Informační systémy

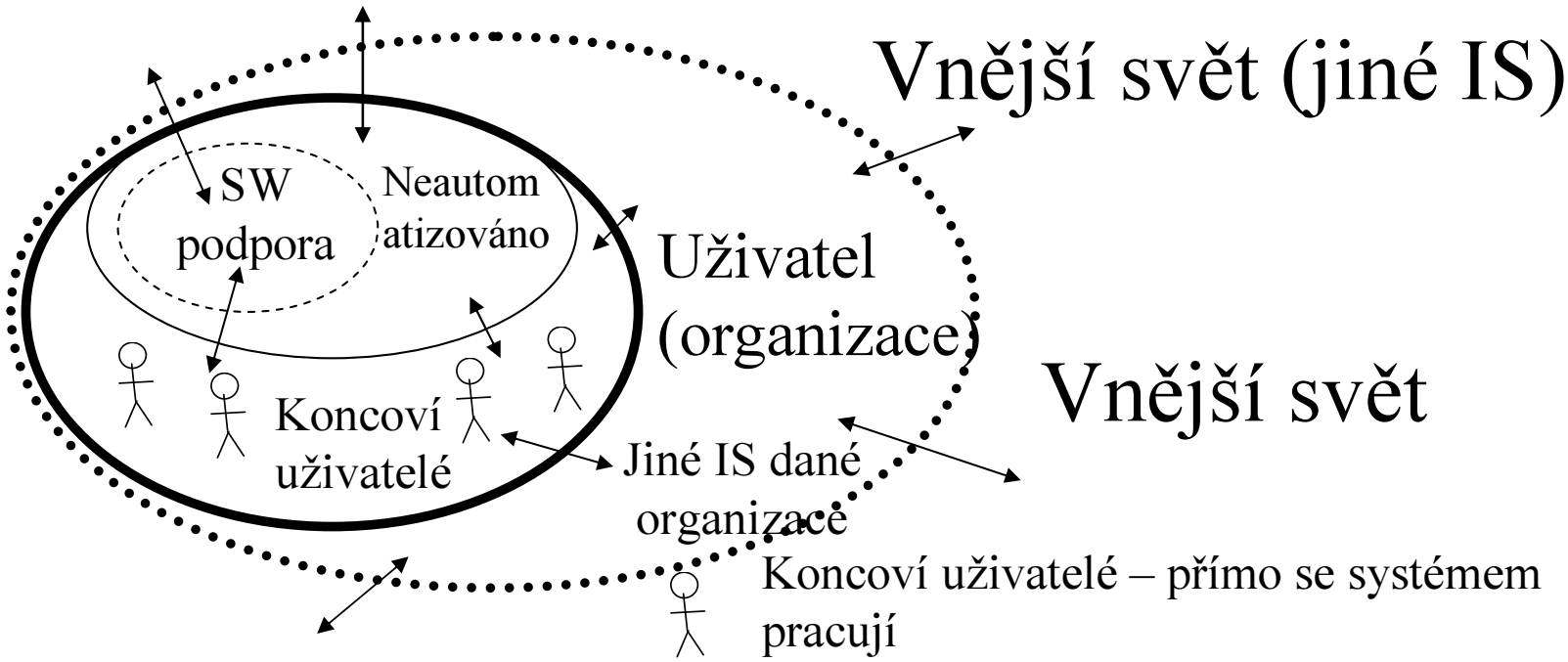
- Informační systém (IS) je systém umožňující ukládání, získávání a presentaci informací. IS je systém, tj. strukturovaný komplex technik, nástrojů, a zdrojů umožňující získávání, ukládání a poskytování informací uživatelům a jiným systémům. Výstupem IS mohou být přímo rozkazy osobám a signály procesům reálného světa (avionika letadla, reaktor, …). IS nemusí využívat SW, my se budeme zabývat případem, kdy IS využívá softwarovou podporu.

- IS jsou základním nástrojem globalizace světové ekonomiky, informatizace společnosti a změn ve výrobních procesech a změn ekonomických procesů

Systém

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

Informační systém je vždy součástí většího systému obvykle zahrnujícího i lidi a neautomatizované činnosti



Zásady

- IS může obsahovat i neautomatizované části
  - Automatizace jen když je to smysluplné, je to často porušovaná samozřejmá zásada, SW podpora by tedy měla být co nejmenší
  - Je nutné umožnit ruční kontrolu a zásahy chceme-li odpovědnost lidí za podnikové procesy
- IS spolupracuje s lidmi i s jinými IS
  - V organizaci i mimo ni

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

- Většina současných SW projektů se týká vývoje IS, IS jsou integrální částí technologií a dokonce i domácích spotřebičů
- IS jsou současně součástí reálných procesů
  - Akce nelze vrátit
  - Akce mohou vyvolat ztráty
- Mnohdy nelze používat klasické formy vývoje monolitů

Vývoj IS je složitý i když jsou jednotlivé funkce jednoduché

- IS mají mnoho funkcí požadavky na funkce se mění
- Jsou velké a mění se
- Obtíže při odhadu toho, jaké mají efekty a tedy i potíže při specifikacích toho, co mají dělat
- Jsou otevřené
- Dotýkají se zájmů lidí a zahrnují činnosti lidí
  - Lidé dělají chyby, mají různé zájmy, IS může ohrožovat jejich prac. místa
  - Musí se zaučovat a kontrolovat
  - Systém k nim musí být vstřícný

Landauer 1993, The trouble with computers, opakování

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

Landauer 1993, The trouble with computers

- Další příčiny
  - Nevhodně používáme (zátěž e-mailu, spamy, mnoho hnoje na www, automatizujeme kde co)
  - Efekty jinde, než čekáme, a jež proto dovedeme využít (výrobní systém, nové služby bank, rychlé publikování), nové typy služeb
  - V r. 1990 byla zrovna recese (to tvrzení jenom zmírňuje, nevyvrací)
- Pravděpodobnost, že se všichni, co do IT investují, mýlí je velmi malá
  - Všichni se nemohou mýlit pořád, inteligence lidí není právě malá

Ztráty v americké ekonomice v r. 1997

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

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

- 16 % v termínu a podle rozpočtu (< +20% překročení), tj úspěšných
- 32 % zrušeno před dokončením
- Více než polovina neúplná nebo dražší až třikrát, termín až dvaapůlkrát

Zdroj *Standish group* 1996, soubor velikosti 8000 projektů

2003 již 30% úspěšných, neúspěšných stále mnoho (30%).

