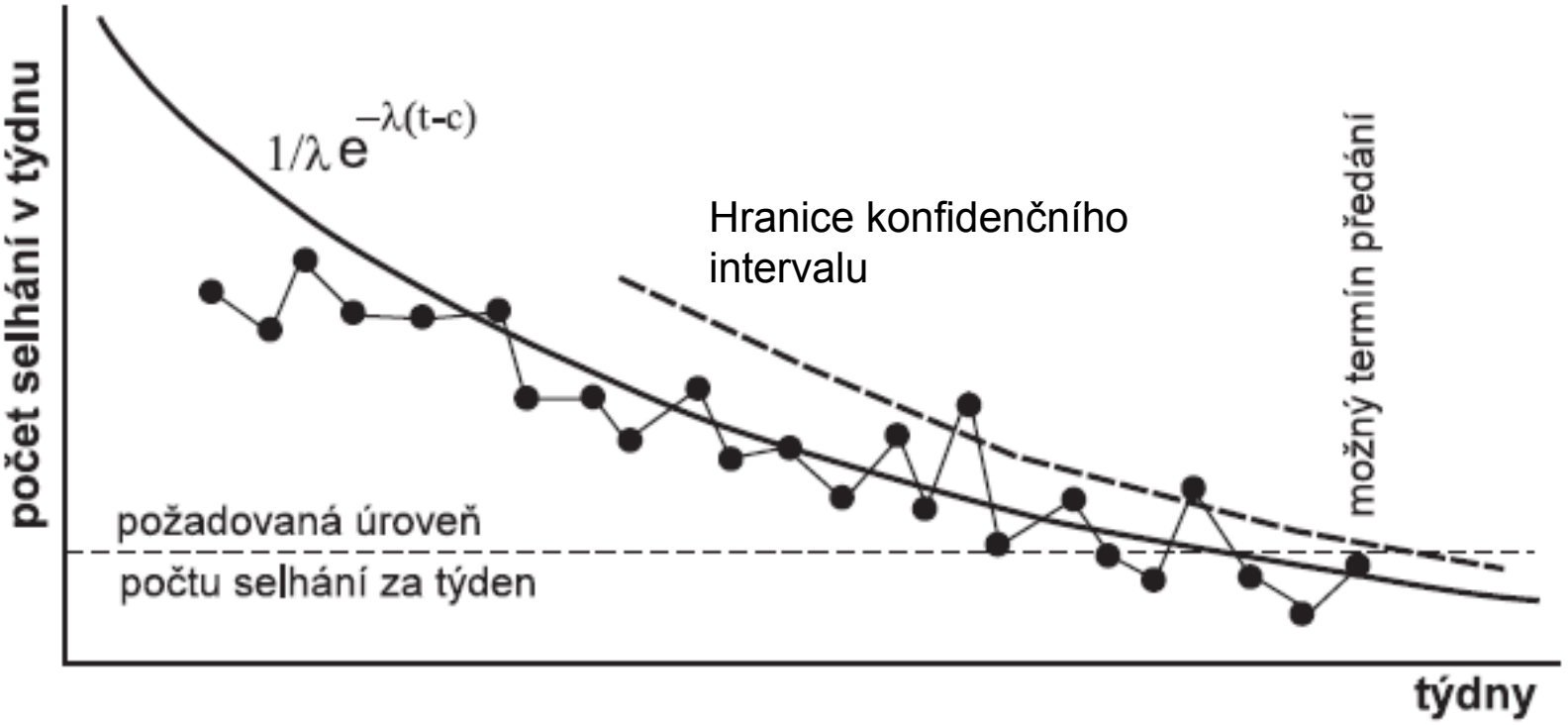
Ale jak řídit, když průběh funkcí neznám?

- 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 vyroste 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

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
- **Systému**
  - 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

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**: Síť testových případů
- **Test**: Síť 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 a testuje jako celek
  - Standardní je regresní testování (opakování podstané části dosud provedených testů)

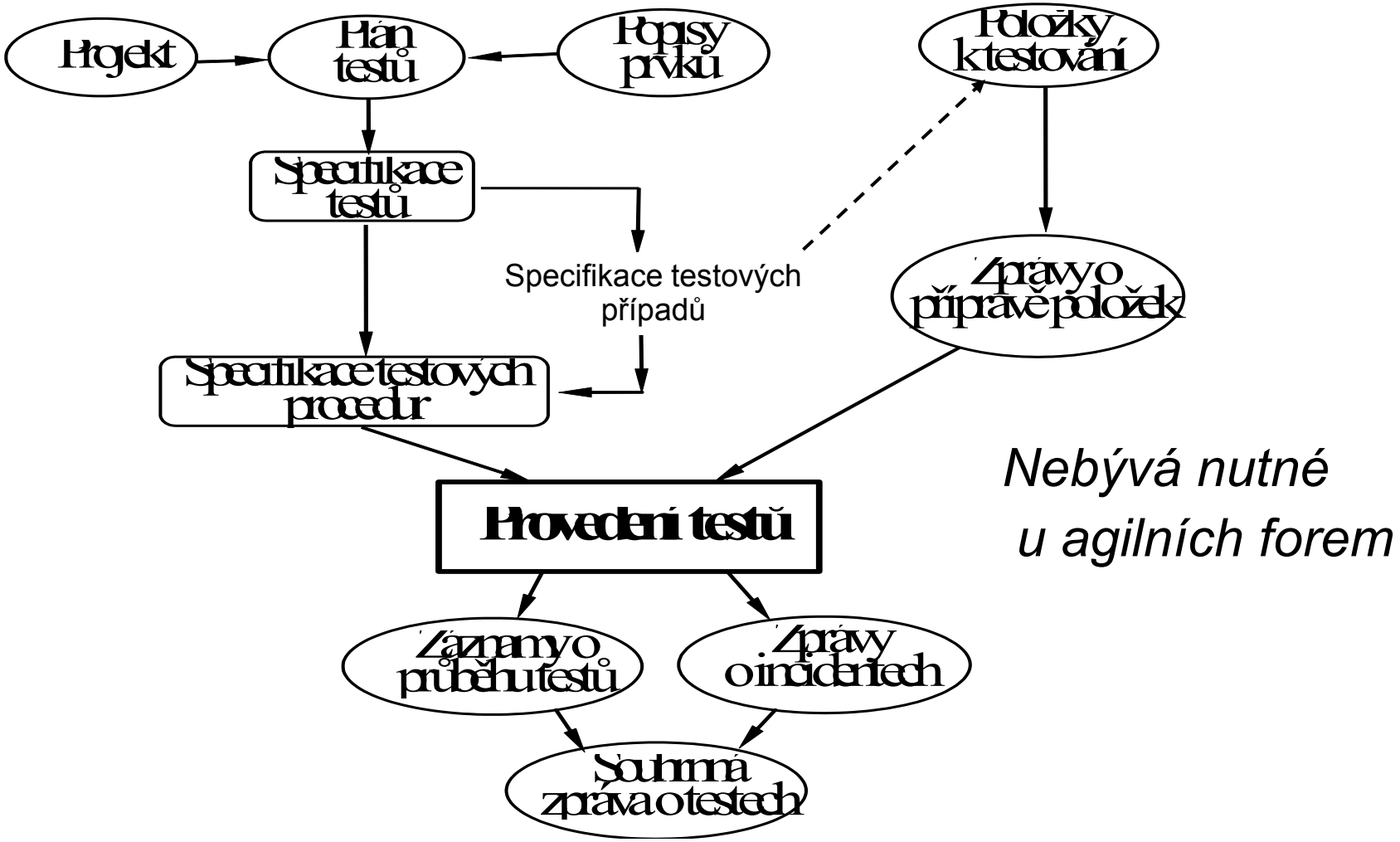
Specifikace testů

- Id
- Podmínky a způsob provedení
- Popis testu, testové procedury
- Kriterium přijetí/zamítnutí testů
- Kriteria pro přerušení 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
- **Nemají být 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)

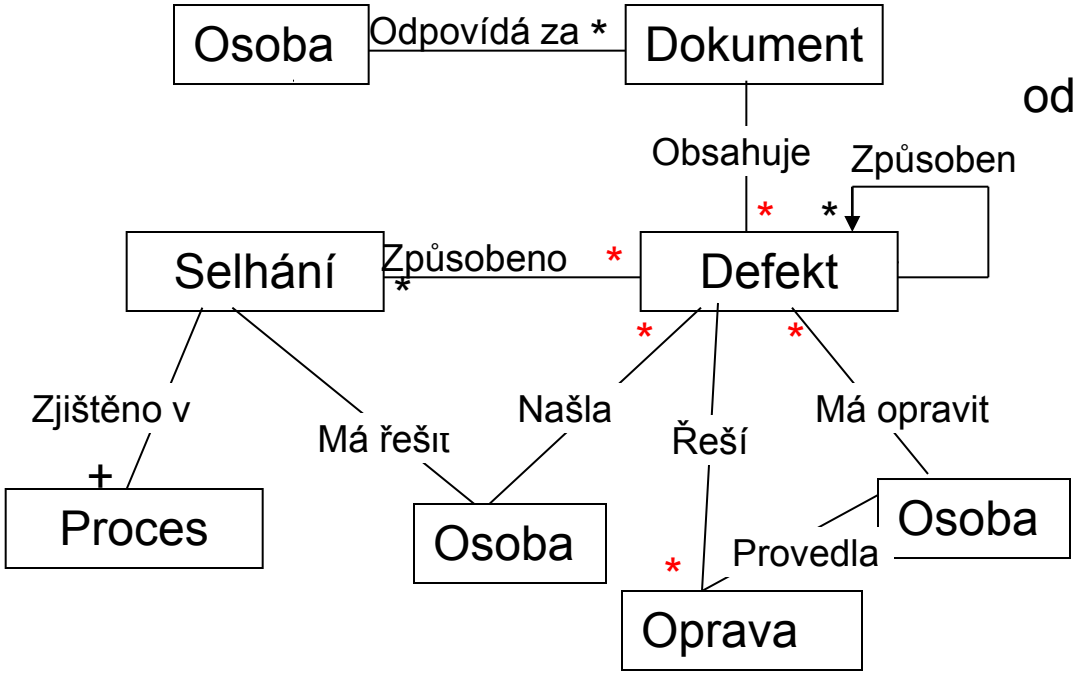
Datová báze výsledků testů (selhání) by měla být stejná jako u oponentur a u reklamací,

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

Brno 3.5

- Bylo vynecháno UML, rozhodovací tabulky, SADT

Zavedení

Jak instalovat a zavádět

Na koho se obrátit

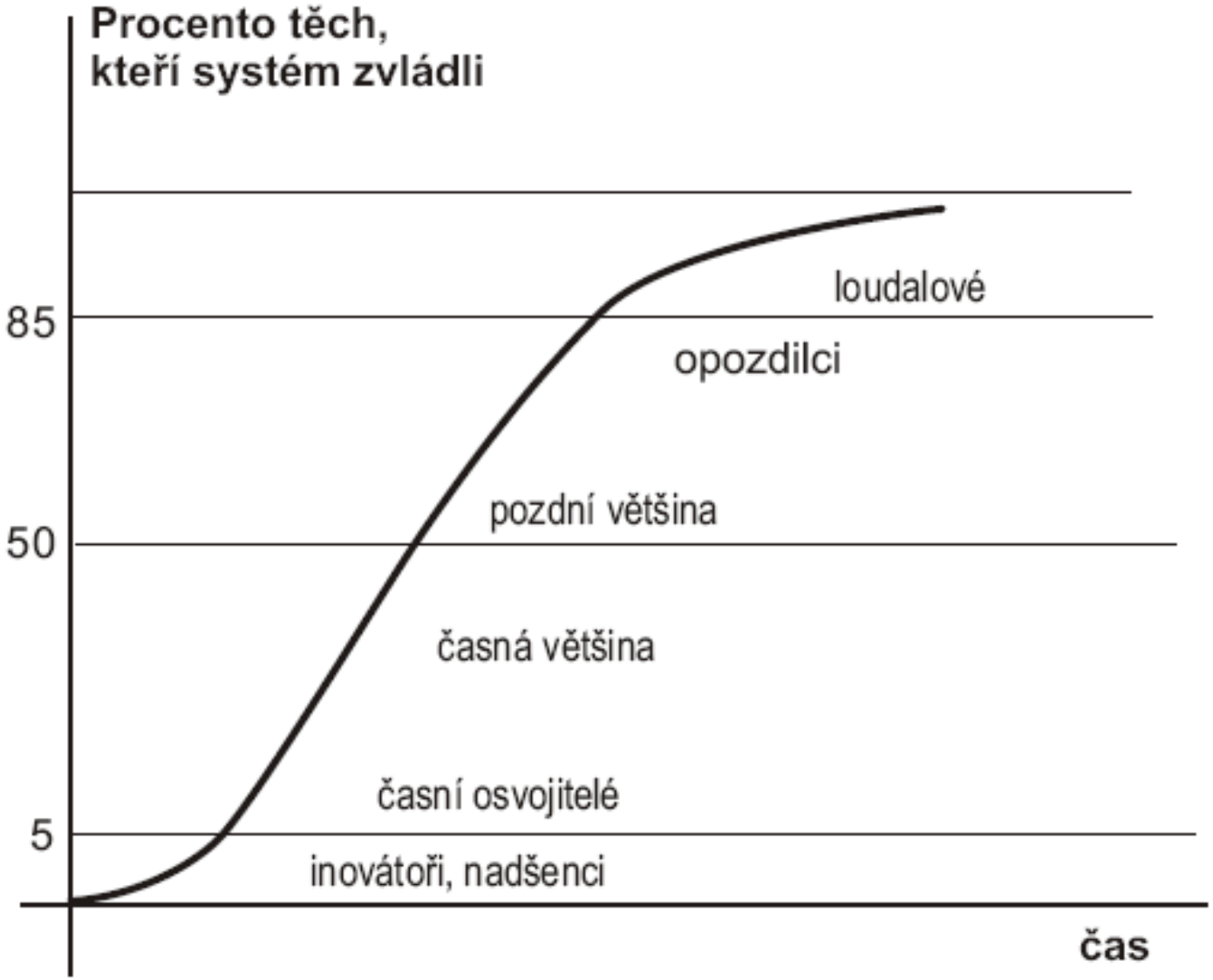
Zavedení informačního systému

- Předání (instalace, předávací testy, zkušební provoz)
- Školení pracovníků
  - Spoluúčast tvůrců (nebývají ale pedagogicky zdatní, nevyužívá se jejich odbornost)
  - Vhodní jsou dobří lektoři, 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