Možné důvody:

- Uživatelé mají více zkušeností s IT a tedy rozumnější požadavky
- Existující systémy slouží jako prototypy vyvíjeného systému
- I vývojáři se poučili
- Obě strany lépe spolupracují, chyby ve vizích přetrvávají

Postup sledování vlivu faktorů v průzkumu Standish group

Dotaz ve tvaru:

Pro projekty, které skončily úspěchem

zaškrtněte ten faktor z následujícího seznamu, který byl rozhodující pro takový výsledek (tj. úspěch)

Obdobný dotaz pro ohrožené projekty a projekty, které byly zrušeny.

Pro každou skupinu projektů se pak vyhodnotilo v kolika procentech případů byl daný faktor uveden.

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

Zainteresovanost uživatelů	18
Podpora managementu uživatele	16
Jasné požadavky	15
Realistická očekávání	10
Správná dekompozice úkolu	9

Všimněme si, že mezi příčinami není technická kompetentnost vývojářů. Jistě by se pochlubili, kdyby si to mysleli. Přesto lze mít jisté pochybnosti.

Příčiny potíží v procentech

Uživatel neschopen říci, co chce	13
Neúplnost požadavků	12
Mizerná podpora managementů	8
Technologická nekompetentnost	7
Chyběly zdroje	6
Přehnaná očekávání	6
Termíny	4

Příčiny krachů v procentech

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

Hodnocení faktorů krachů projektů Standish

1994	1996	1998	2000
1.) User Involvement	1.) User Involvement	1.) User Involvement	1.) Executive Mgmt. Support
2.) Executive Mgmt. Support	2.) Executive Mgmt. Support	2.) Executive Mgmt. Support	2.) User Involvement
3.) Clear Statement of Requirements	3.) Clear Business Objectives	3.) Clear Business Objectives	3.) Experienced Project Manager
4.) Proper Planning	4.) Firm Basic Requirements	4.) Experienced Project Manager	4.) Clear Business Objectives
5.) Realistic Expectations	5.) Competent Staff	5.) Smaller Project Milestones	5.) Minimized Scope
6.) Smaller Project Milestones	6.) Smaller Project Milestones	6.) Firm Basic Requirements	6.) Standard Software Infrastructure
7.) Competent Staff	7.) Experienced Project Manager	7.) Competent Staff	7.) Firm Basic Requirements
8.) Ownership	8.) Proper Planning	8.) Proper Planning	8.) Formal Methodology
9.) Clear Vision and Objectives	9.) Ownership	9.) Ownership	9.) Reliable Estimates

Pozorování

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

Další důvody, těžko se měří

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

Stížnosti na vlastnosti vývojářů

Výroky vedoucích SW firem

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

Hackerský syndrom u informatiků

- Raději práce s počítači než diskuse s lidmi, nemilují práci v týmu
- Tendence k černobílému úzkoprsému uvažování
- Přeceňování čistě informatických znalostí a schopnosti programovat
  - V tomto ohledu může málokdo našim programátorům vyrovnat, snadno seženou dobrý job, cenné, nemusí ale stačit na celý život
- Práci považují hlavně za fascinující intelektuální hru
  - Positivní – tvorba volně šiřitelného softwaru
  - Negativní – tvorba virů, trojských koní a někdy přímo kriminalita

Silně vyvinuto u hackerů – **hackerský syndrom**

Hackerský syndrom, další symptomy

Podceňování neinformatických oborů, znalostí a schopností koncových uživatelů a významu spolupráce s nimi

Odpor k filosofii experimentálních věd a zvláště k matematické statistice jako nástroji analýzy dat

- To je ale nutné k podpoře managementu uživatele i vlastní firmy
- Bez toho nelze kvalifikovaně hodnotit kvalitu softwaru a zlepšovat procesy vývoje SW
- Je to nutný předpoklad k porozumění potřebám zákazníků
- Kvalita dat zásadním způsobem ovlivňuje specifikaci požadavků (kritický řetěz při řízení projektů, rozvrhování, atd. bude diskutováno níže) ISO250xx

Hackerský syndrom, symptomy 2

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

Rande u počítače

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

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

Rande u počítače

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

- Nedisciplinovanost a nesoustředěnost
- Tendence k přetěžování organizmu až k vyhoření
- Neschopnost spolupráce s neinformatiky

Pokus zorganizoval doc. Jirovský

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

HS blokuje uplatnění v kvalifikovanějších rolích při vývoji softwaru (analytik, vedoucí projektu), analytik se dobře uplatní i mimo infromatiku

Z toho důvodu se jako vedoucí IT oddělení se často uplatňují lidé, kteří studovali něco jiného než informatiku

Délka profesní kariéry je mezi 25 až 50 roky. Během této doby dojde k významným změnám na trhu práce.

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

HS zhoršuje adaptibilitu na změny na trhu práce (dnešní studenti půjdou do důchodu po více než 40 létech, programátorská virtuosita se ztrácí v 35 létech, schopnost analýzy se ztrácí později, viz praxi IBM, kde se zbavují třicátníků, jsou výjimky – J. Moudrý), je proto velmi žádoucí, aby se neuzavírala možnost uplatnění mimo informatiku. Pro to je dobrou průpravou práce analytika – má i jiné dovednosti než ty, které se uplatní jen při práci s počítačem.

Proč je třeba prevence hackerského syndromu

Hackerský syndrom se těžko se léčí

- Predispozice a důvod volby dráhy informatika při rozhodování, co studovat.
  - Nevíme, jaké je rozložení talentů a zda se prevence vyplatí
- Utvrzován běžnou praxí výuky. Ve výuce je obtížné navodit situace ukazující např. potřebu spolupráce s koncovými uživateli nebo dokumentování
- Často spojen s programátorskou virtuozitou – cesta k dobrým rychlým výdělkům (a k podceňováním studia a hlubších znalostí v informatice a hlavně mimo informatiku).

V některých zemích nejsou místa informatice už dnes

- Austrálie – největší míra nezaměstnanosti je mezi informatiky
- Anglie – prudký pokles zajmu o studium informatiky (až o 50%)
  - Studium je náročné (matika) a uplatnění horší
- USA – stálé stesky na konferencích na konkurenci programátorů z Indie a nejnověji z Číny.

Deloite and Touche 1999, situace v ČR

- Profesní znalosti +++
- Používání moderních technologií ++
- Marketing -
- Nejasnost cílů
- Nestálost vztahů
- Management není na výši

Proč jsou IS složité i když dělají jednoduché věci, pokračování

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

IS jsou složité i když dělají jednoduché věci

- Nedohlédnutí efektů nasazení
  - Kvalifikace lidí
  - Organizační změny
  - Efekty jinde, než se čekalo, nevyužití synergie spolupráce lidí a softwarových systémů
  - Efekty se objeví později a závisí na systémových změnách
  - Zvýšení produktivity, možnost na některé práce používat méně kvalifikované síly
  - Ztráta produktivity při nevhodném použití (mail, Internet)

IS jsou složité i když dělají jednoduché věci

Někdy se výhody přeceňují, jindy nedoceňují

- Slepá víra ve výstupy IS bez ohledu na kvalitu použitých dat(hnůj tam hnůj ven) a chyby ve specifikacích, závislost a počítačích. Význam některých aspektů kvality dat je silně podceňován.
- Snaha o alibi (jinde SAP funguje), podceňování nutnosti ručit za svou práci

IS jsou složité i když dělají jednoduché věci

Podceňování nevýhod

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

IS jsou složité, věcná dimenze

- Běží stále po dlouhá léta
- Zabezpečení, některé IS mají charakter kritických aplikací (mohou způsobit ztráty životů nebo alespoň peněz)
- Obtíže se specifikací požadavků, především nejasnost cílů
  - Nečekané efekty (příklad výrobního systému)
  - Časté změny
  - Nedostatečná analýza potřeb a možností IT a lidí

IS jsou složité, věcná dimenze

- Mnohé IS vyžadují specifické formy vývoje při kterých nejsou využitelné klasické metody a také zavedené nástroje jako je model driven architecture (velké konfederace popsané níže - servisní orientace)
- Složitost vývoje pro některé typy použití (počet koncových uživatelů, velikost,…)

IS jsou složité

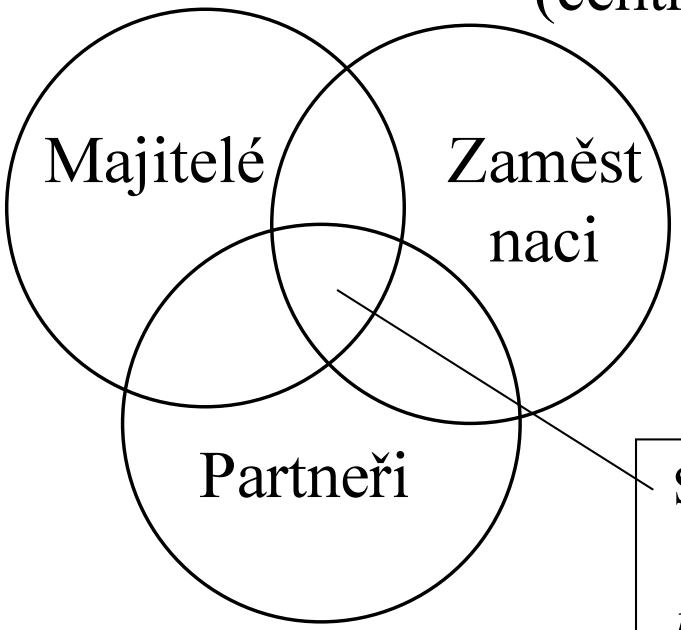
- Jsou nositeli nových paradigmat v softwaru.
- Zvládnutí nového paradigmatu je velice obtížné

Podceňování lidské dimenze
Koalice zainteresovaných v podniku

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

Koalice v podniku

Platí pro strojovou byrokracii - práva a povinnosti z pozice v (centralizované) hierarchii



Společný zájem, dlouhodobá prosperita *IS musí být prospěšný pro všechny členy koalice*

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

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

- Udržet dobré pracovníky – dobře platit, mimoplatové stimuly
- Nežádat nereálné mzdy (viz příklad Baťa za krize)
- Ofensivní strategie, přednost má úspěch na trhu a až pak propouštění,
- Snažit se vyjít vstříc partnerům
- Podpora strategie firmy (to ale závisí i na SW architektuře)

Koalice v profesních byrokraciích

- Profesní byrokracie – hlavní vliv mají lidé, kteří na to mají papír
  - Jsou zvoleni nebo mají profesní kvalifikaci
  - Jsou jmenováni na určitou dobu
- Příklady
  - Státní a místní správa
  - Zdravotní organizace
  - Školy
  - Ad hoc spolupráce (CRM)

Subjekt ve vztahu ke státní správě může vystupovat v různých rolích

- **Daňový poplatník** – co nejméně platit
- **Příjemce péče** – co nejvíce brát (zdraví, bezoečnost, podpory)
- **Úředník** – co nejvíce moci a platu, co nejméně práce (ta se navíc obtížně měří)
- **Politik** – prosazovat zájmy svýchvoličů, udržet se u moci (i na účet fungování státu). Přímá odpovědnost všech za svá rozhodnutí je omezená.
- *Společný zájem – prosperita státu* (fungující infrastruktura, schopnost obrany, policie, sociální smír, vzdělávání (Marie Terezie!, …). Společný zájem není dostatečně zřejmý – příklady z historie, kdy při hrozbě zvenčí nebyl o udržení centrální moci zájem (Čína)

IS se v profesní byrokracii buduje obtížně

- IS by měl usnadňovat práci úředníků, usnadňovat úřední procedury pro občany a umožňovat analýzu politických rozhodnutí (např. ve zdravotnictví nebo při školských reformách). Zvláště slabé je to s tou analýzou.
- Nějakým způsobem by měl podporovat odpovědnost a spolupráci lidí v různých rolích.
- Podobné úkoly jsou i v jiných profesních byrokraciích (školy, nemocnice, GSM)