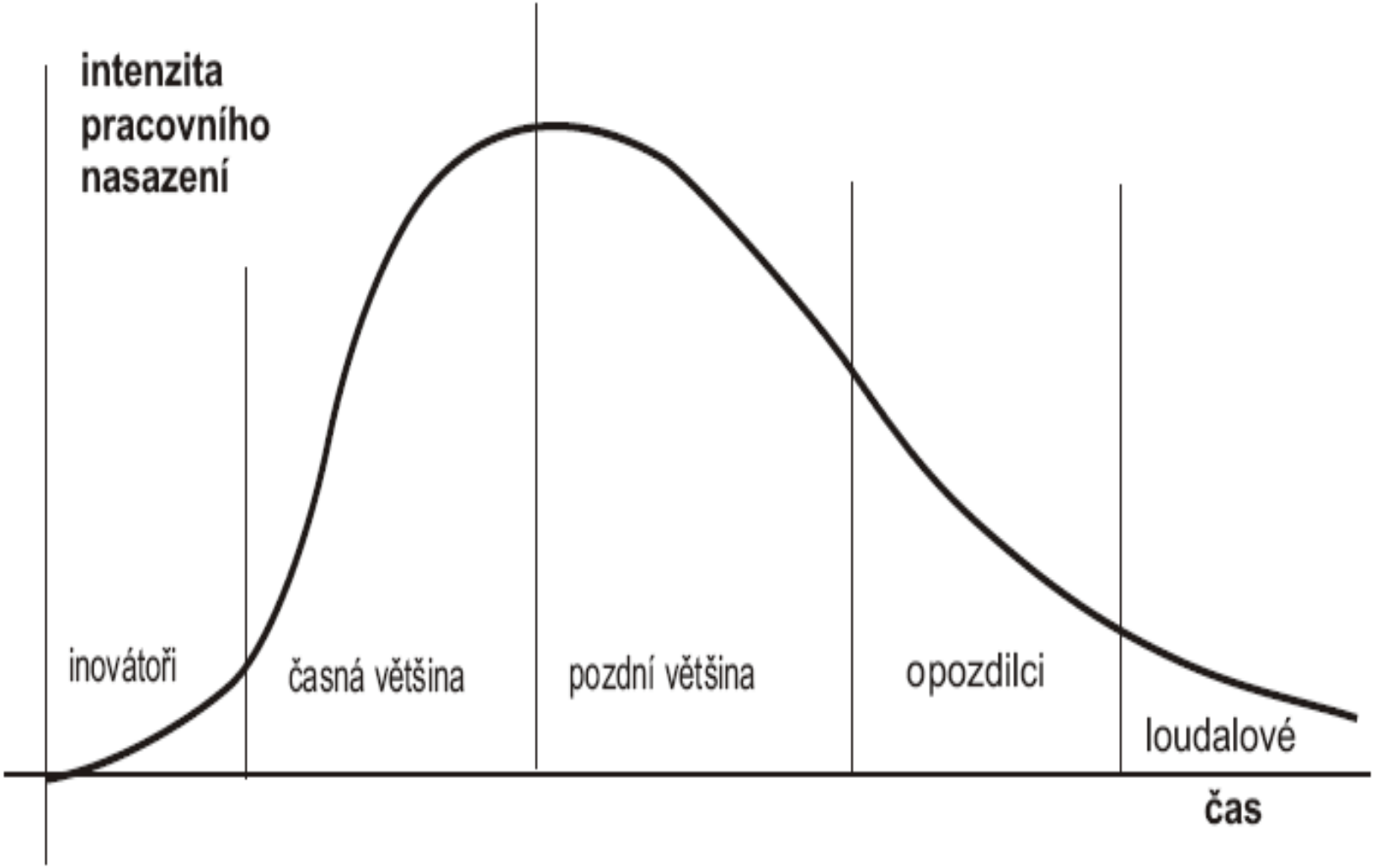
Dokumenty při zavádění

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

Křivka učení a typy uživatelů



Křivka učení a typy uživatelů

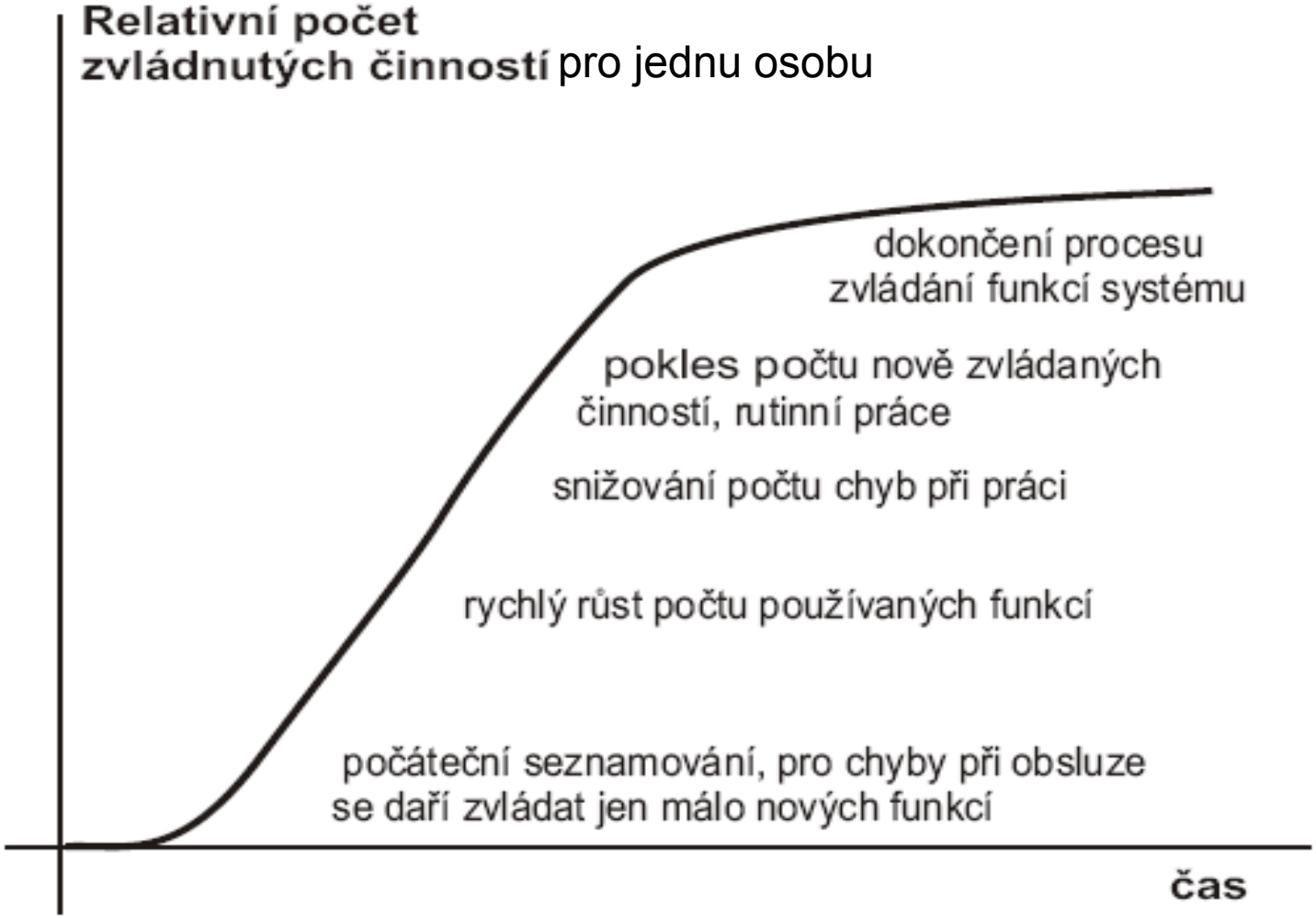


Koho získat jako spojence

- 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

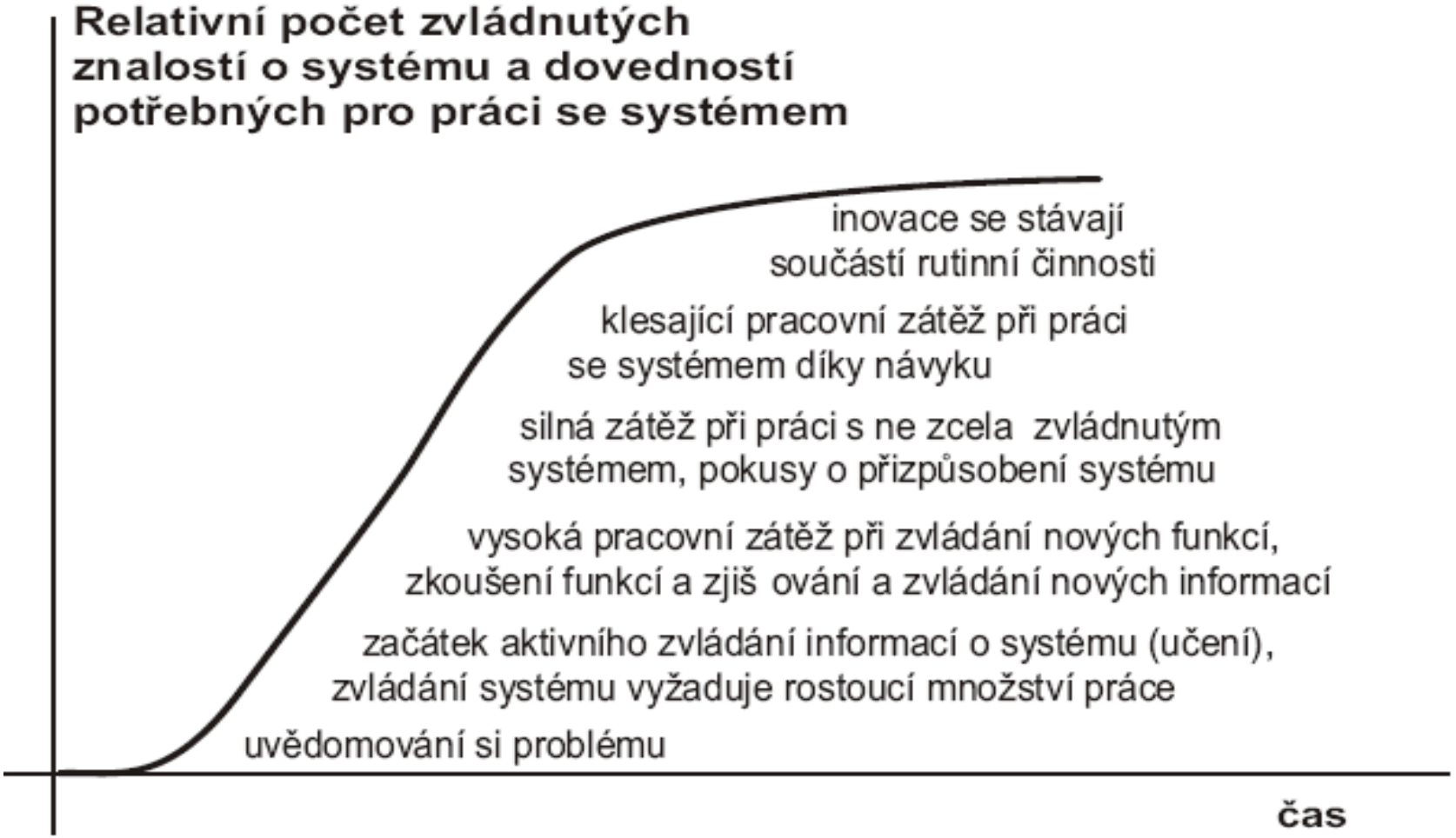
Křivka učení

- Nemá být příliš strmá (intensita učení příliš vysoká) ani příliš plochá
- Je nutné zvládání 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

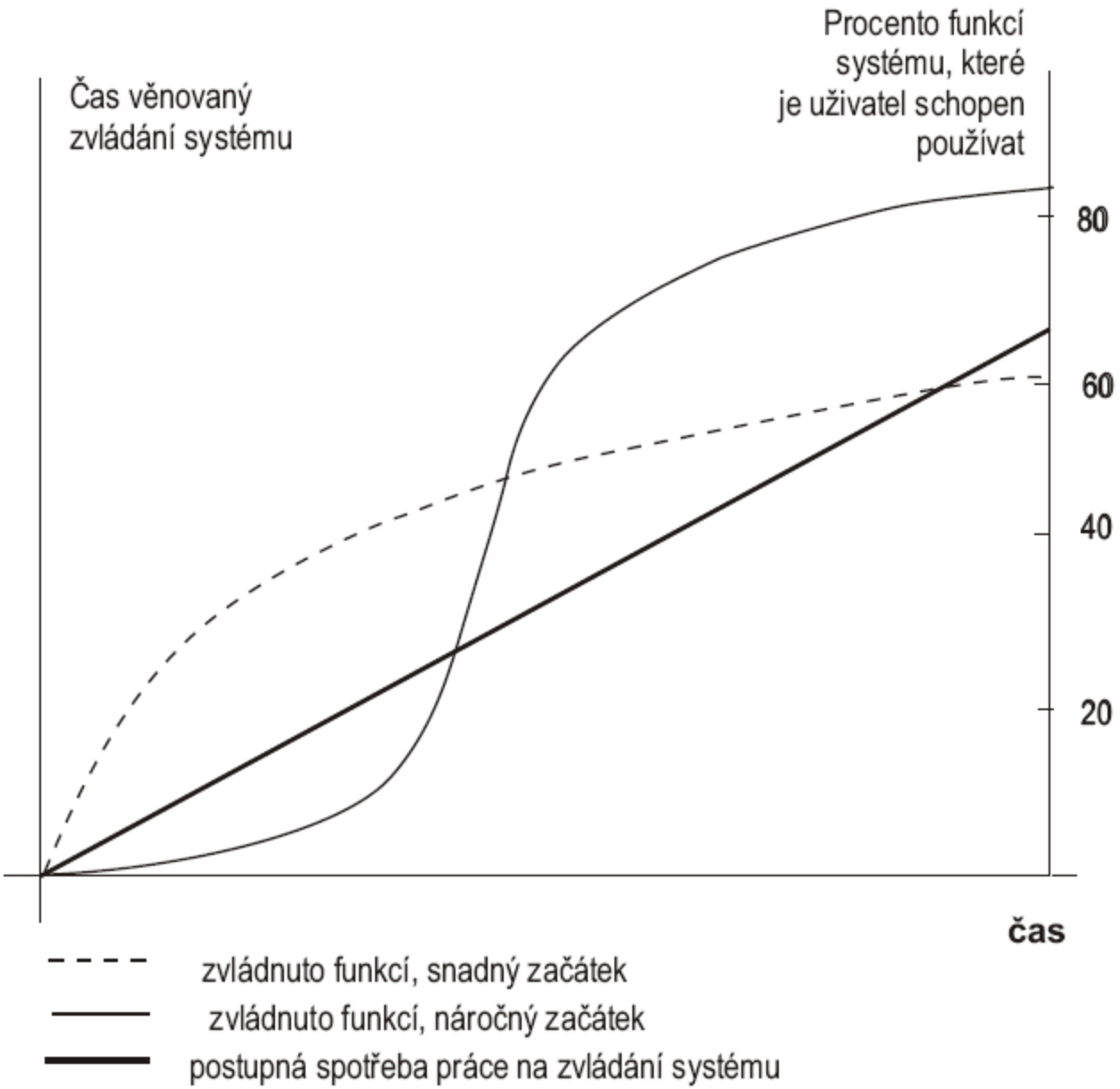


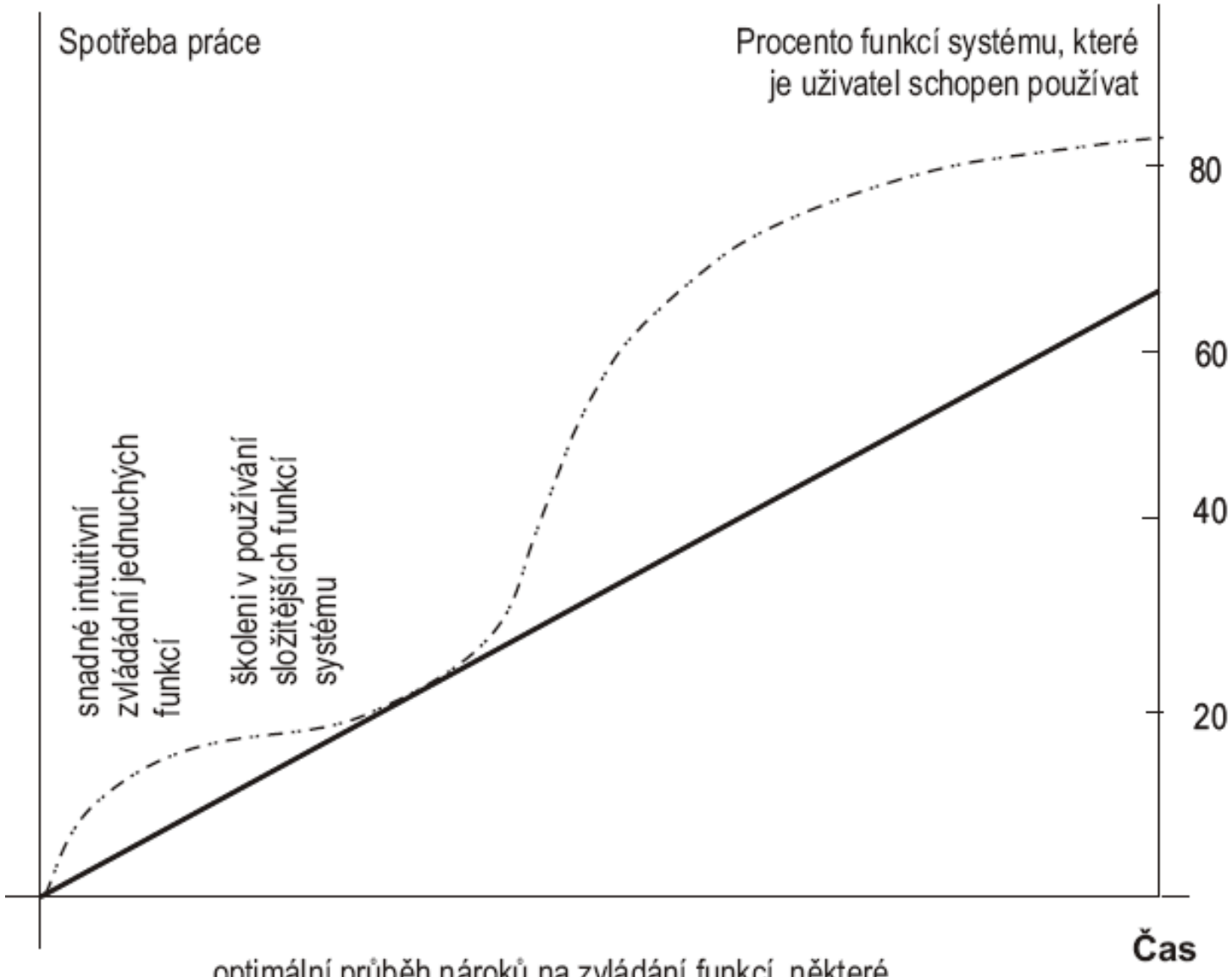
Učení metodou pokusů a omylů

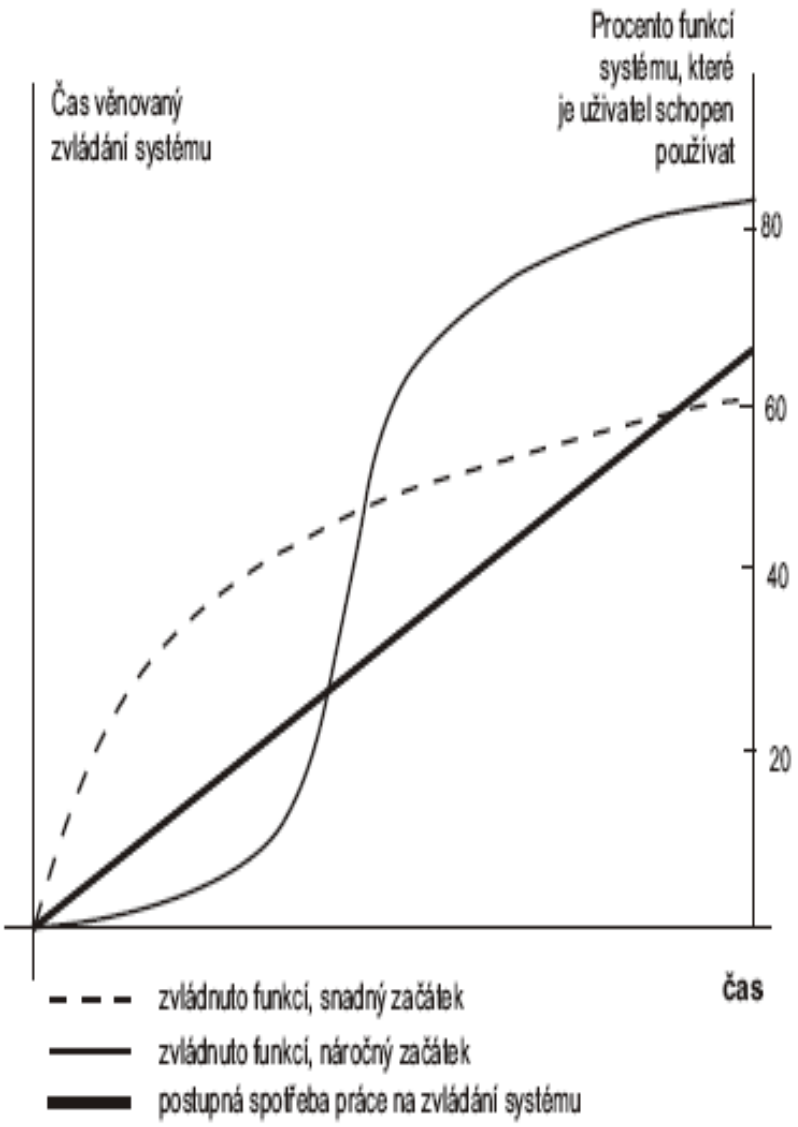
Křivka učení a typy zvládání

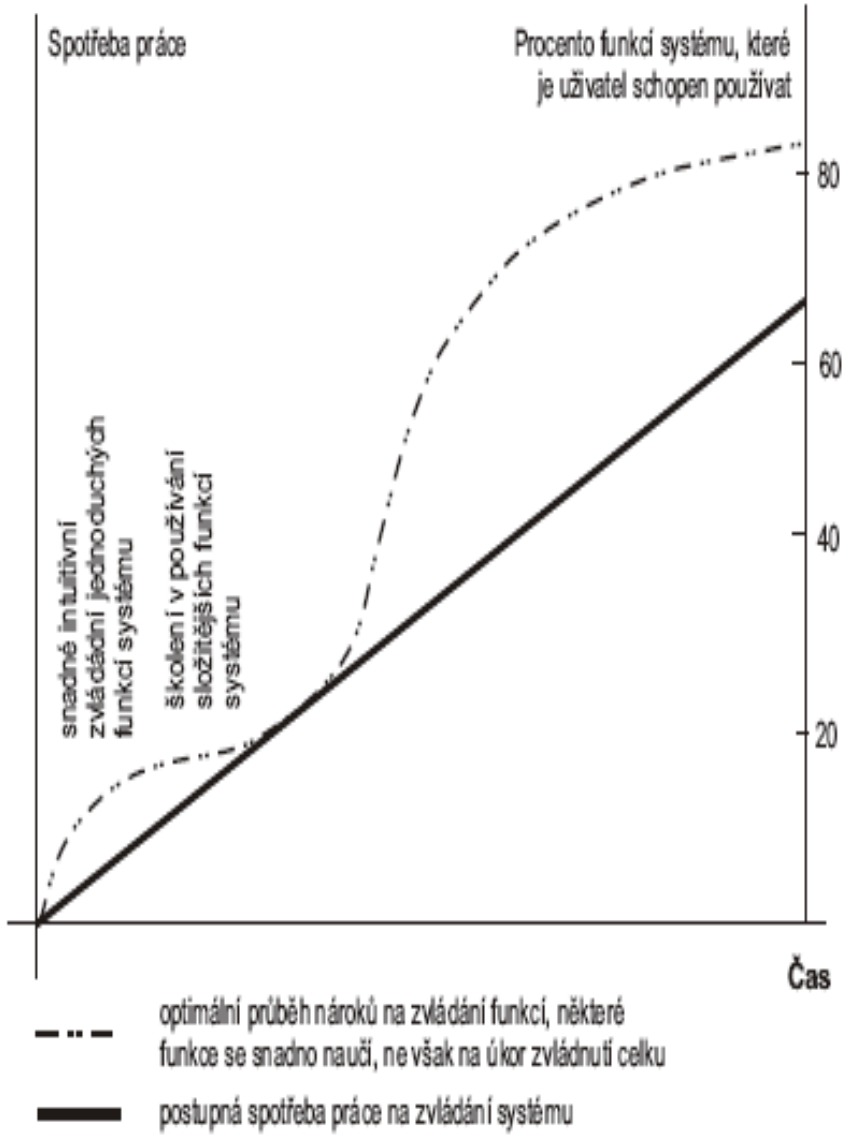


Charakter činností pracovníků při zvládání systému.


Procento funkcí systému, které je uživatel schopen používat
Čas věnovaný zvládání systému
80
60
40
20
čas
zvládnuto funkcí, snadný začátek
zvládnuto funkcí, náročný začátek
postupná spotřeba práce na zvládání systému


Spotřeba práce
Procento funkcí systému, které je uživatel schopen používat
80
60
40
20
snadné intuitivní zvládání jednuchých funkcí
školeni v používání složitějších funkcí systému
Čas
optimální průběh nároků na zvládání funkcí, některé funkce se snadno naučí, ne však na úkor zvládnutí celku
postupná spotřeba práce na zvládání systému


Čas věnovaný zvládání systému
Procento funkcí systému, které je uživatel schopen používat
80
60
40
20
Čas
zvládnuto funkcí, snadný začátek
zvládnuto funkcí, náročný začátek
postupná spotřeba práce na zvládání systému


Spotřeba práce
Procento funkcí systému, které je uživatel schopen používat
snadné intuitivní zvládání jednoduchých funkcí systému
školení v používání složitějších funkcí systému
80
60
40
20
Čas
optimální průběh nároků na zvládání funkcí, některé funkce se snadno naučí, ne však na úkor zvládnutí celku
postupná spotřeba práce na zvládání systému