IS by měl podporovat sedm S

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

Strategie versus operativní řízení

Strategie:

klíčová rozhodnutí na dlouhou dobu

- Převažují koncepce, dlouhodobé problémy
- Velký význam zkušenosti a intuice
- Převažuje mezi úkoly vyššího managementu
- Značné využívání externích a historických informací
- Malá opakovatelnost akcí
- Silné zastoupení analýzy dat
- Obvykle pomalá odezva na provedená opatření

- *Systém by měl podporovat operativu i strategii, má proto dvě technicky odlišné části*

Strategie versus operativní řízení

Operativa
Převažuje řízení ze dne na den

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

Taktika
Řízení s výhledem měsíců až let

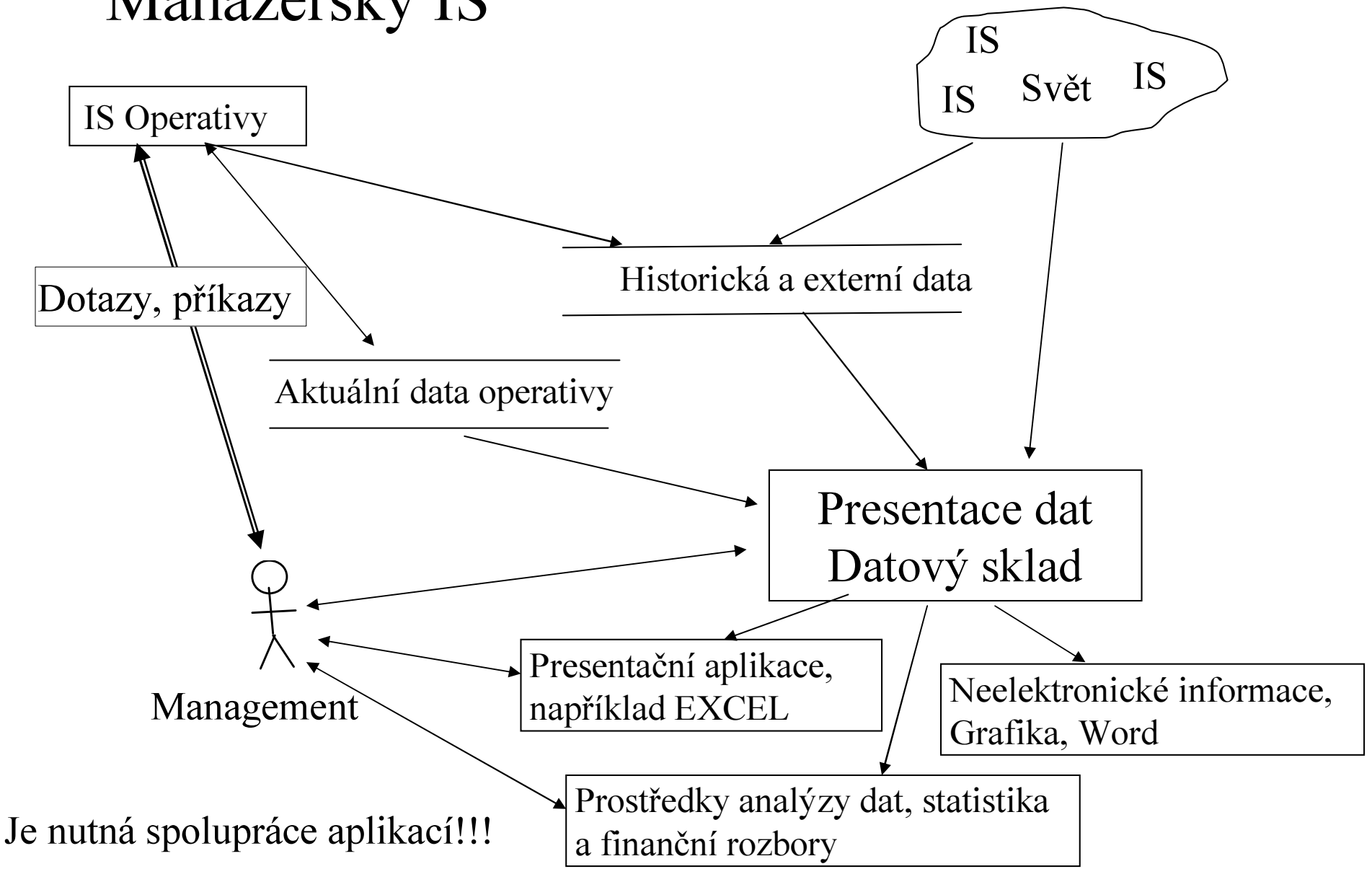
Mix metod operativy a podpory strategie (běžné obchodní akce a procesy, roční plány)

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

Pozorování

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

Manažerský IS



Je nutná spolupráce aplikací!!!

Pozorování

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

Je žádoucí, aby MIS spolupracoval s obdobnými externími systémy.

Manažeři vyžadují podporu své intuice.

Podpora strategie firmy

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

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

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

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

Úzké místo může být jiné pro různé pro různé doby výhledu

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

IS by měl usnadňovat hledání úzkých míst a pomáhat je řešit

Vývoj od začátku versus nákup (customizace)

- **Vývoj od začátku „na míru“**
  - Převažují nově vyvinuté moduly
  - Požadavky nemusí být omezovány tím, jaký SW máme nebo jaký je ke koupi
- **Customizace - Nákup softwarového balíku a jeho přizpůsobení konkrétním podmínkám (přesnost dat, formáty, účetní schéma …)**
  U velkých firem dnes customizace převažuje, trend se ale možná obrací

Životní cyklus softwaru, vývoj od začátku

1. *Marketing a specifikace cílů (formulace problému)*, hledání odpovědí na otázky *proč* a rámcově *co*. Výstupem je ***vize***
2. *Specifikace požadavků*, formulace přesné odpovědi na otázku *co*, zčásti na otázku *jak*. Výstupy
   - Model systému, diagramy a (formalizované) požadavky,
     - závisí na architektuře SW modelovacích prostředcích a použitém paradigmatu (např objektovém), musí být srozumitelné oběma stranám
   - Termíny řešení, náklady, zdroje
   - Oponentura *feasibility study*
3. *Návrh systému.* Řešení technických otázek dekompozice, návrhu rozhraní, struktury dat, algoritmů a volba softwarových vývojových prostředků a systémového softwaru. Dostatečně podrobná dokumentace (u agilního vývoje nemusí být rozsáhlá).

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

Výše uvedený cyklus, pokud se provádí striktně, se nazývá metoda vodopádu a je kritizován

- Existují modifikace, vycházejí ale z metody vodopádu a budeme se jim věnovat později

Obsah vize

- Ekonomické a organizační důvody, pro zavedení/modernizaci IS,
  - Co by měl systém dělat (rámcově), proč je třeba změna
  - Přínos by se měl dát vyčíslit v penězích, nebo alespoň v měřitelné změně, která by měl zlepšit ekonomické vyhlídky firmy a její chování na trhu.
  - Hodnocení, zda lze systém koupit na trhu (předběžná marketingová analýza)

Obsah vize

- Rizika spojená s realizací
- Perspektiva projektu (kolik uživatelů, opakovaná instalace)
- Vývoje nebo customizace
- Předběžný odhad termínů realizace a akceptovatelných nákladů

Vize je podkladem pro předběžnou smlouvu.

Životní cyklus softwaru, customizace

1. *Marketing a specifikace cílů (formulace problému)*, hledání odpovědí na otázky *proč* a rámcově *co. Výběr dodavatele*.
2. *Specifikace požadavků*, formulace přesné odpovědi na otázku *co*, zčásti na otázku *jak* od daného dodavatele.
   - Model systému, diagramy, závisí pravidlech dodavatele modelovacích prostředcích a použitém softwaru,
   - Výběr modulů a funkcí
   - Termíny řešení, náklady, zdroje
   - Oponentura *feasibility study*
3. Customizace *systému (generace systému)*. Zadávání funkcí, jejich vazeb a parametrů (např. přesnost, formáty dat). Tato etapa je poměrně pracná a náročná na práci expertů

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

Výhody customizace

- Ověřený dodavatel, zná obor, ověřené techniky specifikací a oživování, know-how z mnoha instalací. Dobré reference
- Alibi pro management (jinde to přece fungovalo)
- Menší nebezpečí selhání projektu a toho, že dodavatel opustí trh
- Úspory-cena (má to ale háček, viz níže), hlavní úspora je u údržby
- Velká nabídka funkcí (ale nebezpečí, že se koupí i zbytečnosti a že cena proto bude zbytečně vysoká)
- Rychlejší realizace (ne závratně)
- *Osvědčuje se díky malé pravděpodobnosti totálního selhání projektu, selhání projektu je výjimečné, plný úspěch také nebývá častý*

Nevýhody customizace

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

Nevýhody customizace

- Ztráta vlastních znalostí a kvalifikace a nezdravá závislost na dodavateli, někdy i nemožnost používat i svoje řešení a řešení třetích stran
- Odstraňuje spíše konkurenční nevýhodu než přináší výhodu (ostatní mají totéž). Vhodné spíše pro operativu.

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

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

Vliv programovacího jazyka

- Programovací jazyk ovlivňuje hlavně kódování (1/6 nákladů), přechod z jednoho PJ vysoké úrovně na jiný PJ vysoké úrovně přinese zrychlení kódování nejvýše dvakrát a tedy celkovou úsporu do deseti procent. Mohou se ztratit starší programy a knihovny a znalosti
- Proto se nové PJ uplatňují převážně v nových nikách (možná výjimka C#)
- Velkou roli hrají vývojová prostředí (další možná výjimka)

Pozor

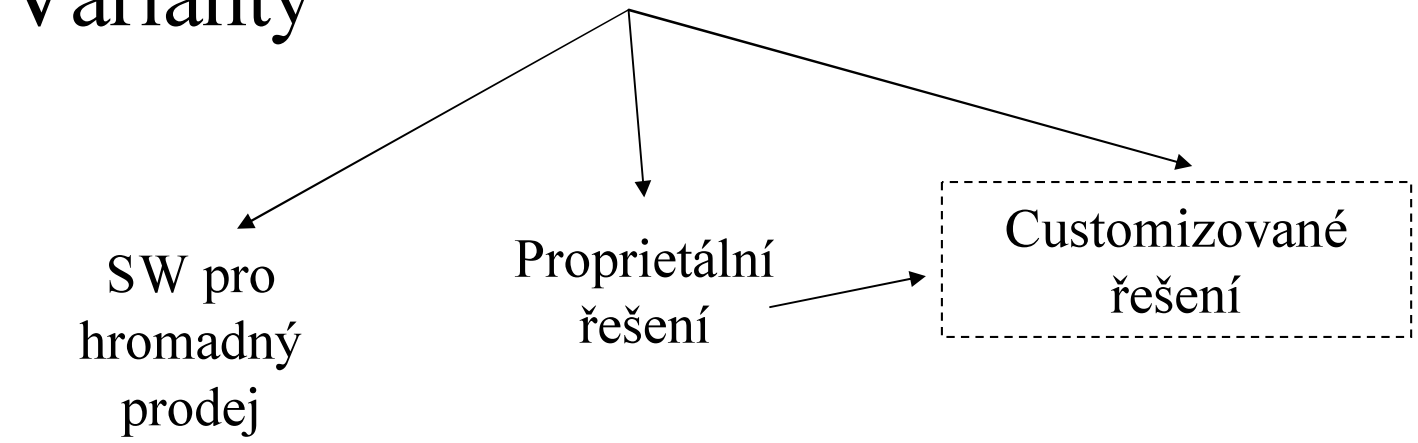
- Customizace ušetří jen max. 50% času a 50% nákladů
- Hlavní úspora ze sdílení údržby pro více zákazníků
- Konfekce, ale dá se nosit
- Dlouhodobé zkušenosti dodavatele
- Výhoda nového programovacího jazyka není ve zrychlení programování, přínos jsou nástroje a nové úlohy.

Typy softwaru

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

Typy softwaru

Varianty



Ani pouze od začátku, ani pouze customizace!