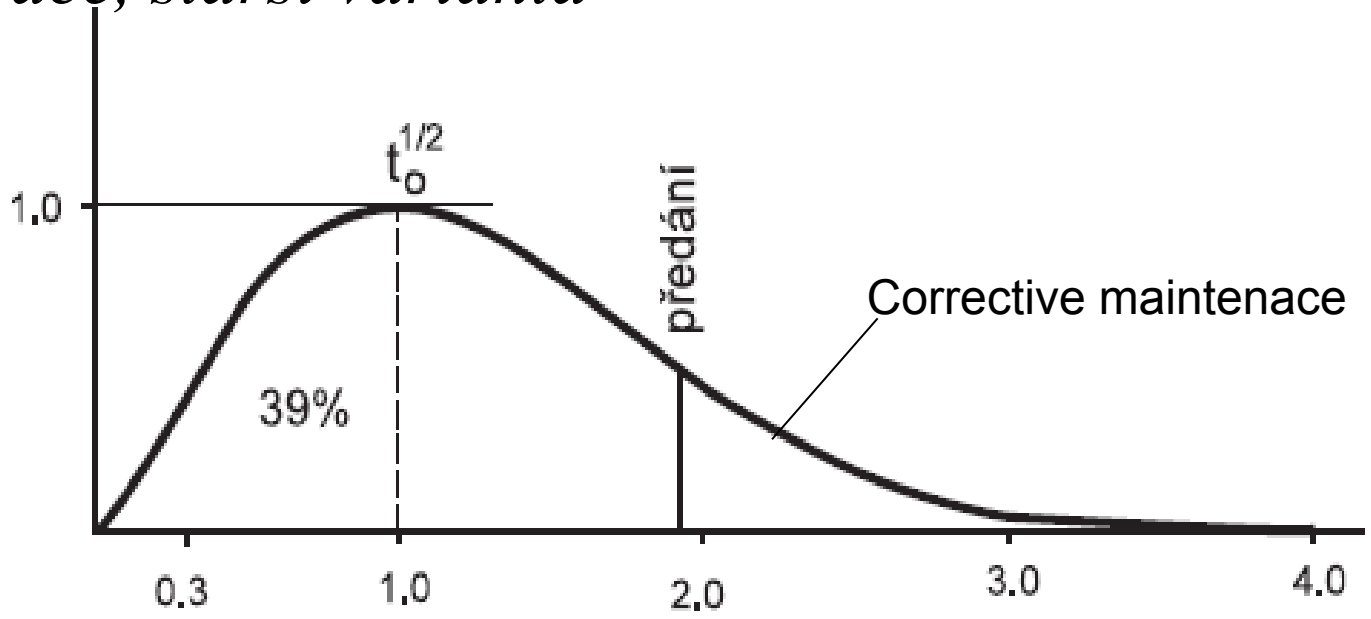
Údržba

I instrukce se z logického pohledu (potřeb) opotřebují

Druhy údržby

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

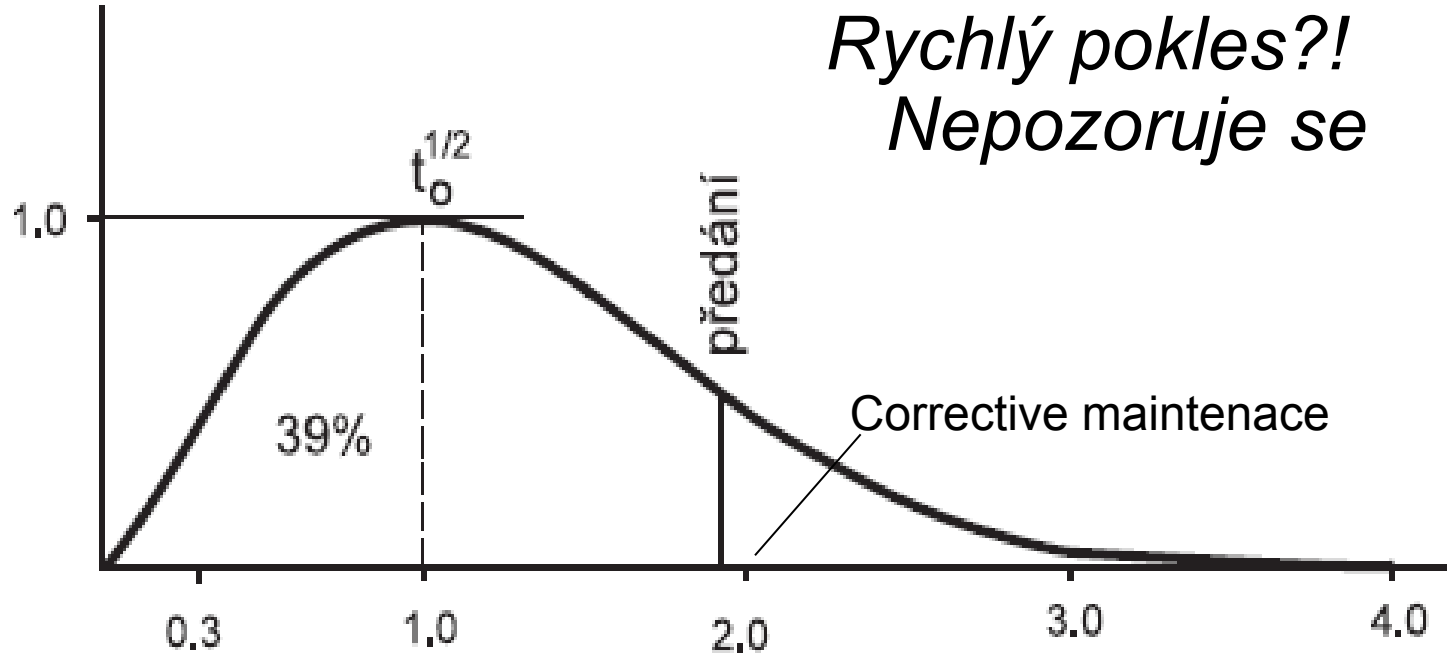
Model průběhu velikosti týmu a tedy intensity práce, starší varianta



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.

$$team(t) \mathrel{\widehat{=}} K \cdot t/t_0 \cdot \exp(-\frac{t^2}{2t_0})$$

Rychlý pokles?!
Nepozoruje se



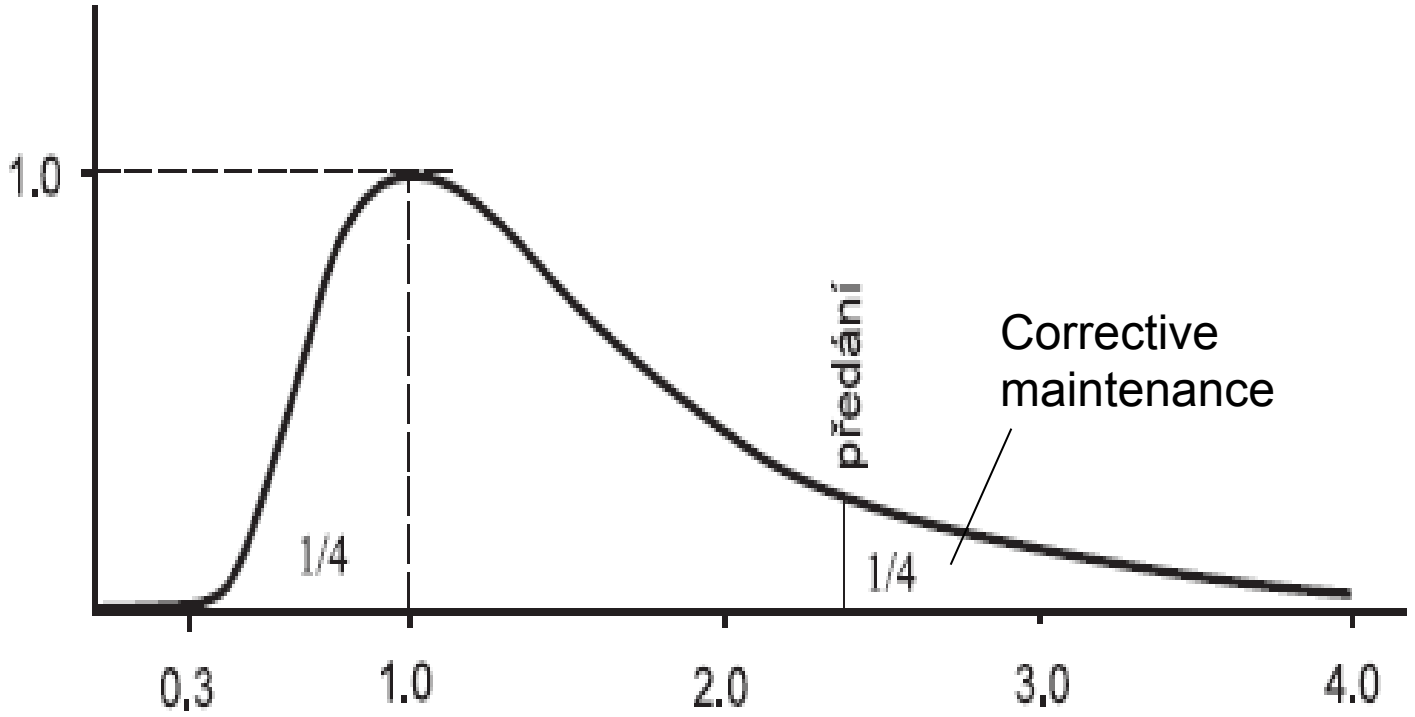
Obr. 15.12: Rayleightova křivka. Část plochy pod křivkou po předání jsou práce, které bud provedeny v rámci údržby (corrective maintenance). To, co se do okamžiku předá nestihne udělat, přechází do údržby, kde se projevuje jako neodstraněné chyby.

Pro t>3 je velikost týmu prakticky nula ⇒ 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

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

Jsou důvody pro komplikovanější model

Zobecnit na t = aT+d

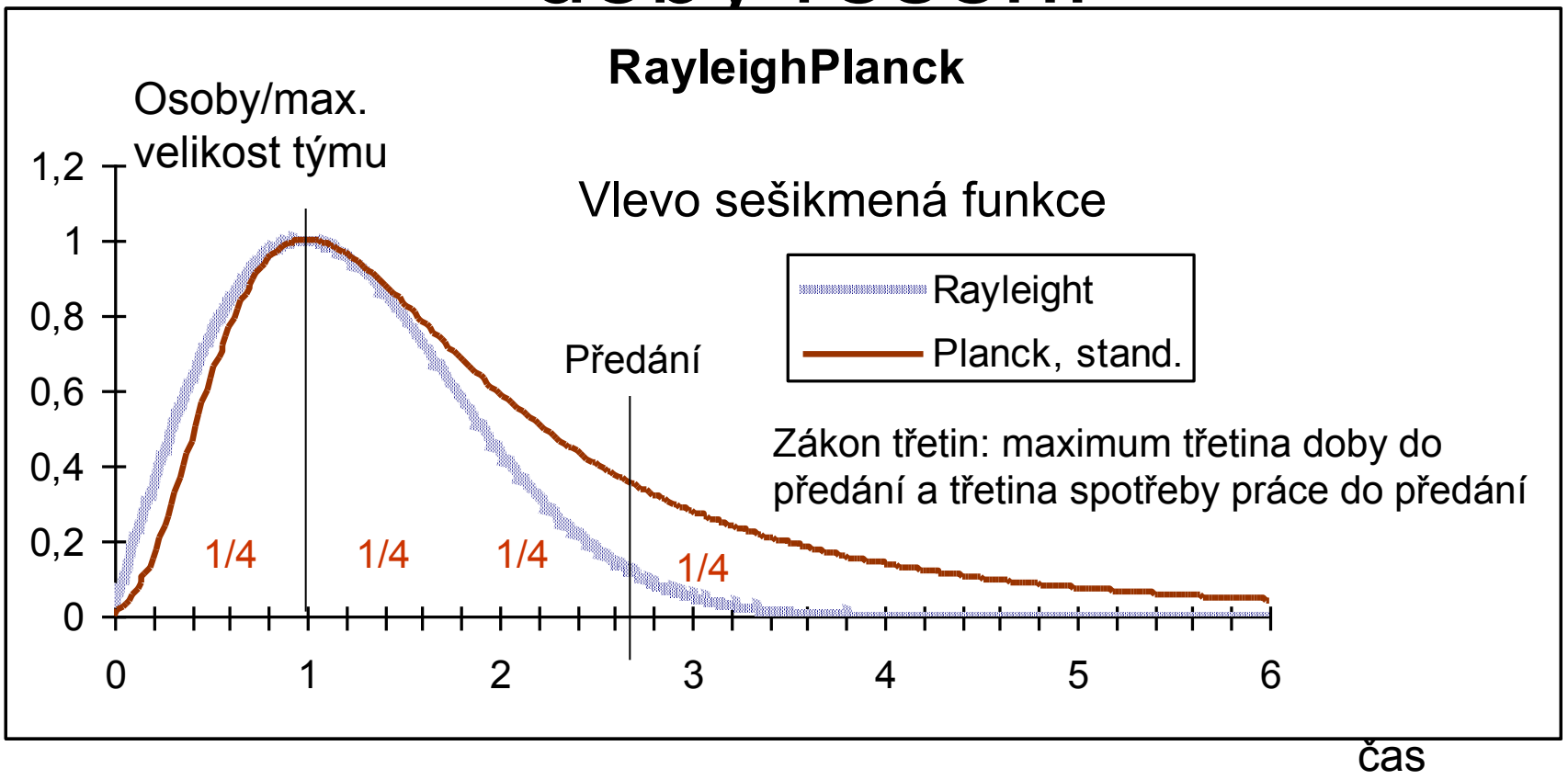


Obr. 15.13: Normalizovaný tvar Planckovy křivky $Planck(t) = 142.32 \cdot t^{-5}/(\exp(4.9651/t) - 1)$.

Mnoho práce se vykoná i pro t >5,

Posun vlevo lineární transformací nezávislé proměnné

Najímaný tým, vrchol a odhad doby řešení

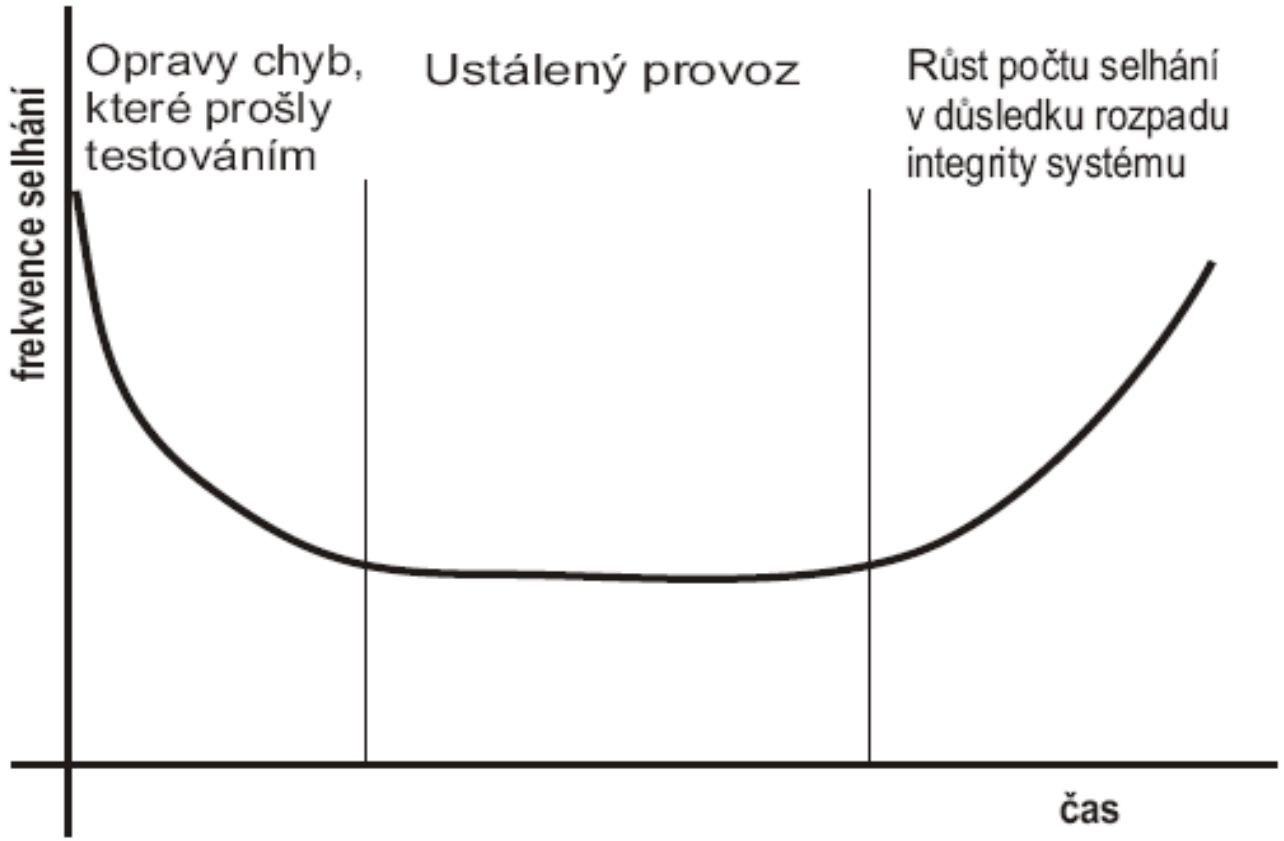


Pravidlo třetin. Do vrcholu 1/3 prací a 1/3 doby (za běžných okolností.Transformace proměnných tak, aby max bylo v bodě 1 a mělo hodnotu 1 a v nule byla hodnota funkce prakticky nula. U pevného týmu odpovídá intensitě práce

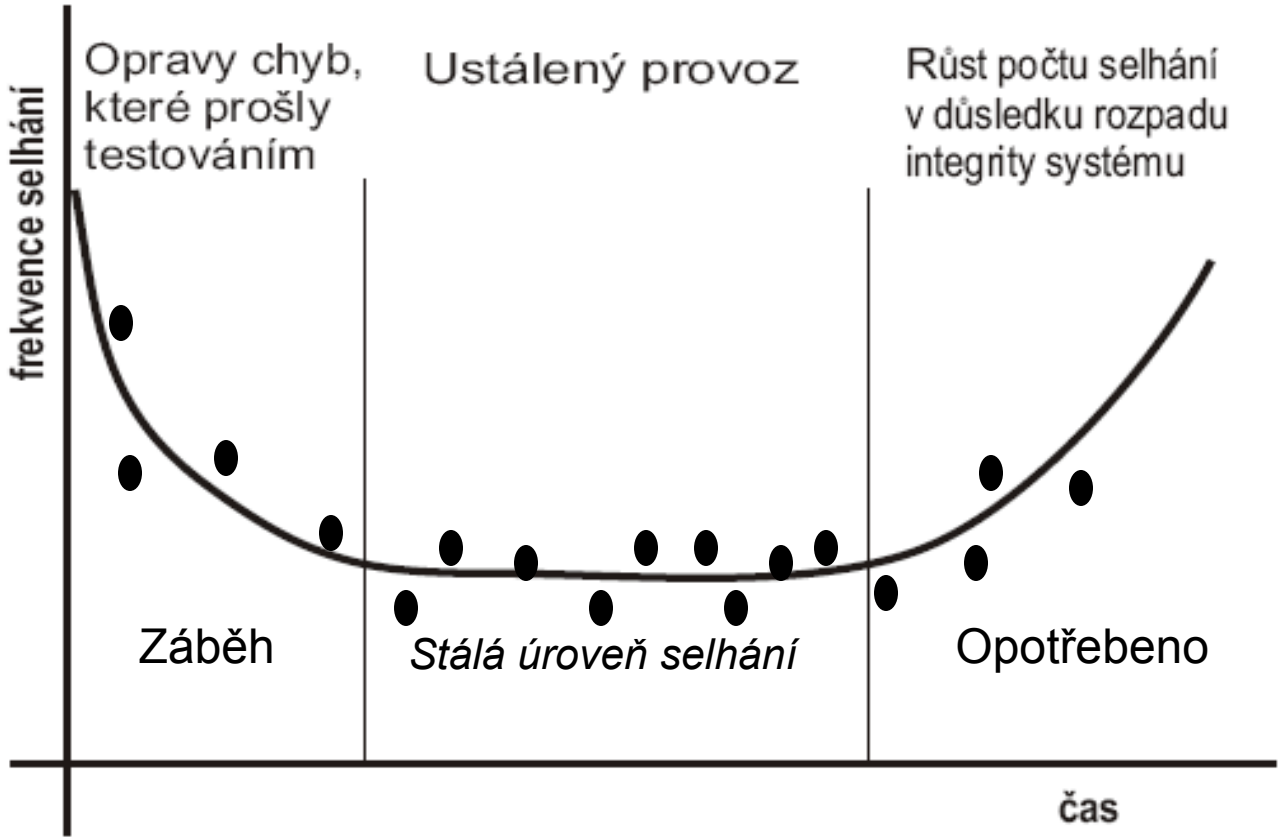
Etapy údržby

- Převzetí
- Etapa investic
  - Corrective maintenance, prvá vylepšení a přizpůsobení
- Etapa maximálního užitku
  - Vylepšení požadované uživateli
  - 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

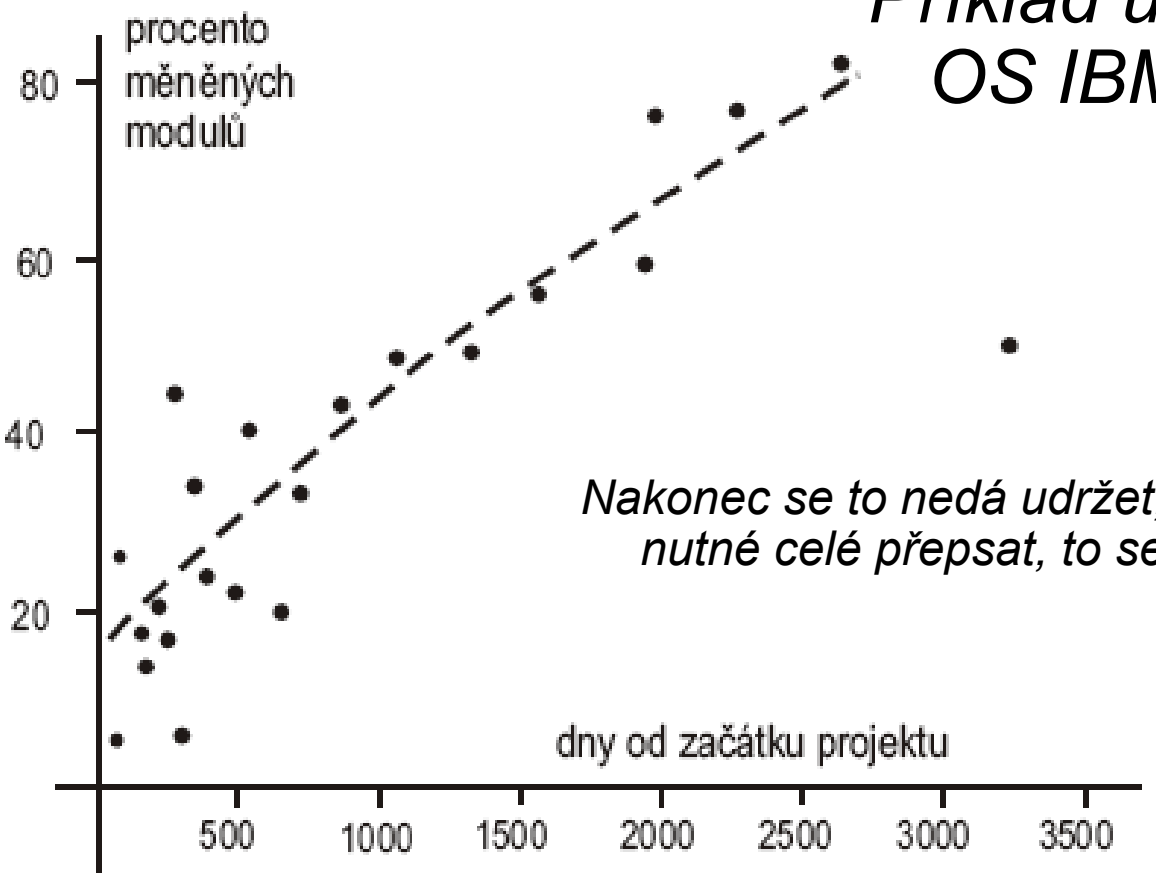


Dno vanové křivky implikuje **nenulovou** střední frekvenci selhání

- 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 pracnější.
  - Za odstraněný defekt přibude často nový, někdy i více než jeden
  - Není proto možné zero defect software

Složitost údržby roste s časem,
důvod stoupající větve vanové křivky

Příklad údržby OS IBM 360

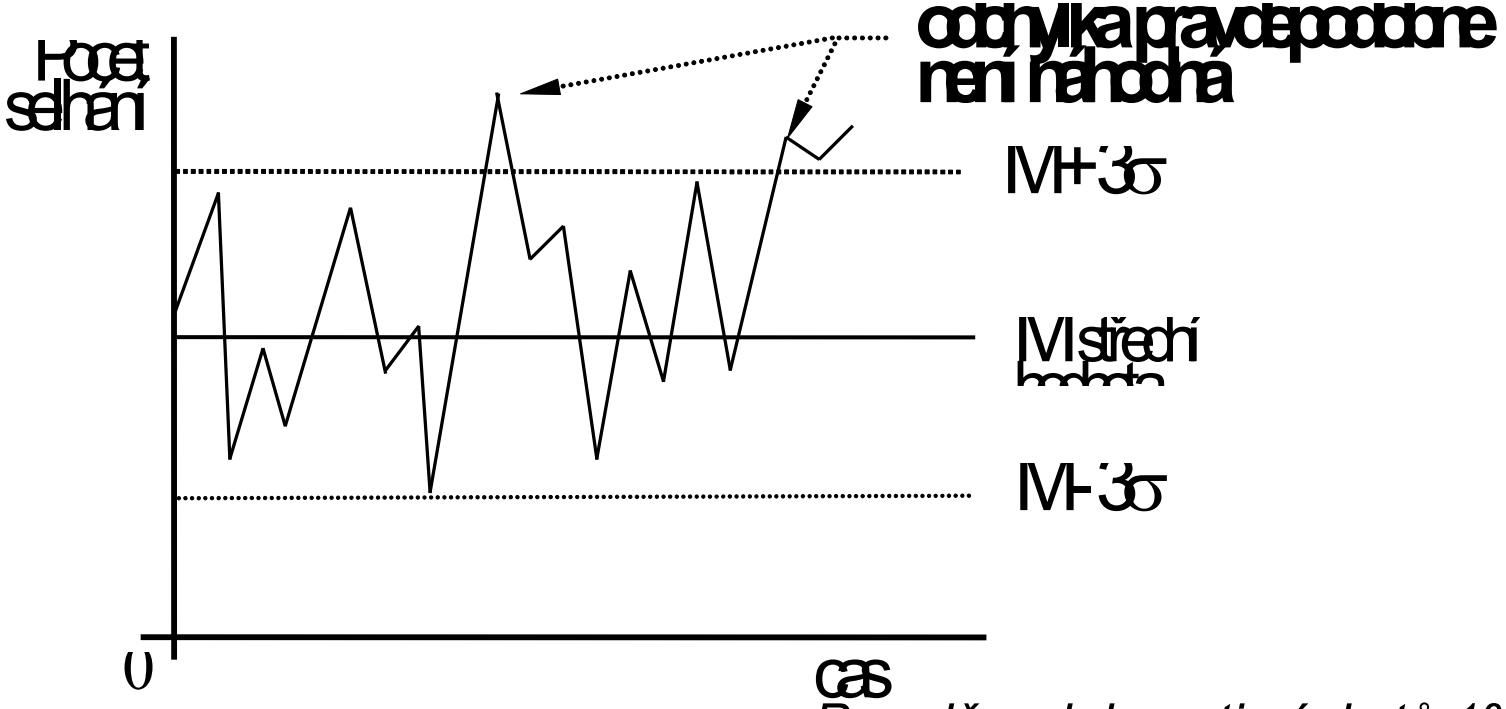


Nakonec se to nedá udržet, je to nutné celé přepsat, to se stalo

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



Pravděpodobnosti výskytů 1% (překročení vlevo), a 0.0001% (několikanásobně vpravo)

Co se tím vlastně prokazuje

- Je málo pravděpodobné ($p << 0{,}001$), 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)
- Převzetí existujícího (knihovny, produkty třetích stram existující SW
- Servisní architektura celku,
- Objektová orientace, komponenty
- Správné procesy vývoje (inkrementálnost, agilnost) a kvalita dokumentace, kvalita oponentur a testů
- Přenositelné technologie (Java)
- 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)
  - Používám moderní metodiky (OO, SOA, metanástroje jako XML a s ním spojené jazyky a DB)
  - Agilní metody vývoje
  - Dohodnuté standardy
    - Komentáře, volby identifikátorů, pravidla strukturování, oponovaní, vývoj testů, ….

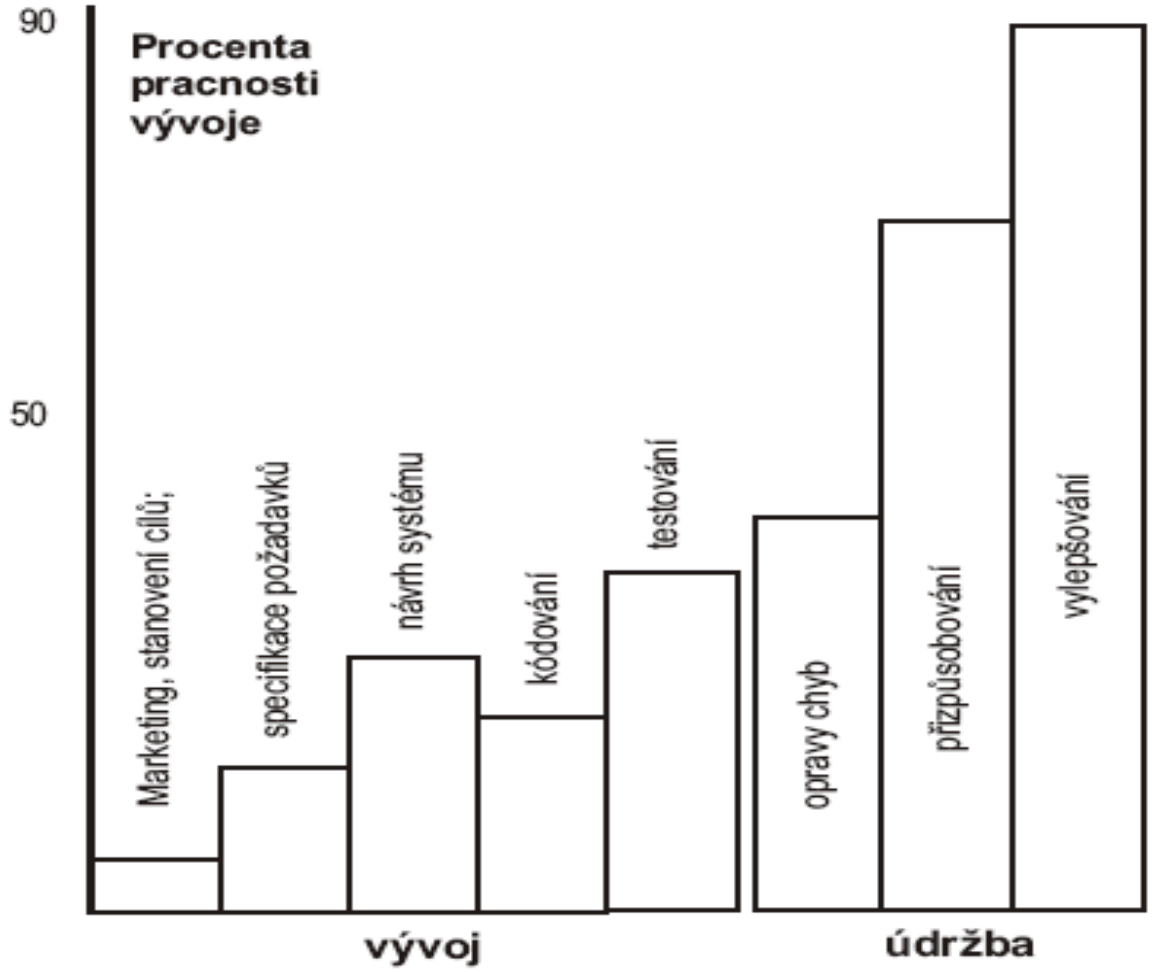
Co snižuje pracnost údržby

- Kvalita technik údržby
  - Automatizace opakování testů
  - DB reklamací a defektů
  - Sledování trendů defektů
  - 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
- 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

Jak se projeví SaaS, software as a service

A také cloud computing

Platform as a service

Zatím ve fázi líbánek

Vývoj uživatelského rozhraní

Návrh a vývoj uživatelského rozhraní je specifickou částí vývoje systému s vlastními problémy a metodami a standardy

Faktory akceptovatelnosti softwaru

1. Sociální a společenská akceptovatelnost.
   1. Potřebné
   2. Dá se s danými lidmi rozumně používat
2. Praktická akceptovatelnost.
   - 2.1 Užitečnost (Usefulness).
   - 2.1.1 Funkčnost.
     - 2.1.2 Použitelnost (Usability).
   - 2.3 Cena.
   - 2.4 Kompatibilita, přenositelnost.
   - 2.5 Modifikovatelnost.
   - 2.6 Spolehlivost.
   - 2.7 Dostupnost pro všechny uživatele.

Faktory použitelnosti softwaru

- snadnost naučení,
- efektivnost při používání,
- dobře se pamatuje, jak systém používat,
- málo chyb uživatele způsobených špatným ovládáním rozhraní,
- subjektivní příjemnost práce se systémem pro uživatele,
- dobré ergonomické vlastnosti

Efekty uživatelského rozhraní

- Ovlivňuje spokojenost zákazníka se systémem
- Je důležité i z čistě obchodního hlediska. Je „obalem“ softwaru
- Podstatně ovlivňuje složitost ovládání IS (až 10% efektu nepočítaje důsledky stresu).
- GUI je ergonomicky náročné
- Čtvrtina požadavků uživatelů se obvykle týká rozhraní

Použitelnost

Jak snadno se se systémem pracuje.

Jde o vlastnost uživatelského rozhraní, v SOA podstatně ovlivňuje SW inženýrské vlastnosti systému

Viz Nielsenovy knihy na téma Usability

Přesvědčit o významu rozhraní

- Přesvědčit management a někdy i koncového uživatele, že je to problém
- Získat prostředky (někteří doporučují i více než deset procent nákladů na vývoj)
- Zařídit, aby se vývoje a především testování účastnili budoucí (koncoví) uživatelé
- Vytvořit nástroje pro sledování rozhraní a jeho zlepšování během provozu
- Ve vývojovém týmu jsou žádoucí specialisté na UI
- Funkce systému mají umožňovat kvalitní rozhraní (exity)

Začátečníci a pokročilí

- *Noví uživatelé (dále začátečníci)* dávají přednost takovým nástrojům, které nevyžadují rozsáhlé zaučování předem a jsou výhodné pro zvládání systému metodou pokusů a omylů. Používají často nápovědu
- *Dlouhodobí uživatelé* budou mít tendenci používat klávesové zkratky a různá urychlení, které většinou znamenají i větší nároky na paměť.
  - Jak usnadnit přeškolení začátečníka pokročilého uživatele

Začátečníci a pokročilí

- Problém je v tom, že se používáním systému ze začátečníka stává pokročilý
- Není to ale úplně hladký proces, je na to třeba myslet při návrhu UI, který by měl upozorňovat na možnosti zrychlení
- Neobejde se to ale ani bez výcviku (seznamování těch, pro které to má větší význam, s možnostmi zrychlení)

Nápověda

- Kombinovat s tutoriálem, interaktivním manuálem a demo
- Interaktivní kontextová nápověda by měla být chápána jako pomoc v nouzi. GUI by mělo, pokud možno, být intuitivní, konsistentní a samovysvětlující

Začátečníci a pokročilí

- Zkušení uživatelé často preferují znakové rozhraní nejen proto, že s ním lze pracovat rychleji, ale také proto, že nemusí tolik pozorovat obrazovku a nemusí tolik používat myš. Klávesové rozhraní je ergonomicky výhodnější. Lze lépe kontrolovat, co se opravdu provádělo
- Je žádoucí obě techniky (ovládání myší a ovládání znaky) kombinovat pro dosažení optimálního výsledku.

Další výhody znakového rozhraní

- Snazší žurnál (log) pro kontrolu práce
  - Rychlejší vytváření „byznys inteligence“
- Snazší vytváření procedur
- Snazší vytváření výkonných příkazů (srv SQL a grafické rozhraní v Accesu)
- Hlavní bariéra – pro mnohé moc složité, nebezpečí příliš strmé křivky učení

Životní cyklus uživatelského rozhraní

- Analýza požadavků, specifikace požadavků.
- Návrh UI.
- Realizace prototypů a jejich testování s uživateli.
- Integrace nejlepších nápadů
  - iterativní vylepšování
- Integrace UI se zbytkem systému a testování UI společně s uživateli.
- Sledování vlastností UI za provozu a další vylepšování vlastností UI.

Životní cyklus uživatelského rozhraní

1. Při vývoji UI je ještě obtížnější než při návrhu funkcí odhadnout optimální vlastnosti UI, protože ty závisí na psychologii, dovednostech a znalostech budoucích uživatelů.
2. Testování UI je možné pouze tak, že systém testují uživatelé. Testování se provádí sledováním práce uživatelů.
3. Vlastnosti uživatelů se během používání systému vlivem zkušeností mění. UI musí vždy počítat se začátečníky i s pokročilými.
4. Důležitou roli hrají problémy ergonomie

Životní cyklus uživatelského rozhraní

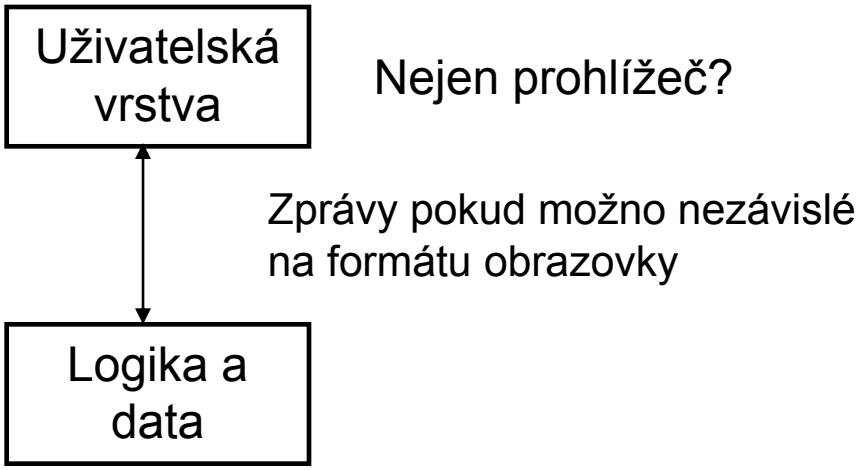
Při vývoji UI se porovnávají různá řešení a vybírají se nejlepší nápady

Obvykle se používá iterativní vývoj (námitky uživatelů se akceptují, hledá se vylepšování metrik a nová řešení se znovu testují)

Činnosti při specifikaci požadavků

- Poznání uživatele
- Analýza podobných či konkurenčních řešení
- Kontrolovatelné cíle
  - Termíny
  - Hodnoty některých metrik
- Odhad finančních efektů
  - 15% výkonu při práci u počítače – 3-5% platů

Uživatelská vrstva



Formát obrazovky závisí na zkušenostech uživatele, módách a také na tom, jak vystihneme psychologii práce. Musíme jej iterativně zlepšovat

Analýza a specifikace požadavků

- Poznání (koncového) uživatele, navázání spolupráce, problémy
  - Vedoucí koncového uživatele nevytvoří prostor
  - Vedoucí vývojářů se bojí, že vývojáři budou fungovat jako nápověda pro líné koncové uživatele
  - Co se hlavně sleduje
    - Kvalifikační předpoklady a zkušenosti (kolik je začátečníků a kdo ze spolupracovníků jim může pomoci)
    - Typy koncových uživatelů a jejich obeznámenost s IT
    - Popis úkolů a scénářů činnosti

Návrh rozhraní

- Paralelní návrh možných řešení, (heuristická evaluace, převzetí dobrých nápadů z existujících sytémů)
- Spoluúčast koncových uživatelů, ti co budou používat. Na začátku lze využít cizí lidi (hlavně u variant pro začátečníky)
- Koordinace a kontrola konsistence návrhu pro různé funkce

Heuristická evaluace

- Sledování, zda UI splňuje ověřené zásady (jsou v různých dotaznících, až 400 zásad). Základní požadavky na UI jsou:

Heuristická evaluace

Analyzovat existující systémy (např. Microsoft). Je vhodná podobnost s těmito systémy, připouští-li to funkčnost (srv. grafické systémy). Lidé se UI snáze naučí

Jednoduchý a přirozený dialog v jazyce uživatele,

- Jen to, co je v dané etapě nepostradatelné, podoba s mezilidským dialogem

Minimalizovat nároky na paměť (bezstavovost)

Heuristická evaluace

Konzistentnost (stejné texty, barvy obrázky)

Zpětná vazba (teploměry, zprávy o postupu práce)

Jasně vymezené exity (ukončení, návrat o několik kroků,)

Zkratky (horké klávesy), ikony i menu pro časté akce

Kvalitní chybové zprávy (srozumitelný název a odkaz na podrobnosti)

Vylučovat situace vedoucí k chybám

Různé varianty nápovědy

Heuristická evaluace – výtvarné aspekty

Na obrazovce nemá být mnoho logicky odlišných údajů (do 12), logicky stejné vstupy se počítají jednou, př. DODACÍ LIST.

Nejdůležitější vlevo nahoře, střed

Nehýřit barvami a tvary, zatěžuje zrak a unavuje to mozek

Různé displeje nemají stejné podání barev a to může významně zhoršit čitelnost (nedávat spektrálně blízké barvy do popředí a do pozadí)

Vliv výtvarného řešení (je to hezké)

Heuristická evaluace – výtvarné aspekty

Teplé barvy vpředu, studené v pozadí

Při zdobnosti snáze uděláme chybný návrh barev

V provozu je důležité, jak je to pracné a jak mne to chrání před chybami

Zdobnost je vítána je-li svým způsobem nápovědou a snižuje-li námahu

Zpětná vazba

- Reakce systému se pociťuje jako okamžitá, je-li do desetiny sekundy. U psaní musí být ještě kratší
- Reakce okolo sekundy je tolerována, jde-li o složitější akci
- Reakce přes více sekund vyžaduje indikaci průběhu a nemá být příliš často

Testování rozhraní

- Realizace prototypů
- Předvedení a zkoušení několika uživateli
  - Někdy se vyplatí prvý test dělat s „brigádníky“ To je zvláště vhodné pro ověřování variant pro začátečníky
  - Ne vždy lze, např. je-li nutná věcná znalost toho, co systém podporuje
- Log průběhu

Zahájení testu

Příprava nástrojů a lidí, místo, nástroje

Test provádí obvykle člen skupiny uživatelů za přítomnosti vývojáře. Je třeba brát ohled na velké individuální rozdíly mezi jednotlivými uživateli a také mezi nezkušenými a zkušenými pracovníky. Ověřování UI je proto třeba provádět s větší skupinou pečlivě vybraných budoucích uživatelů. Bývá výhodné provést nejprve zaškolení v ovládání systémových prostředků. Optimální počet testujících je 3 až 6.

Zahájení testu

Testéři pracují s tou částí UI, se kterou budou skutečně pracovat při provozu systému.

Při organizaci testů je žádoucí navodit atmosféru spolupráce:”Pracujete na tom, aby vám to, co budete používat, sloužilo dobře“.

Uživatelé nemají pracovat ve stresu.

Výsledky testů by měly být anonymní.

Upozornit, že se systém na základě požadavků testérů může změnit

Zahájení testu

Testování lze do jisté míry provádět na částečně funkčních prototypech. Vždy je však nutné nakonec provádět testy i na oživeném systému. Důvodem je potřeba testovat doby odezvy.

Je obvyklé, že se na základě analýzy výsledků testů UI podstatně mění. Testovat je nutné i nové verze UI.

Body instruktáže při zahájení testu

účelem testů je testovat systém, nikoliv uživatele;

účelem testů je dosáhnout spokojenosti uživatelů;

je žádoucí, aby se všichni svobodně vyjadřovali;

poněvadž se jedná o testování, může výsledný systém fungovat poněkud jinak;

poprosit, aby o testu testující nehovořili, aby tím ostatní testéry neovlivňovali;

Body instruktáže při zahájení testu

zdůraznit, že účast na testu je dobrovolná a lze jej kdykoliv ukončit a že výsledky testu jsou anonymní a důvěrné;

uvést, že jsou možné otázky, že je vítáno vyjádření pochybností, a že se má systém v budoucnu používat bez cizí pomoci;

požádat o ”myšlení nahlas“, tj. poprosit, aby testující nahlas komentoval, co dělá a proč;

vítány jsou pochvaly i kritické poznámky

poprosit o pokud možno rychlou práci bez chyb.