Umožnit použití existujících vlastních aplikací, nakupovaného SW a nově vyvíjených věcí vhodnou architekturou SW - servisně orientovanou architekturou
Výrobci SW z toho nemají radost

Kdo systém vyvíjí

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

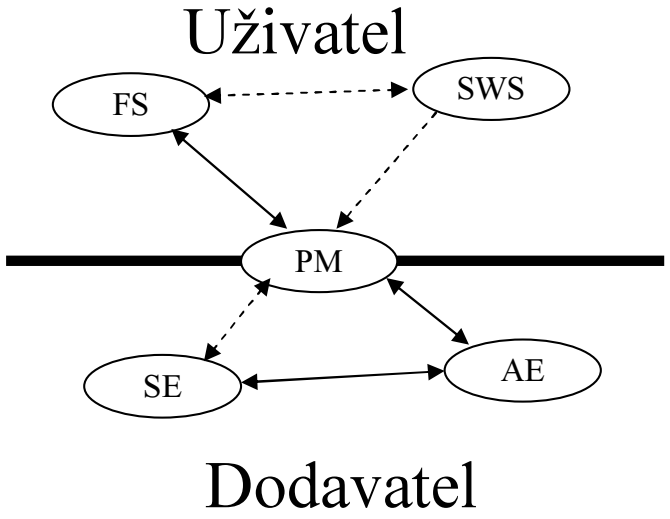
Role ve velkých projektech, iterativní vývoj (Standish)

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

PM je úzké místo.

Vhodné pro velké a kritické systémy.

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



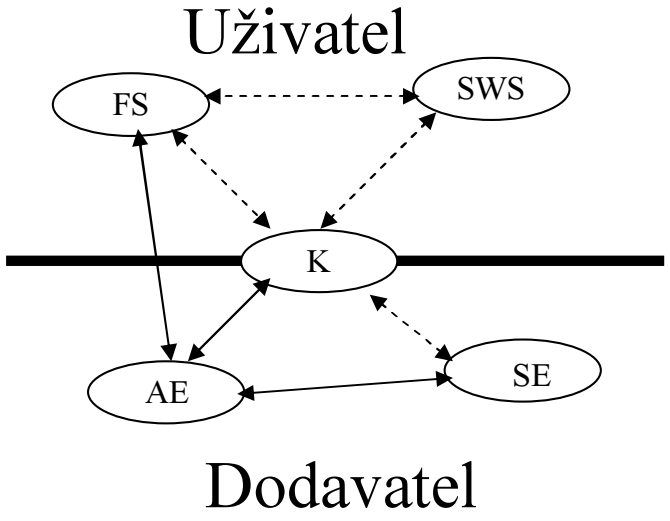
Role v nekritických projektech (Agilní vývoj)

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

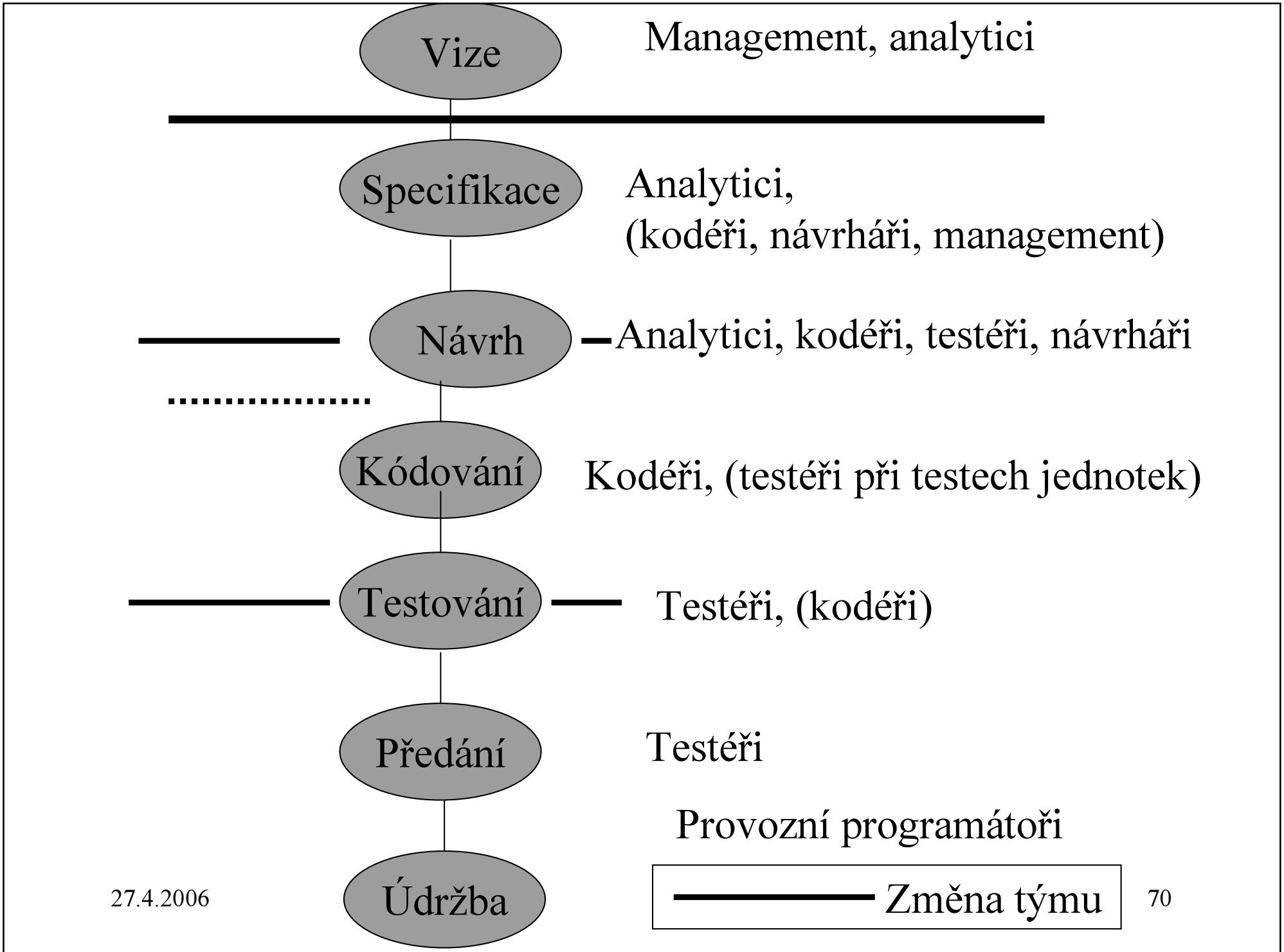
Úzké místo je udržení integrity a spolehlivost.