Provedení testu

- Je důležité, aby testování nebylo přerušováno telefony, návštěvami atd.
- Je výhodné, když uživatel provede na počátku rychle a úspěšně nějakou část dialogu, je pak méně nervózní.
- Atmosféra při testování by měla být uvolněná.
- Nedávat najevo, že uživatel je pomalý nebo že dělá chyby.
- Vyloučit kibice, včetně (to především)šéfů testujícího.
- Zastavit testování, vznikne-li pocit, že testér toho má již dost.

Vyhodnocení testu

- Požádat uživatele o celkový názor, především o to, jak je spokojen.
- Výsledky testů zaznamenat v dohodnuté formě.
- Provést analýzu žurnálu (ten je dodré sledovat i za (zkušebního) provozu
- Výsledky zavést do databáze projektu (pokud existuje).
- Sestavit vlastní hodnocení

Vyhodnocení testu

- Hodnocení problémů
  - 0 – celkem OK
  - 1 – kosmetická - opravit, bude-li čas
  - 2 - méně závažná – opravit,pokud nebouu závažnější
  - 3 – závažná – opravit co nejdříve, ale není překážkou pro zahájení provozu
  - 4 - katastrofická – systéím nelze provozovat

Metriky pro hodnocení UI

Doba provedení určitého úkolu.

Počet variant úkolů úspěšně provedených za určitou dobu.

Poměr úspěšně provedených úkolů k počtu chyb.

Doba potřebná pro nápravu chyby.

Počet příkazů použitých během testů.

Počet příkazů, které nebyly vůbec použity.

Metriky pro hodnocení UI

Frekvence použití nápověd a manuálů.

Počty kritických a pochvalných komentářů uživatele.

Počet případů, kdy byl uživatel frustrován a kdy potěšen.

Rozsah mrtvých dob“, během nichž uživatel nekomunikuje.

Počet případů, kdy uživatel nevěděl jak dál.

Sledování za provozu

- Log průběhu a jeho analýza
- Dotazníky, sledování reakcí
- Úpravy a jejich efekty

Metoda	Etapa	Počet testérů	Hlavní výhody	Hlavní nevýhody
Heuristické evaluace	Návrh	0	Detekuje jednotlivé problémy použitelnosti, lze využít experty.	Není kontakt s uživateli.
Měření výkonů.	Testování UI, analýza podobných a konkurenčních produktů.	alespoň 10	Přesná data. Snadná kritéria porovnání.	Obvykle se nedaří porovnat všechny aspekty.
Myšlení nahlas.	Návrh, informativní evaluace.	3 až 5.	Levné. Zachycuje chyby v pochopení systému v poměrně reálné situaci.	Nepřirozené pro testujícího. Nevhodné pro zkušené. Ti rychleji jednají než mluví.
Pozorování.	Analýza a evaluace UI, tvorba scénářů.	alespoň 3 pro každý subsystém	Sleduje skutečné činnosti v reálu. Inspiruje návrh funkcí.	Subjektivní, obtížně kontrolovatelné, časově náročné, často se nenajde dost času na provedení.
Dotazníky	Analýza, sledování problémů za provozu.	Většina uživatelů	Anonymní. Lze opakovat, zachytí názory více uživatelů.	Nebezpečí, že důležití uživatelé neodpoví nebo dotazník přesně nepochopí.
Interview.	Analýza úkolů.	Několik	Flexibilní, lze jít do hloubky.	Časově náročné, náročné na kvality moderátora, subjektivní.
Žurnál (log)	Závěr testování, provoz.	Všichni uživatelé	Běží stále. Výhodné pro vyhodnocování efektivnosti a trendů.	Je nutno realizovat jednoduchý IS pro vyhodnocování.
Sledování názorů uživatelů.	Provoz.	Co nejvíce.	Lze sledovat trendy v názorech a potřebách uživatelů.	Obtížně se zajišťuje. „Proč se o to starat, když už to pracuje.“

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 přepoklad 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 řídící zásahy během údržby,procesy zajišťujíící 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

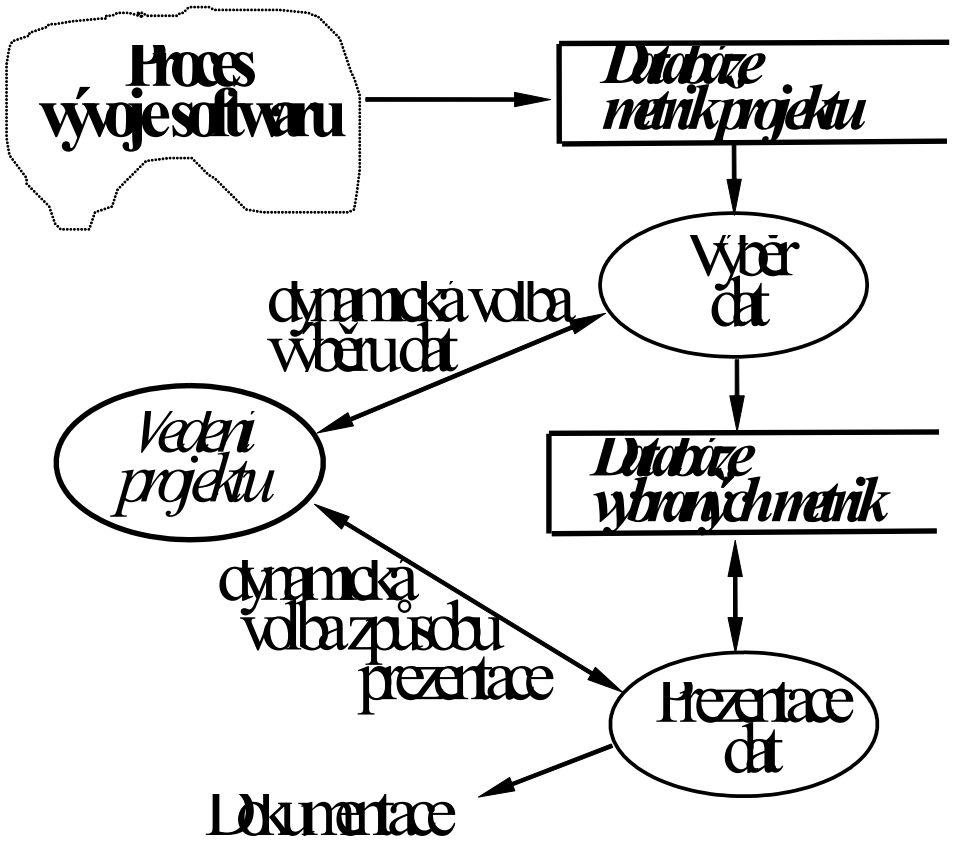
Výsledkem výzkumu SW metrik jsou metodiky vývoje (SOA, OO, strukturované programování, znalosti a vlivu kvality specifikací atd.) a metodiky odhadu pracnosti a doby řešení COCOMO, funkční body, a filosofií SOA ITIL

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

Kategorie	**Dimenze**
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 řízeni

- 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 řízeni

- 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,
    - paradigmata

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 norme - 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 vloženosti 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 *e*1 a zjištěných v etapě *e*2 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).

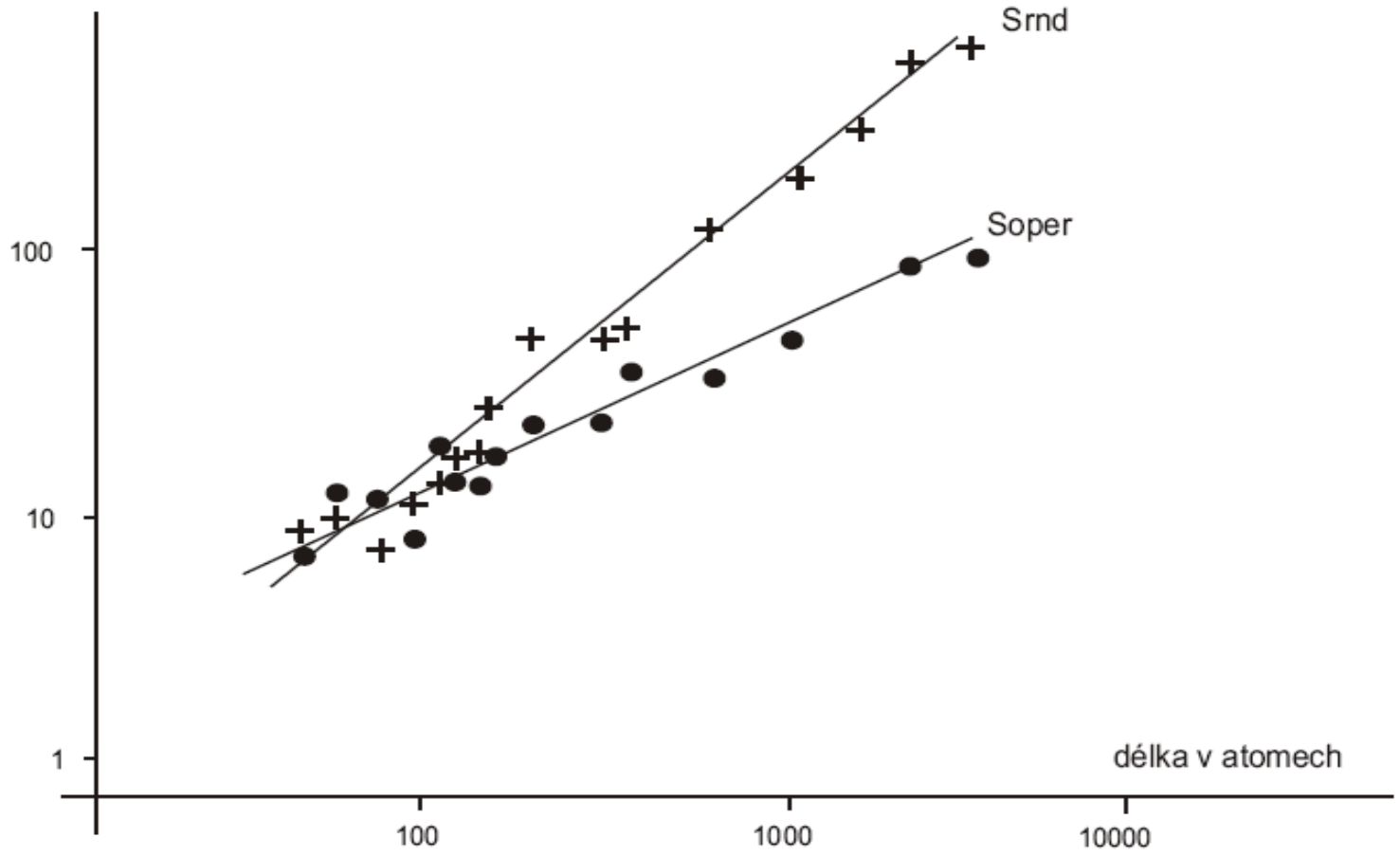
DobaOpr – průměrná doba opravy selhání.

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

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

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

$$Soper \cong C\text{“ } Del^{b}$$

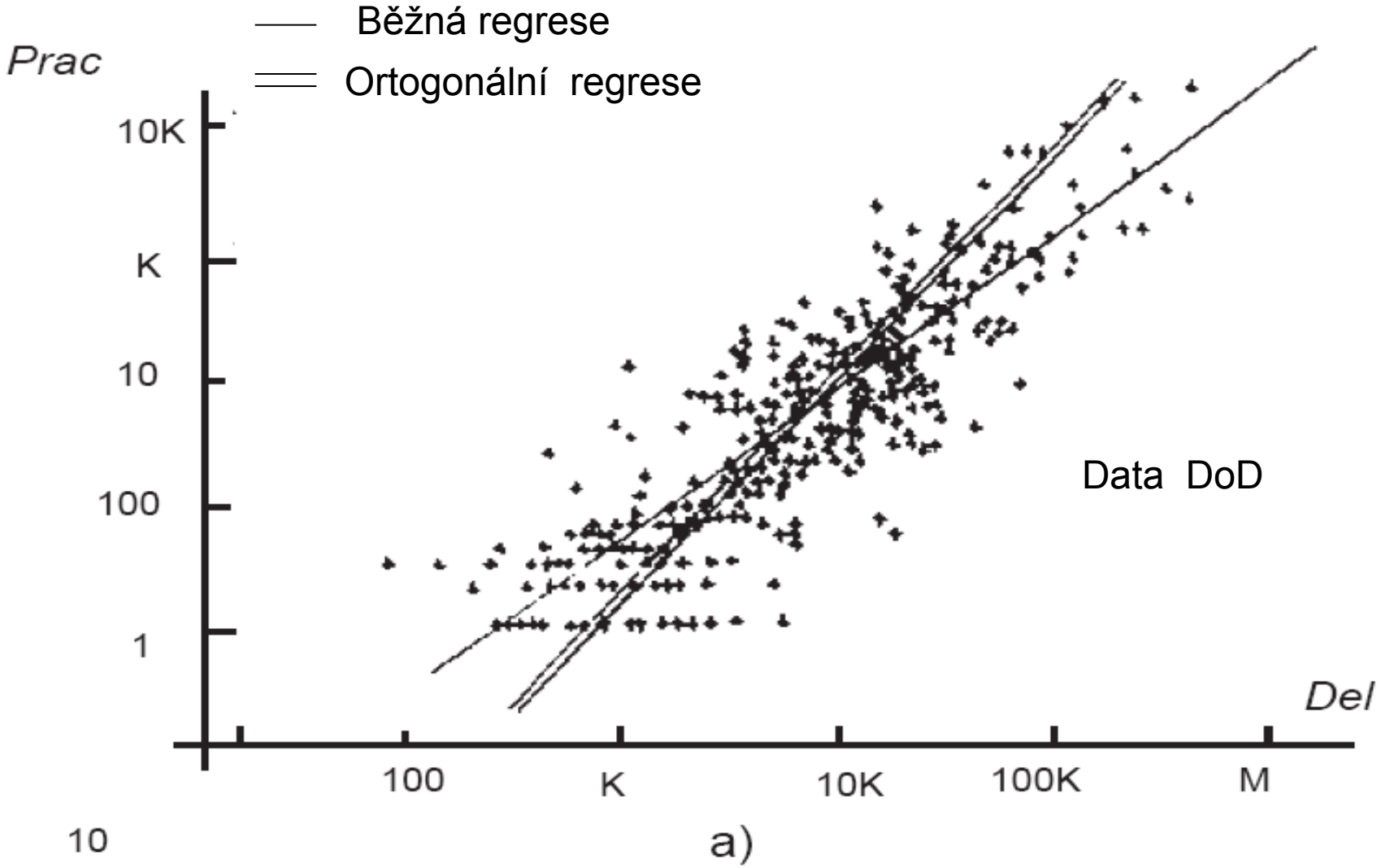
dostaneme

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

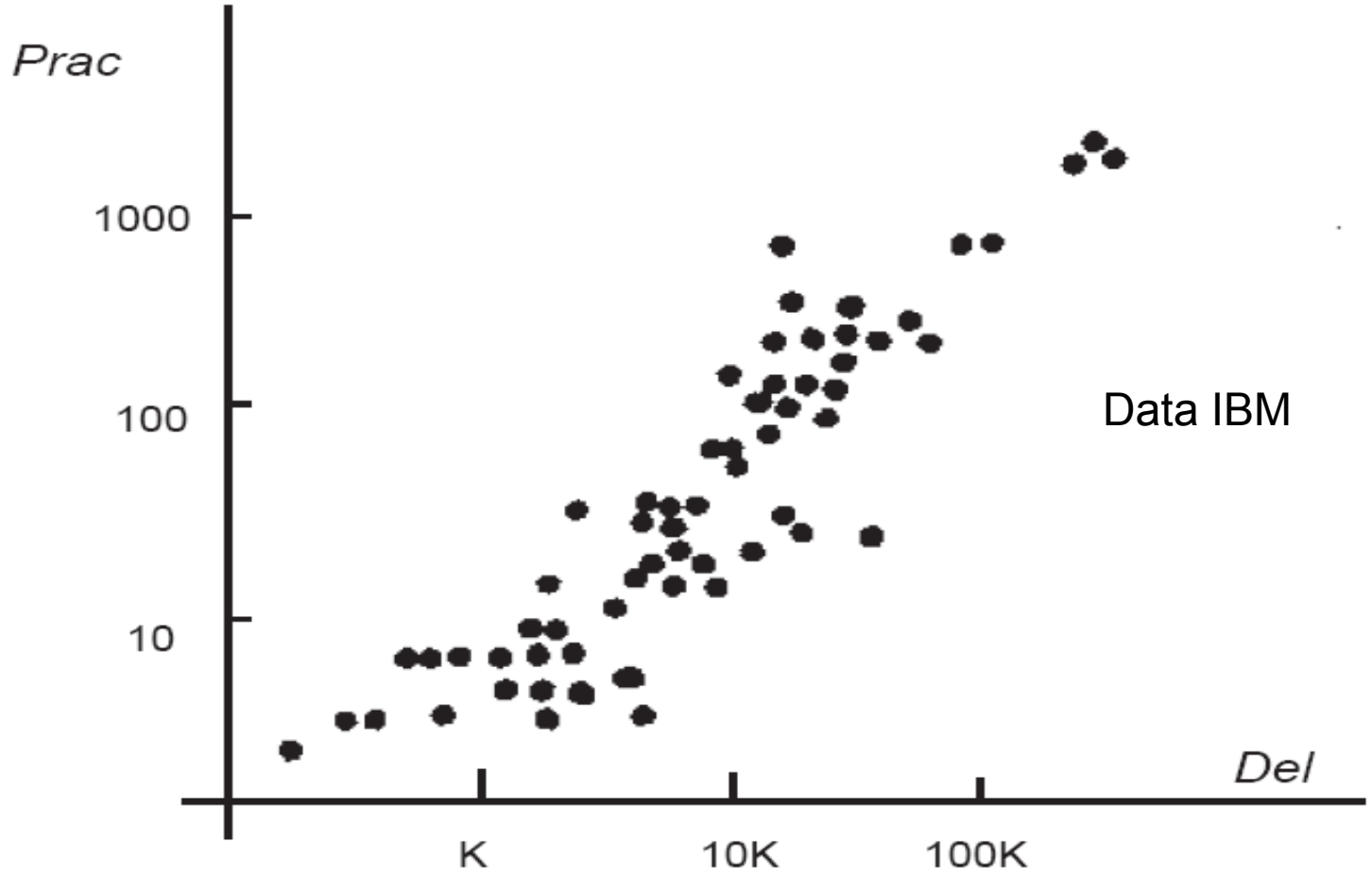
Pro velké programy je hodnota b příliš vysoká. Je to důsledek toho že vztah modifikujtakové technologie, jako dekompozice a znovupoužití částí.

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

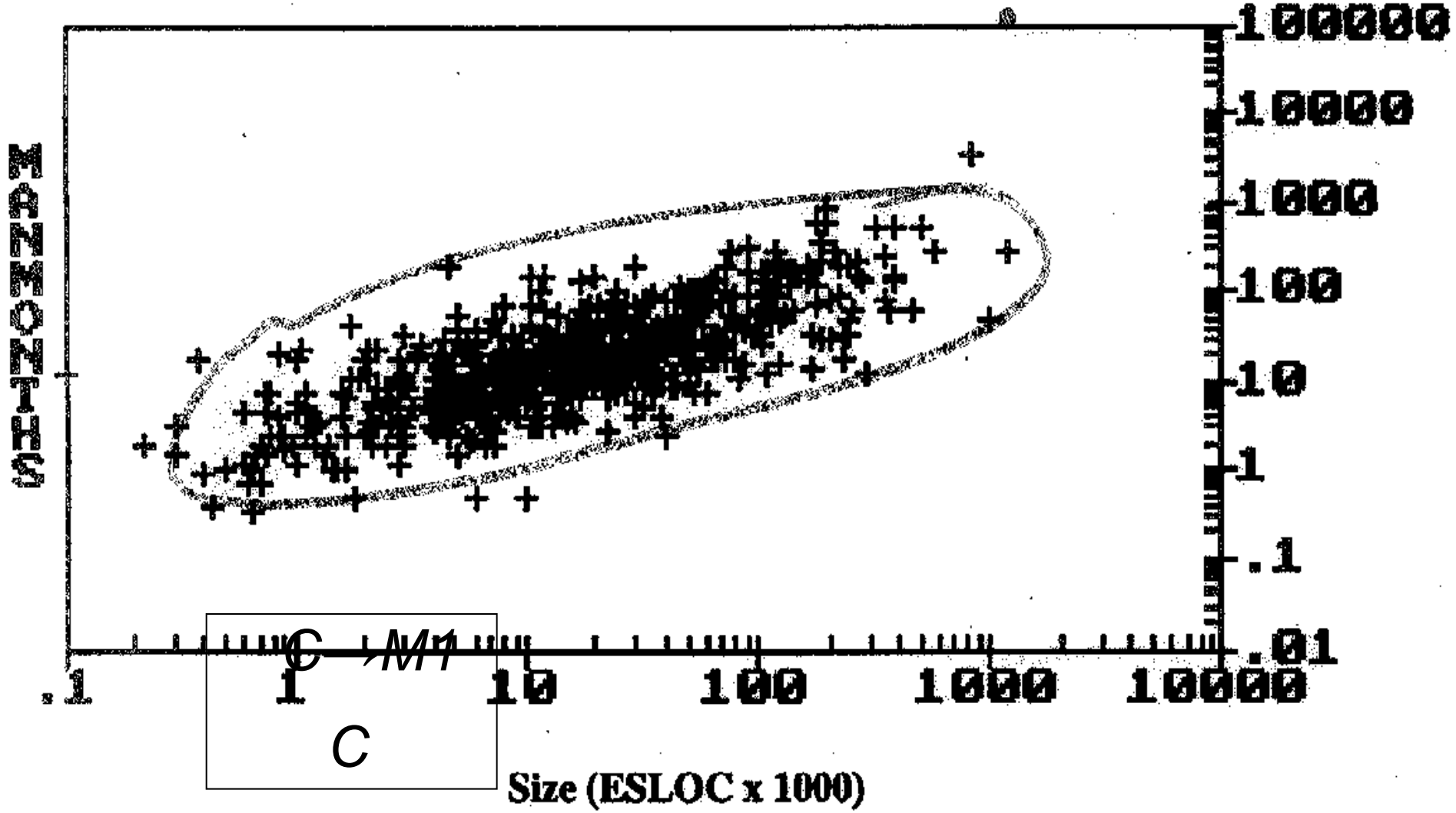
Závislost pracnosti na délce programu



Závislost pracnosti na délce programu

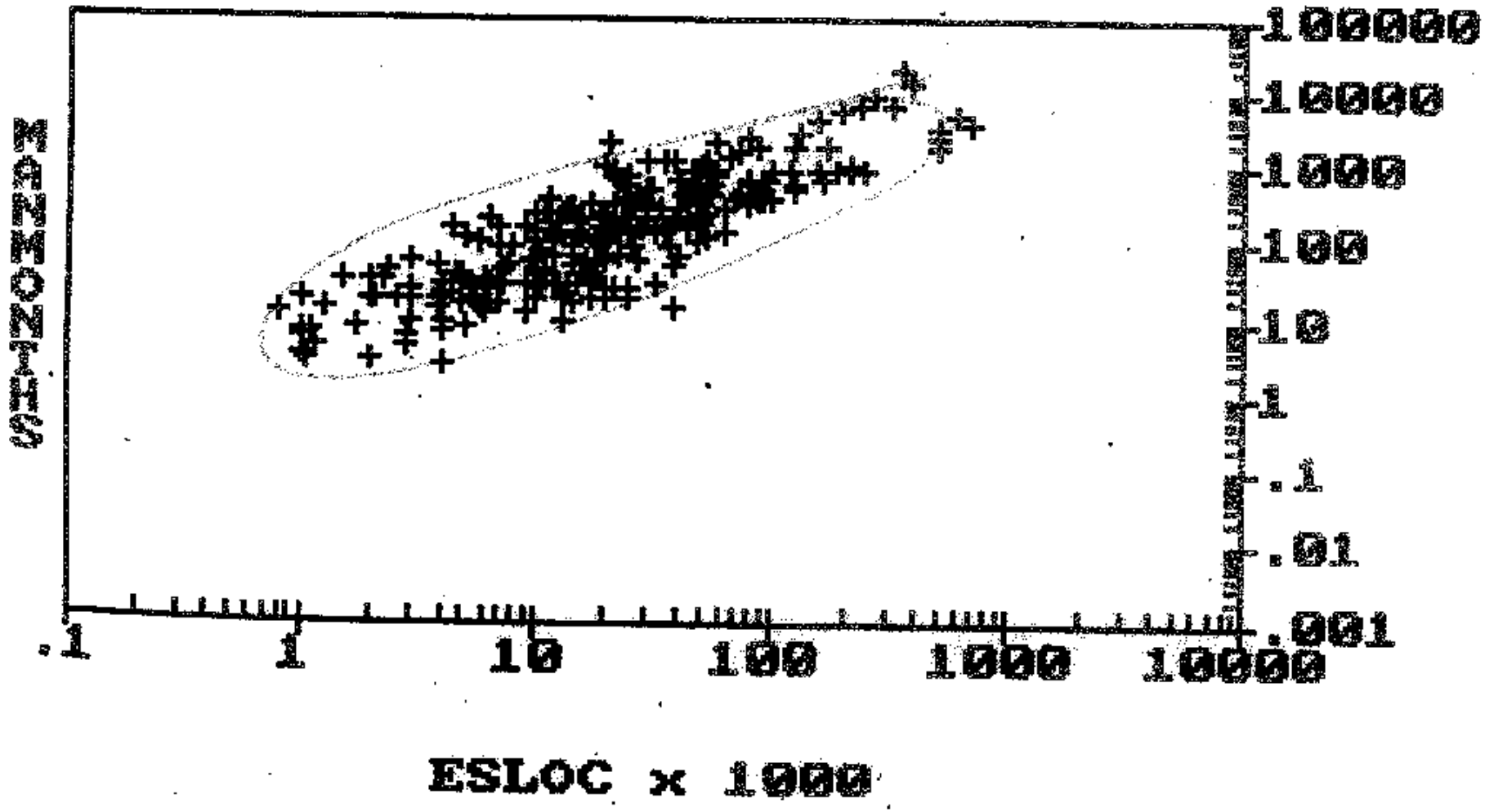


Business Systems

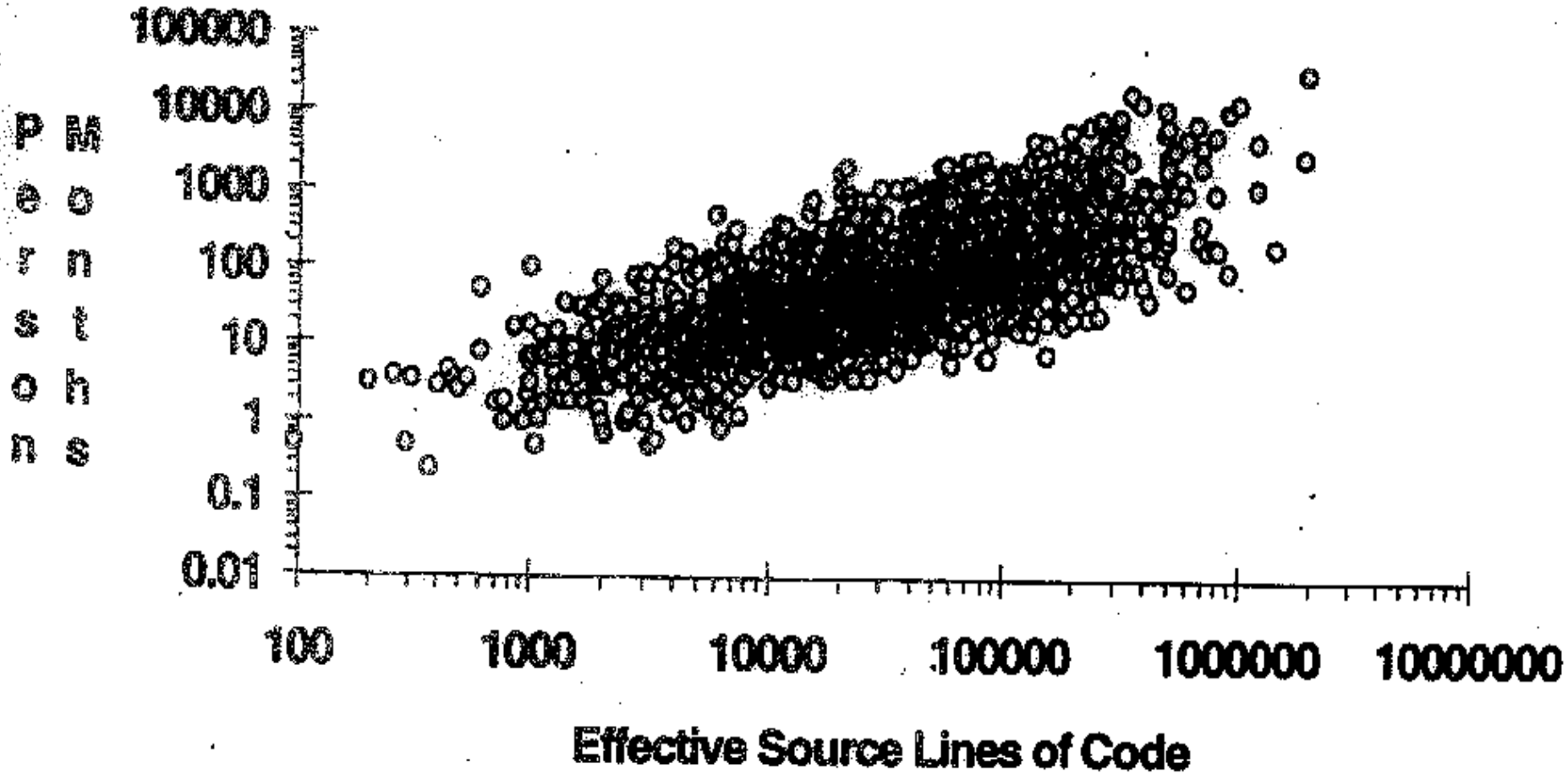


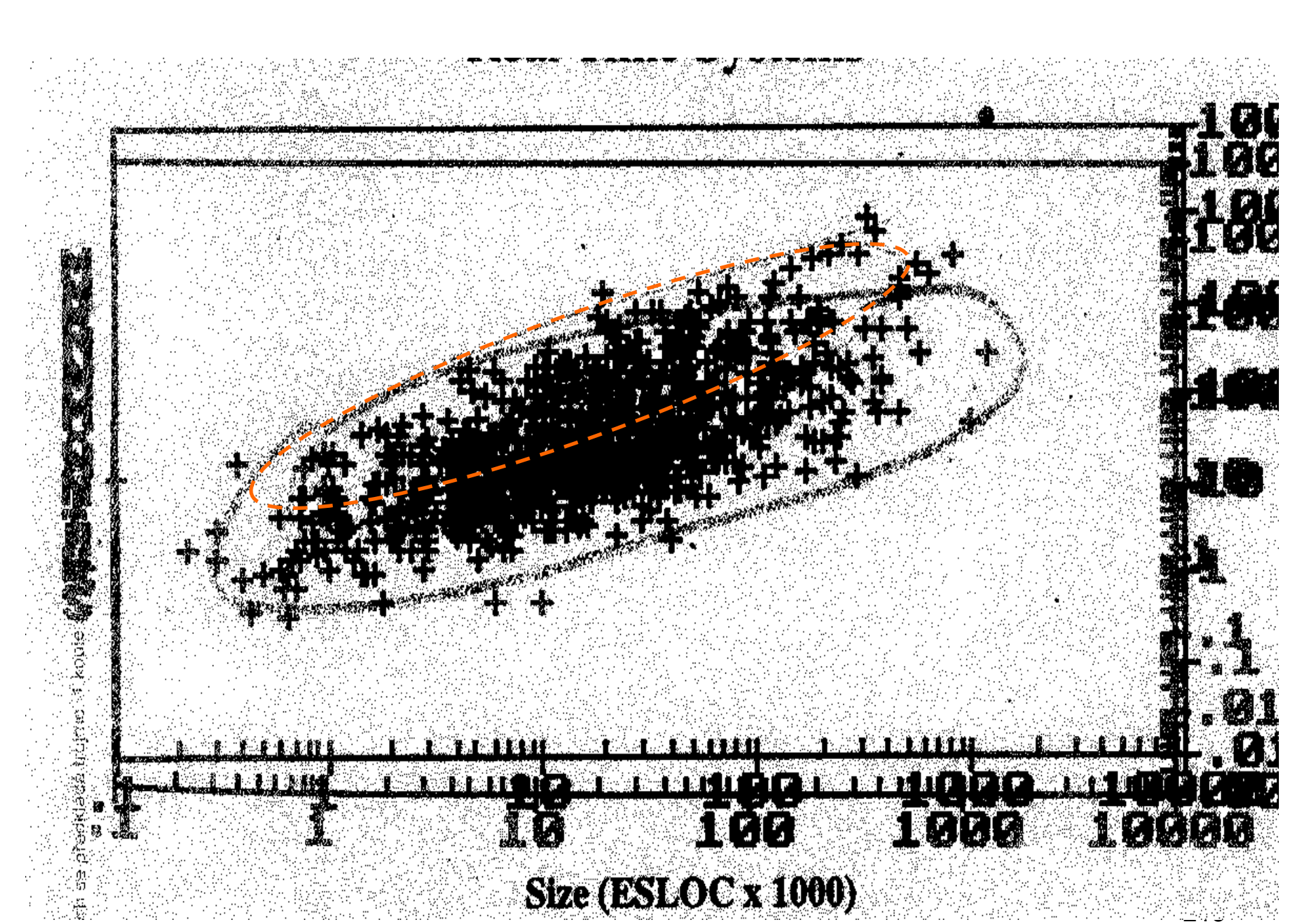
Effort vs. Size

Real Time Systems



Effort Behavior




1
10
100
1000
10000
Size (ESLOC x 1000)

Pozor na statistiku

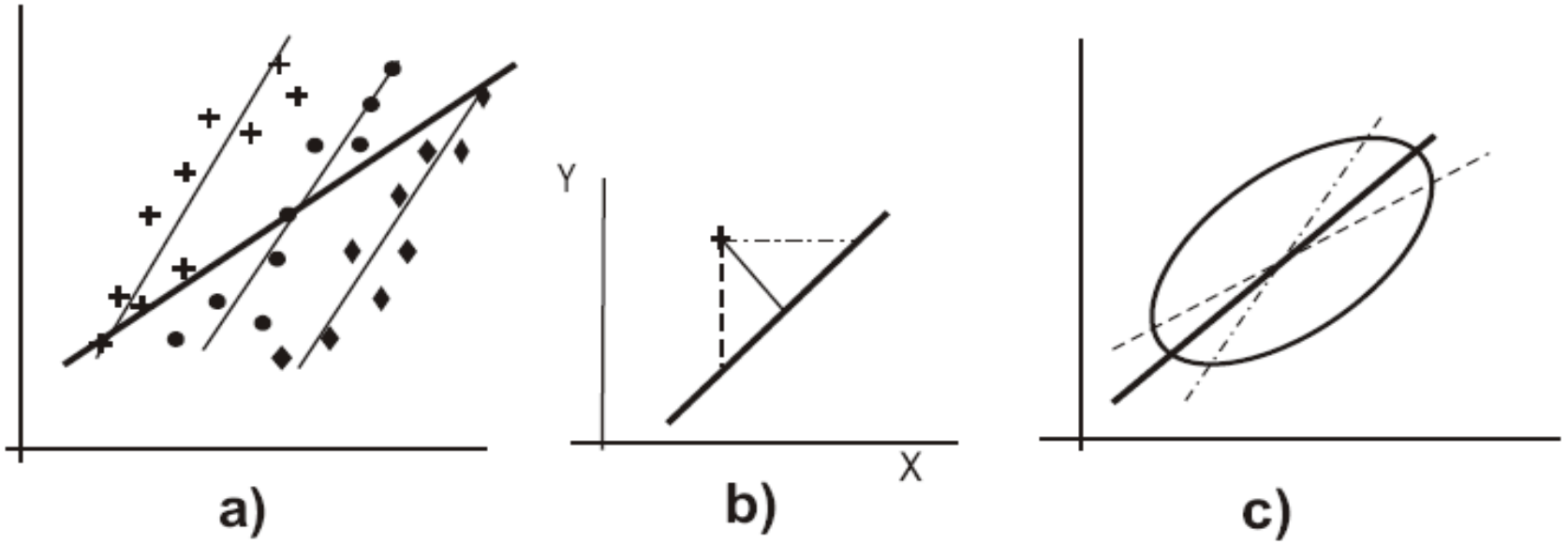
- Analýzou dat klasickou regresí dostaneme

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

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 regresi.

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

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

Ortogonální regrese dává

$$a \cong 1/8$$

Prvá zákonitost

Pro délku a pracnost platí vztah

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

takže

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

Ortogonální 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.	Putnamn, 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 regresní analýzy pro různé soubory dat. Data zřádků 9 až 1 se týkají malých podprogramů.