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

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



Budeme se nejprve věnovat vývoji od začátku jako základu pokročilejších technologií

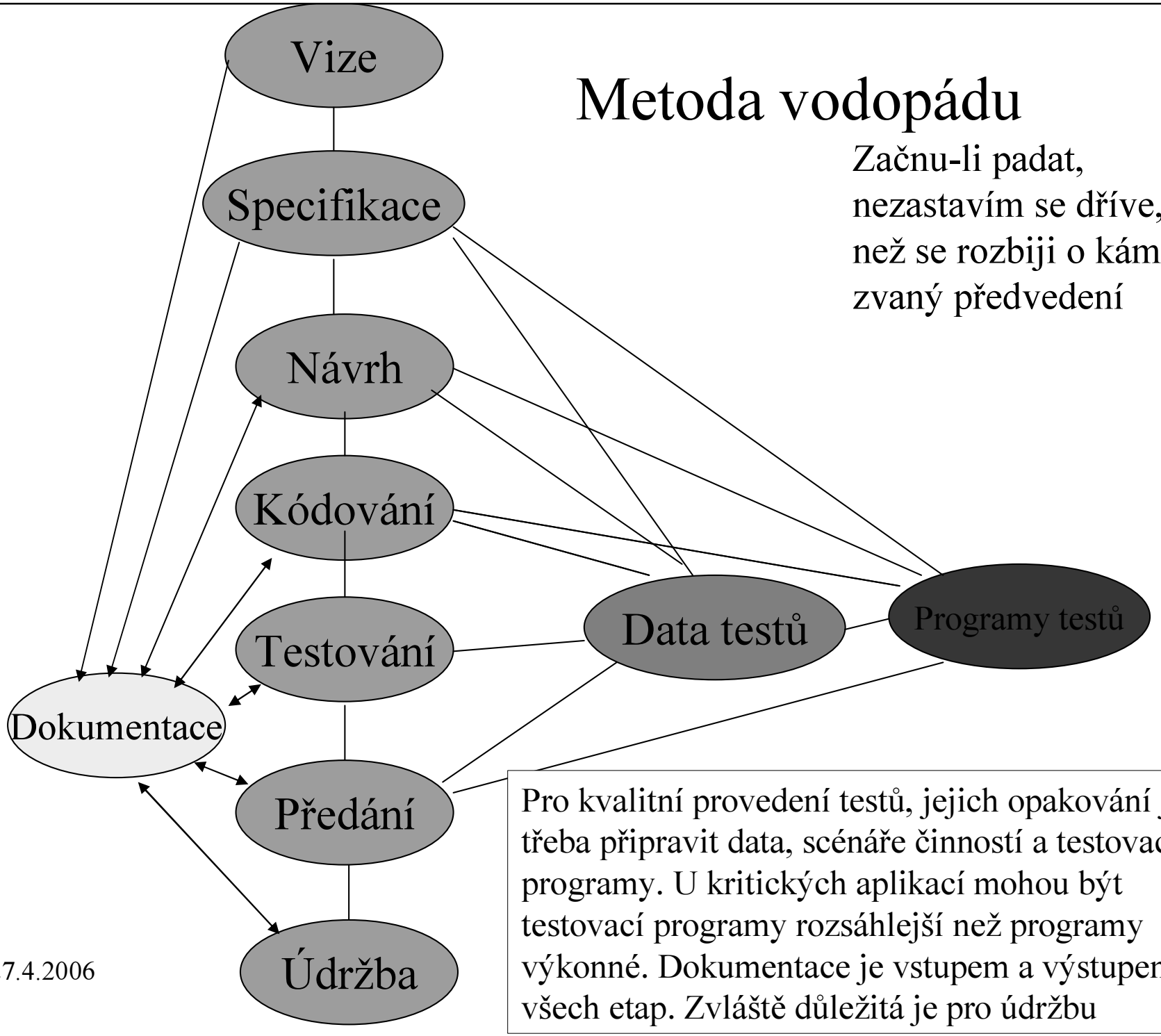

Vize
Management, analytici
Specifikace
Analytici,
(kodéři, návrháři, management)
Návrh
Analytici, kodéři, testéři, návrháři
Kódování
Kodéři, (testéři při testech jednotek)
Testování
Testéři, (kodéři)
Předání
Testéři
Provozní programátoři
Údržba
Změna týmu

Úzké místo je specifikace požadavků a často také kvalita vize

Nedostatek je analytiků.