Putnamova rovnice

Podle třetí řádky tabulky

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

čili

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

Z definice $Del = Prac\ Doba,$ takže

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

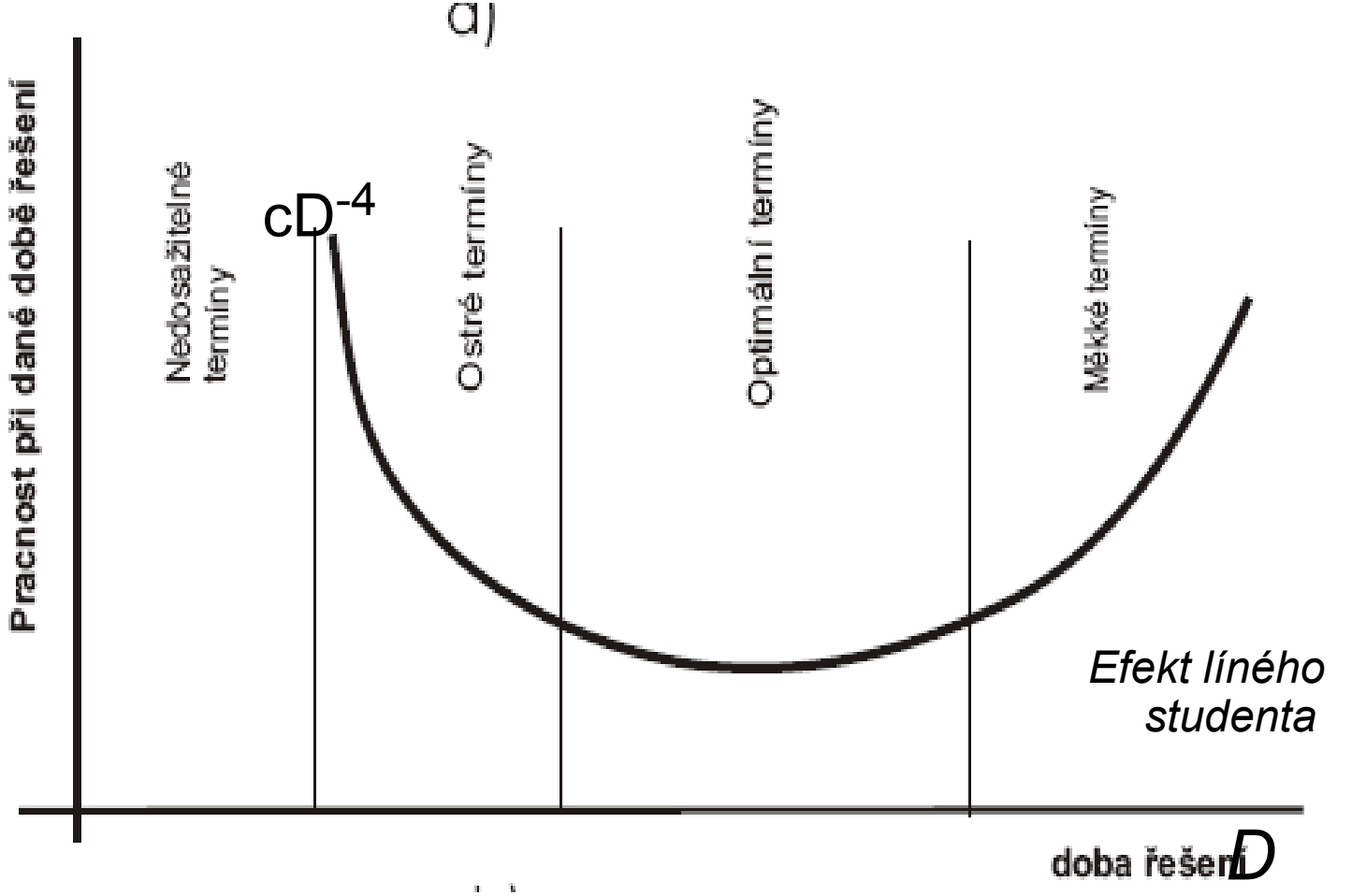
To je Putnamova rovnice

Vliv napjatých termínů

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

$$Doba < ¾\ Prac^{1/3}$$

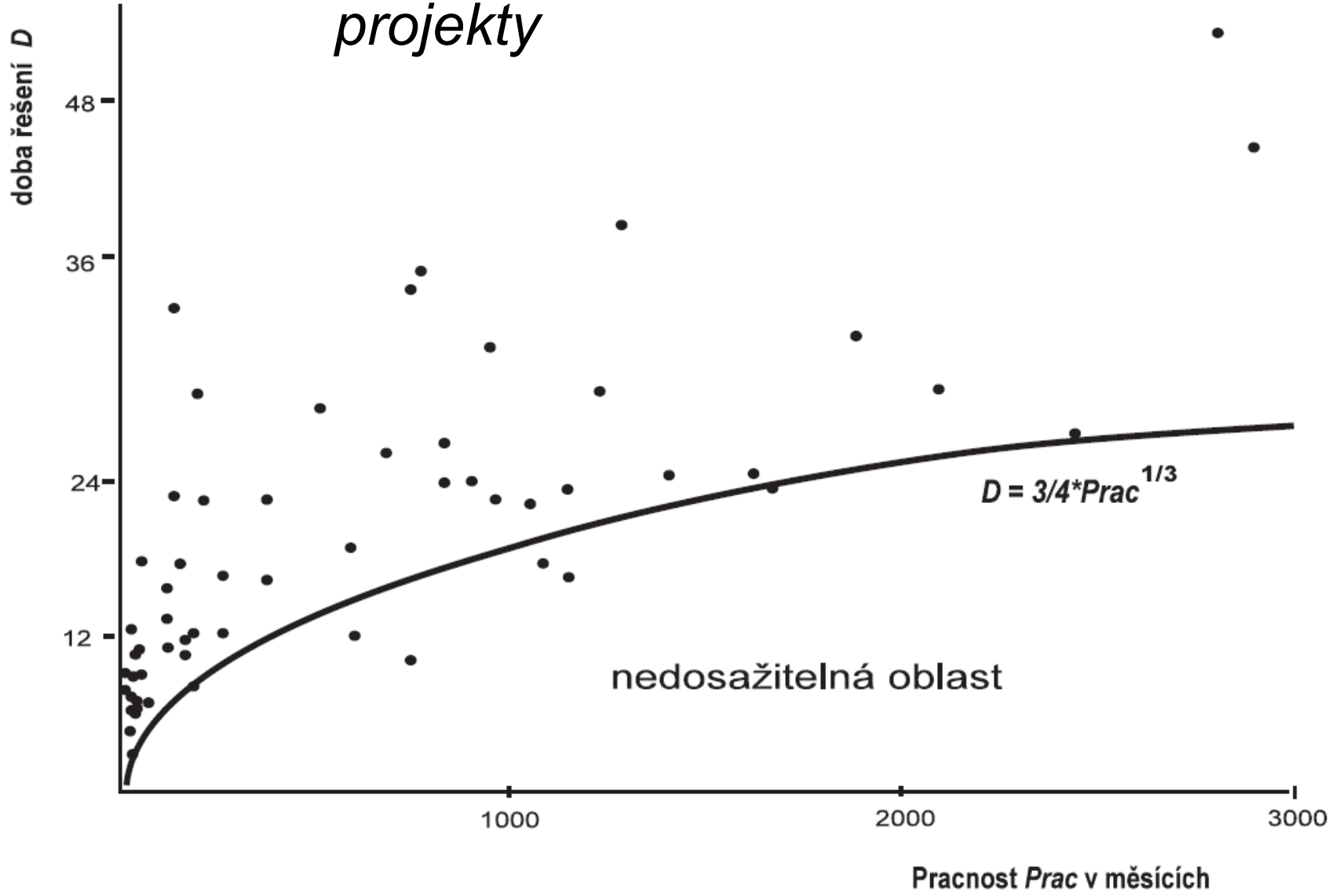
Pracnost u nedosažitelné oblasti



m

–D

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

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

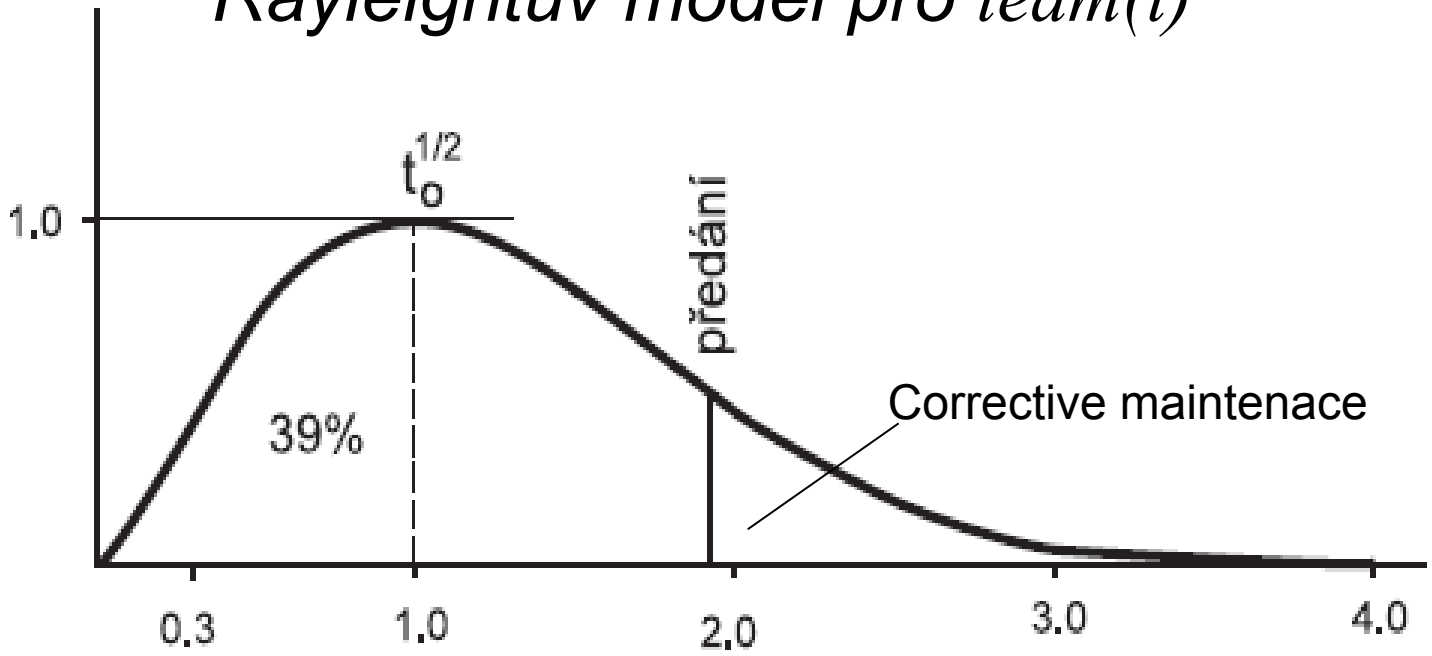
Použití ve function points

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.

$$team(t) \mathrel{\widehat{=}} K \cdot t / t_0 \cdot \exp(-\frac{t^2}{2t_0})$$

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-Botzmannův zákon
- Má málo parametrů a změna parametrů nemění příliš tvar
- Proto si půjčíme Planckův zákon

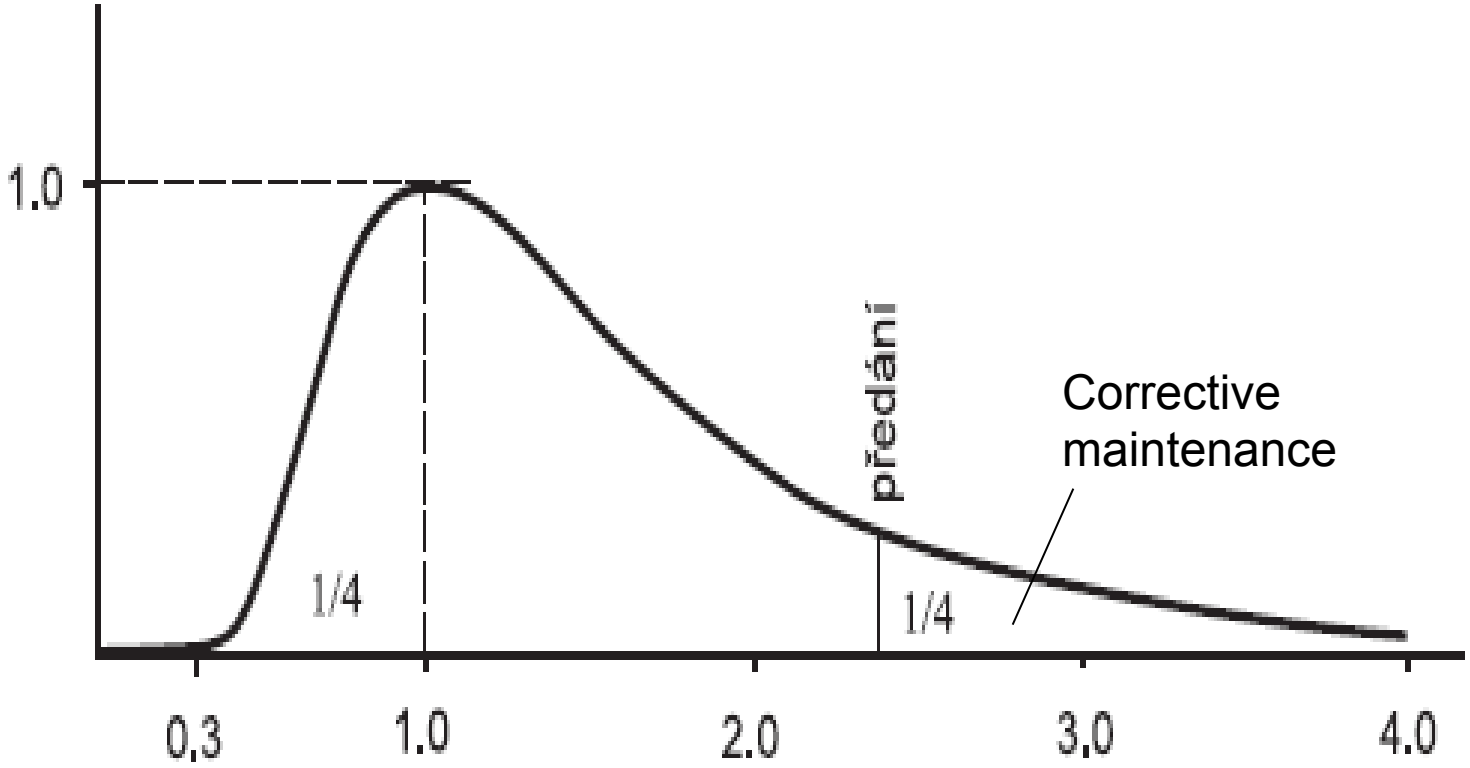
Planckův model

- 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

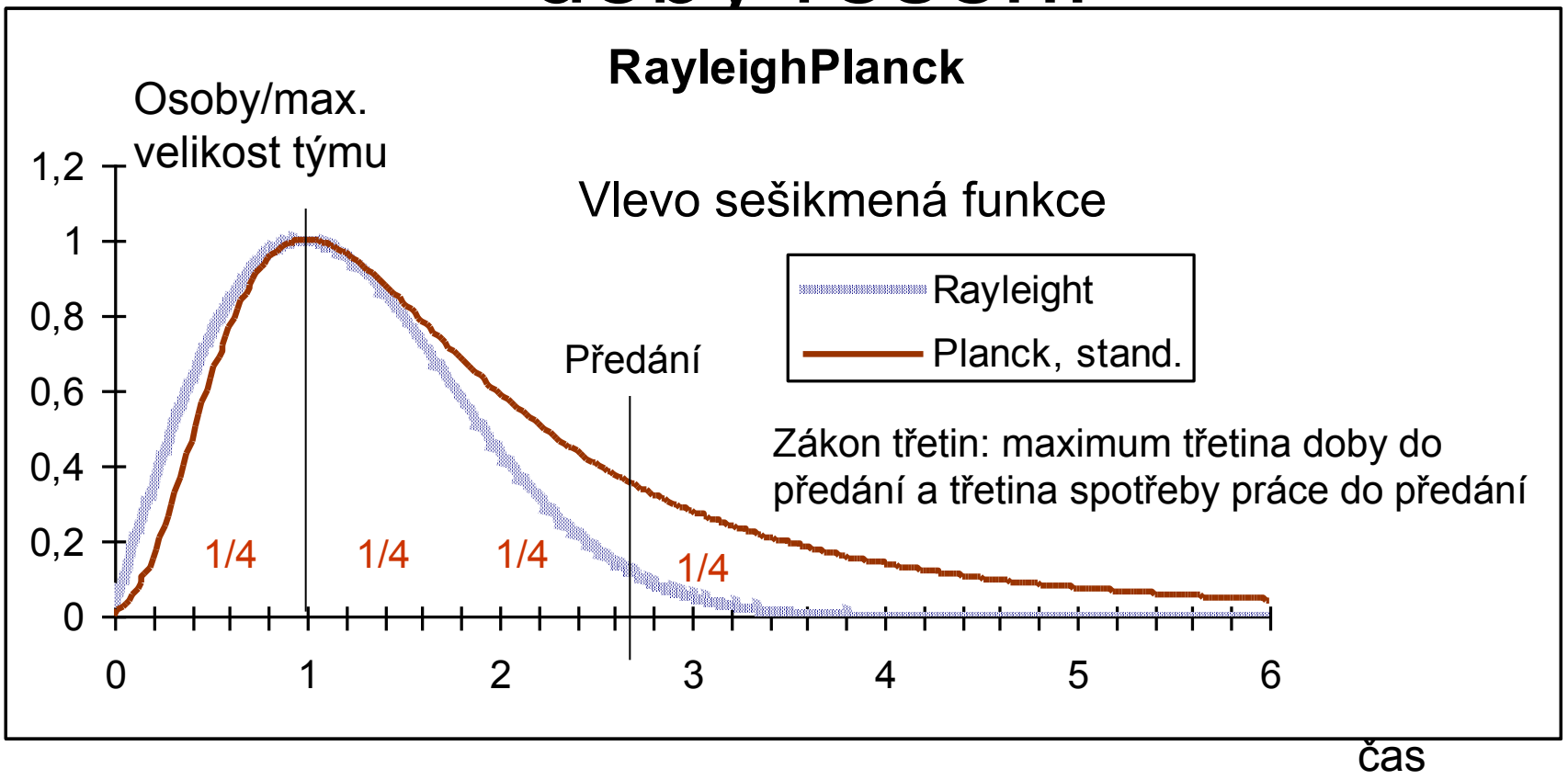
Modifikace Planckova modelu

A. Transformace nezávislé proměnné

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

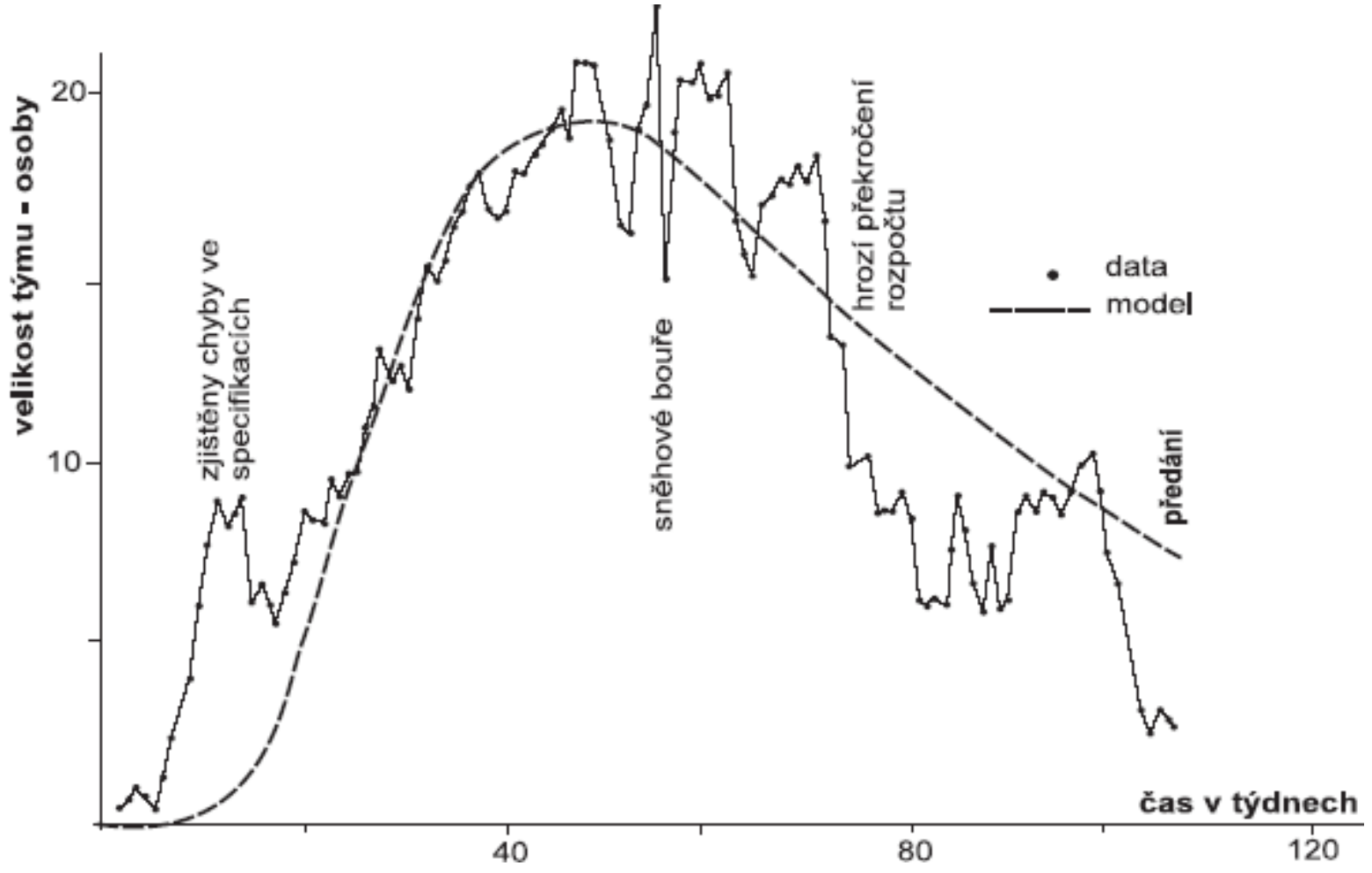
Dává dobré výsledky

Najímaný tým, vrchol a odhad doby řešení

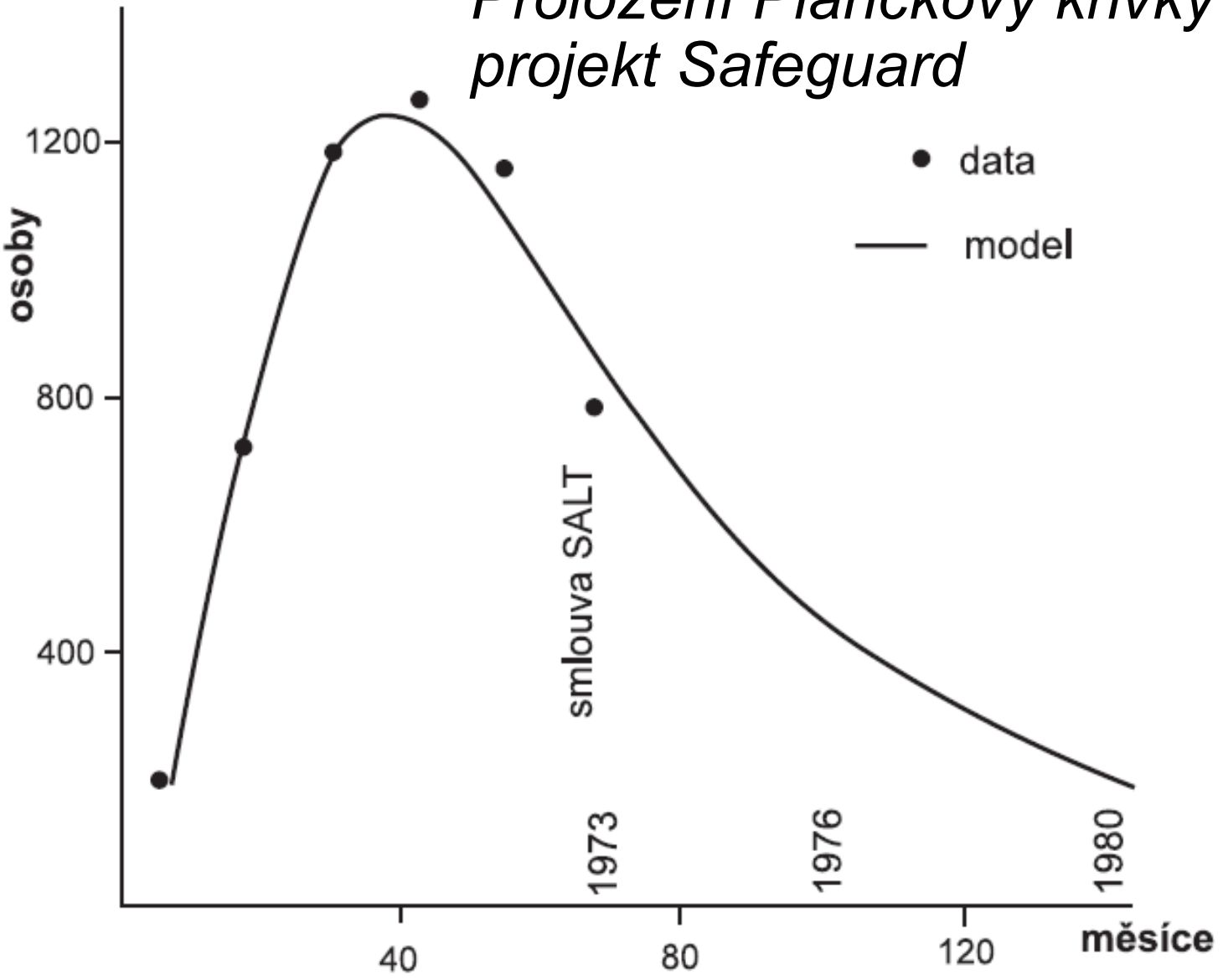


Transformace proměnných tak, aby max bylo v bodě 1 a mělo hodnotu 1 a v nule byla hodnota funkce prakticky nula. U pevného týmu odpovídá intensitě práce

Proložení Planckovy křivky pro SW řízení ponorek



Proložení Planckovy křivky pro projekt Safeguard



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

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

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

Paramery jsou $C, D, k, d\ q$. Stefan-Botzmannův 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 Putnamům zákon tvar
  - $Del \cong cPrac^{1/p}Doba^{q/p}$

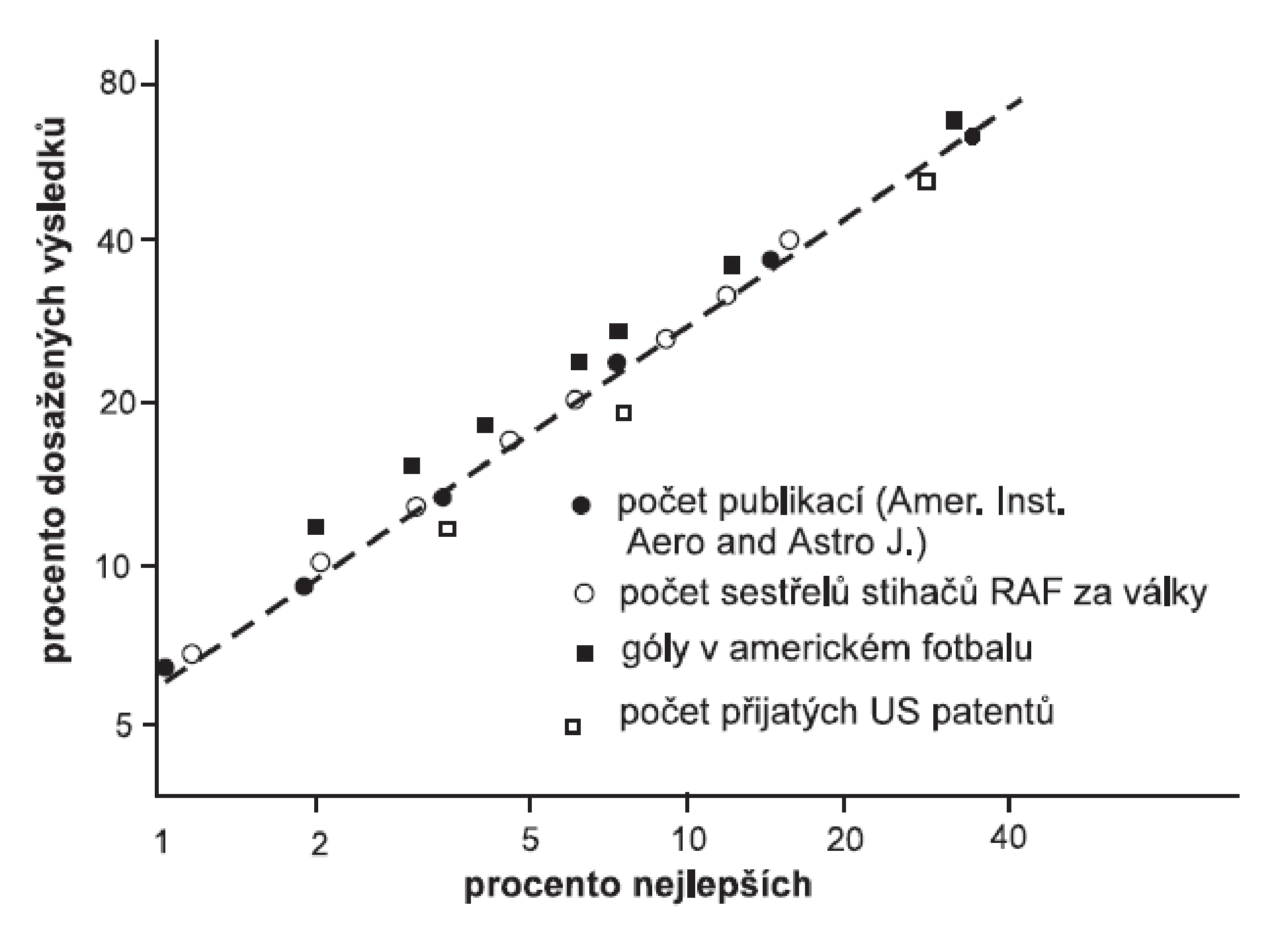
pro vhodné kladné p

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 řešeni 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 dvojitě logaritmickém měřítku dostaneme následující obrázek.


procento dosažených výsledků
80
40
20
10
5
1
2
5
10
20
40
procento nejlepších
počet publikací (Amer. Inst. Aero and Astro J.)
počet sestřelů stihačů RAF za války
góly v americkém fotbalu
počet přijatých US patentů

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

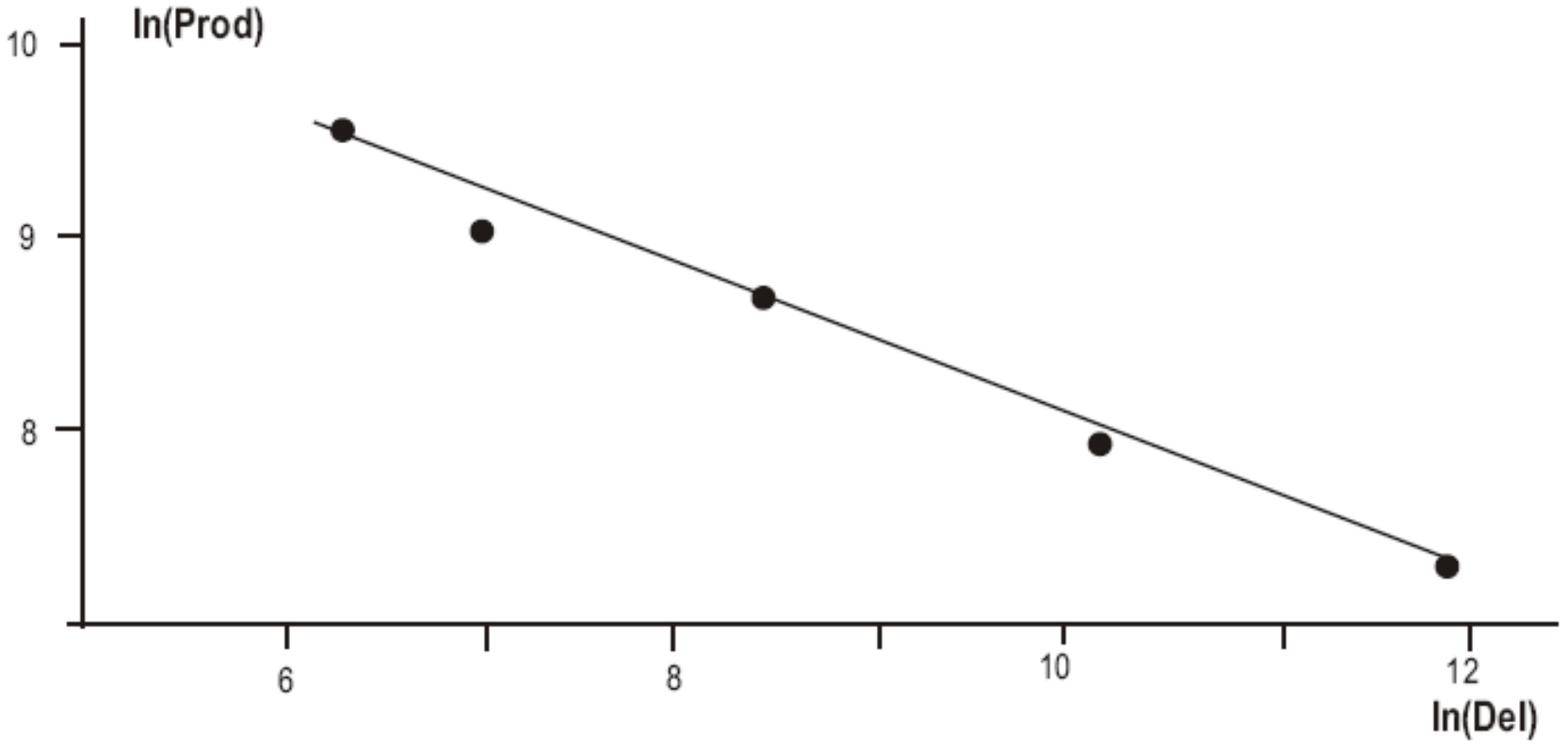
Seřadíme-li pracovníky podle výkonnosti, a budeme-li vynášet kolik udělal každý jednotlivec, bude mít příslušná funkce tvar

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

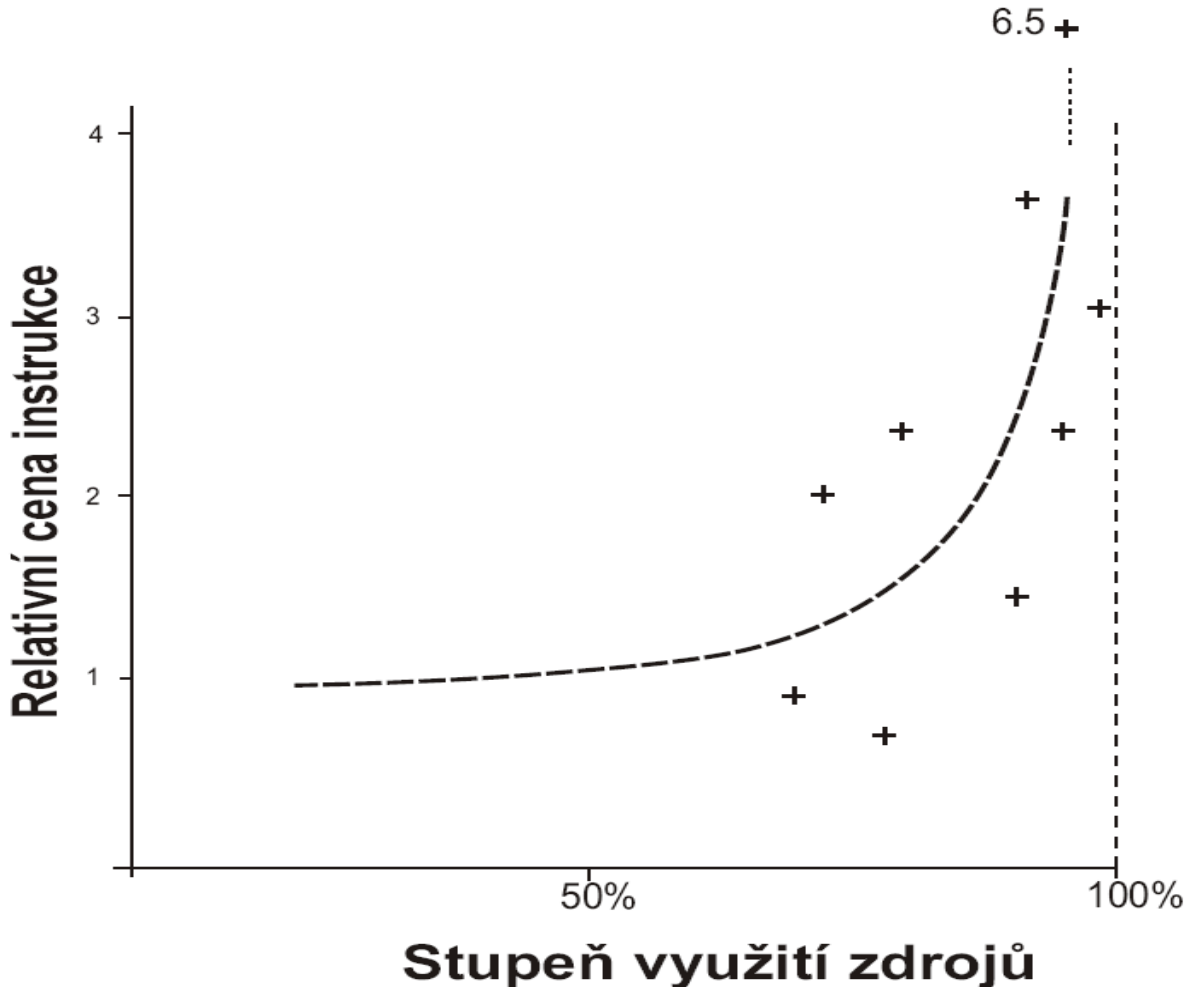
Použití

- 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 nevytíženi)

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



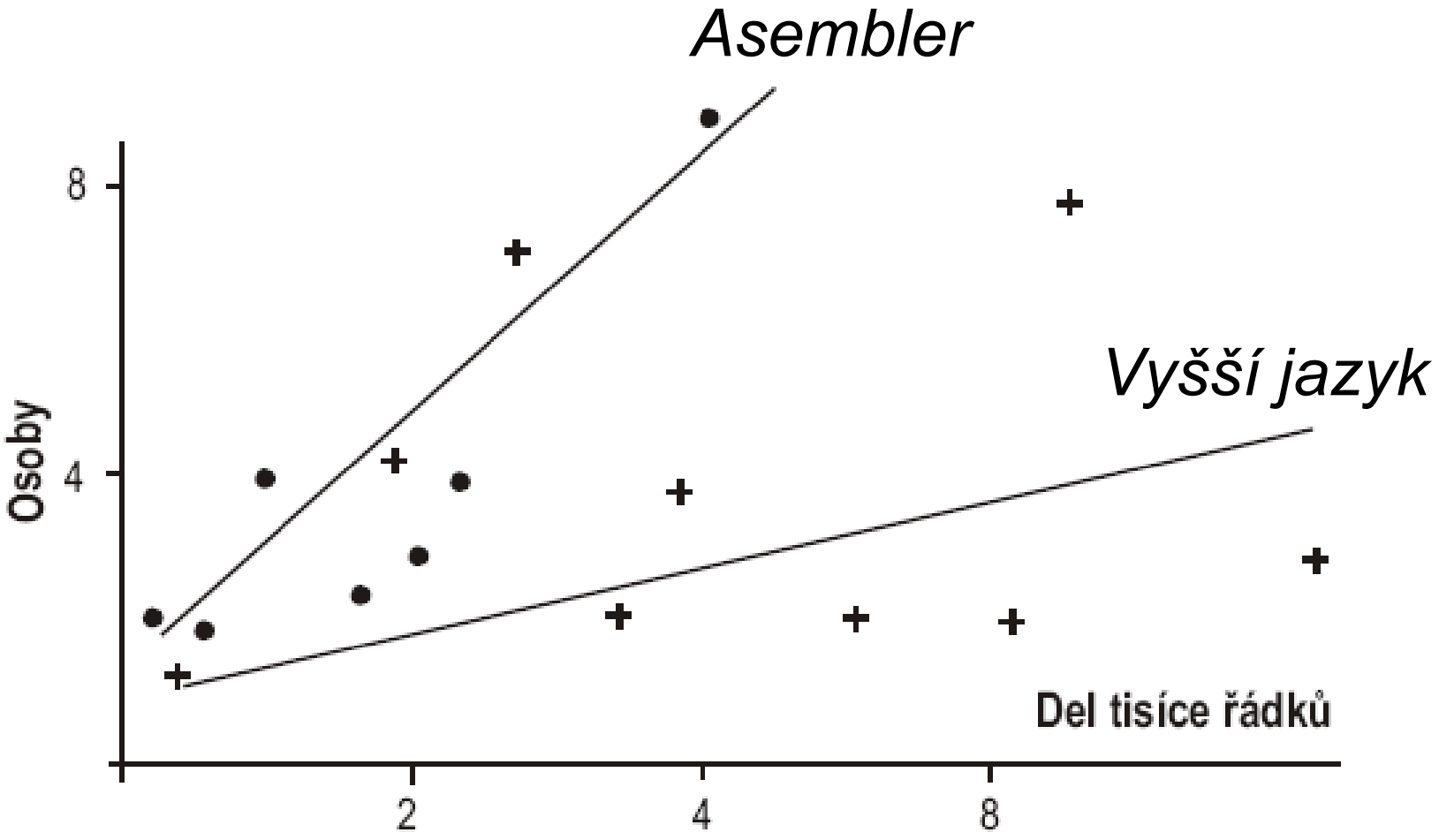
Vliv vytíženost HW



Využití

- U produktů, které nemají charakter masové spotřeby
  - Instalovat systém s 50-60% rezervou aby byl prostor na úpravy
  - Zvětšit reservu na alespoň 50%a, klesne-reserva pod 40%

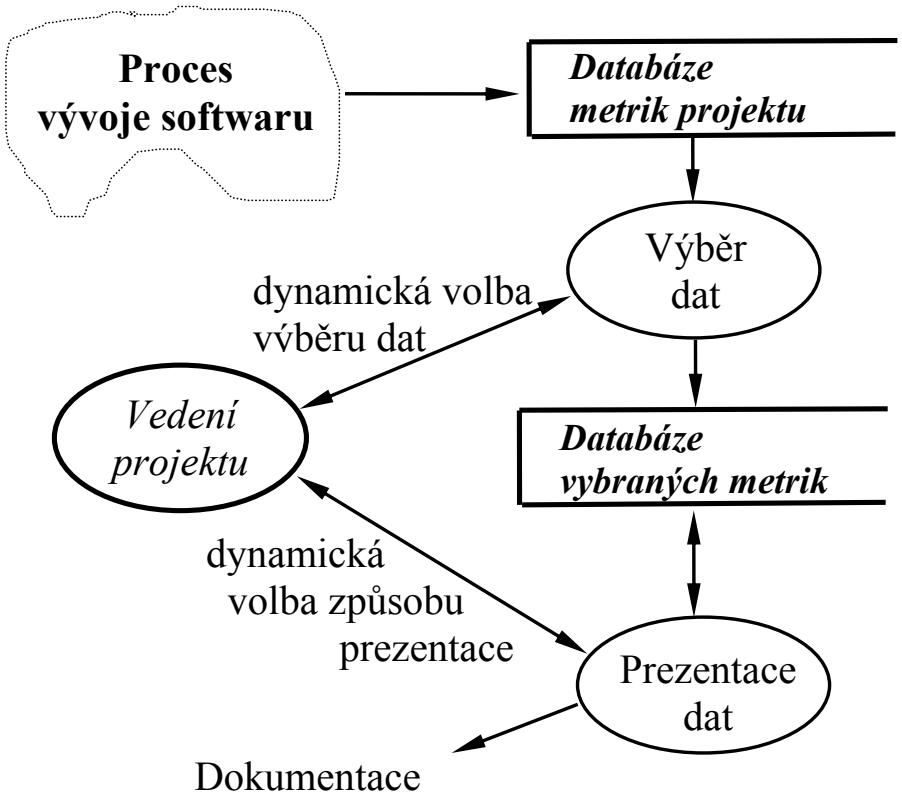
Náročnost údržby

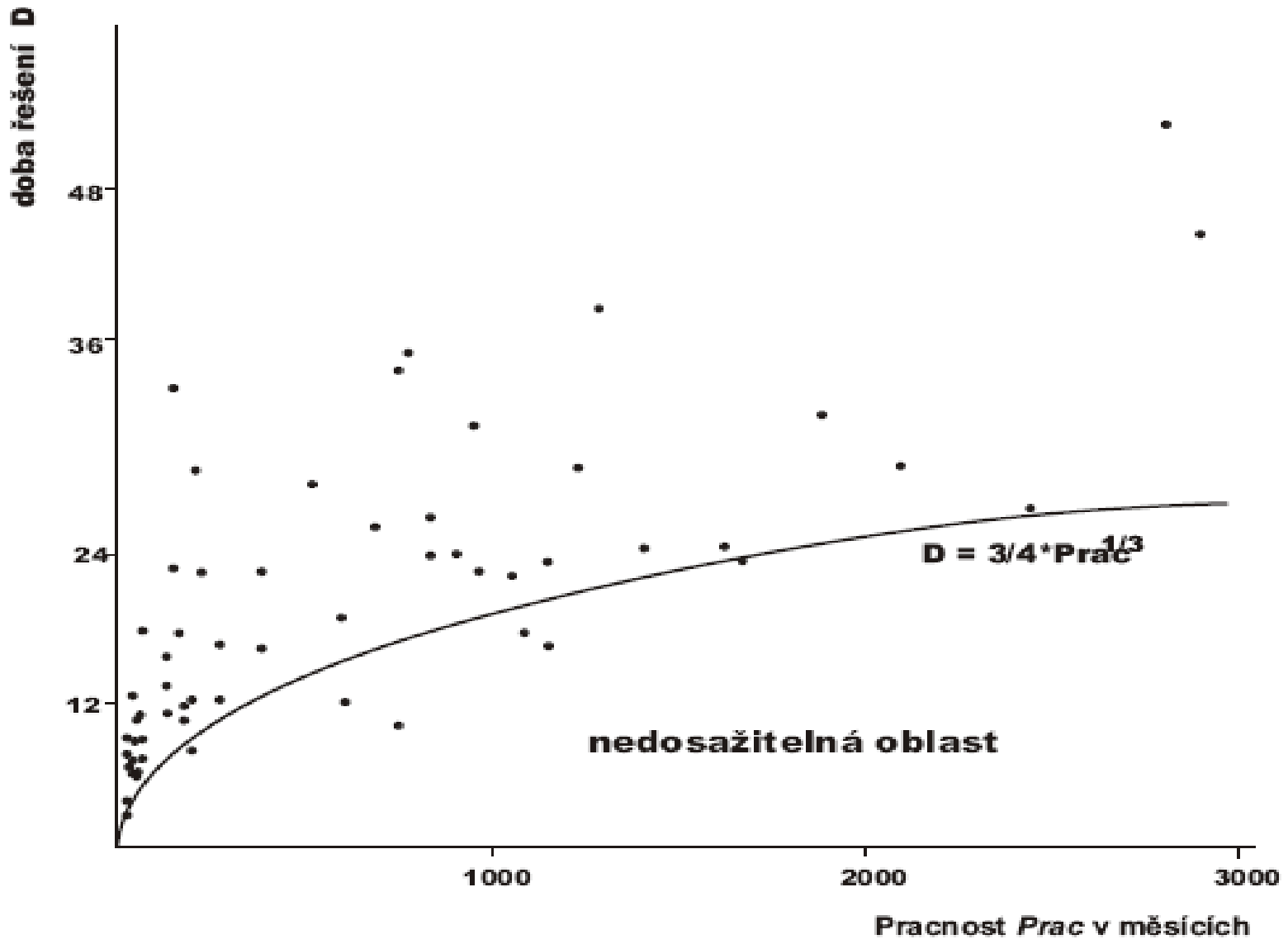


Využití

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

Databáze metrik




doba řešení D
48
36
24
12
D = 3/4*Prac^1/3
nedosažitelná oblast
1000
2000
3000
Pracnost Prac v měsících