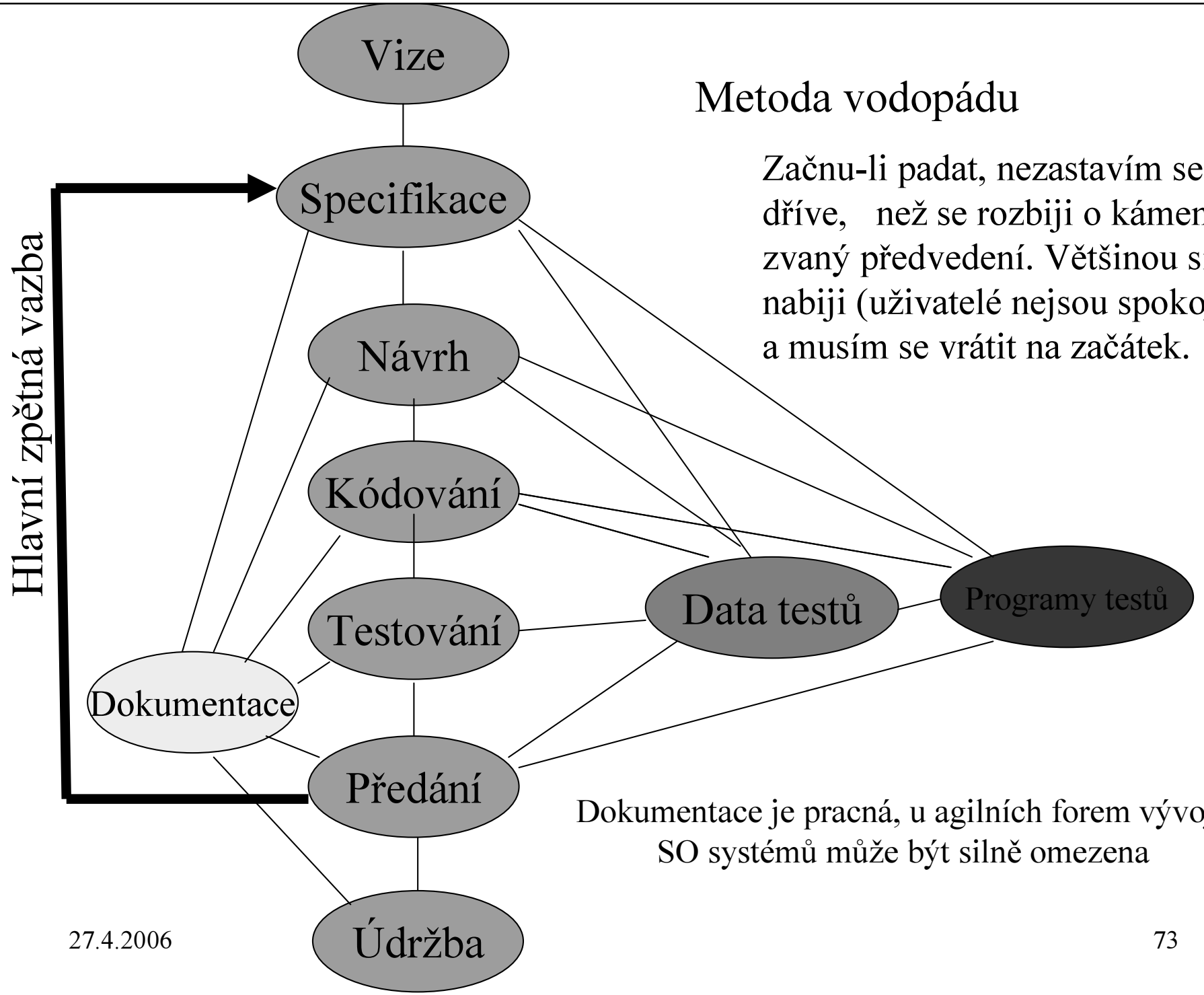
Metoda vodopádu

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



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

Metoda vodopádu



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

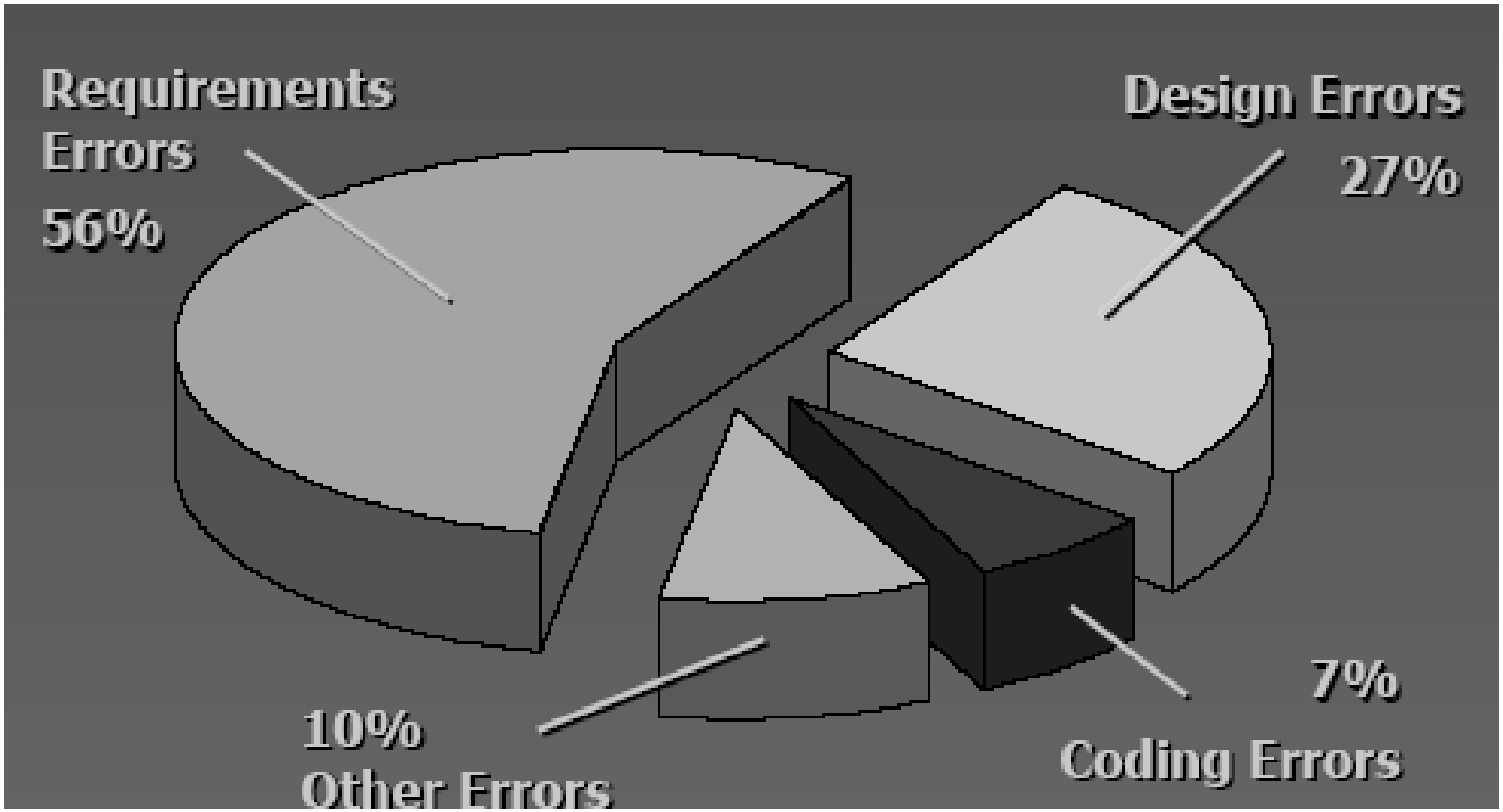
Dokumentace je pracná, u agilních forem vývoje SO systémů může být silně omezena

Kde je úzké místo při vývoji SW

Problém jsou specifikace, ukažme si příklady

V následujících snímcích jsou použity výřezy ze snímků prezentace:

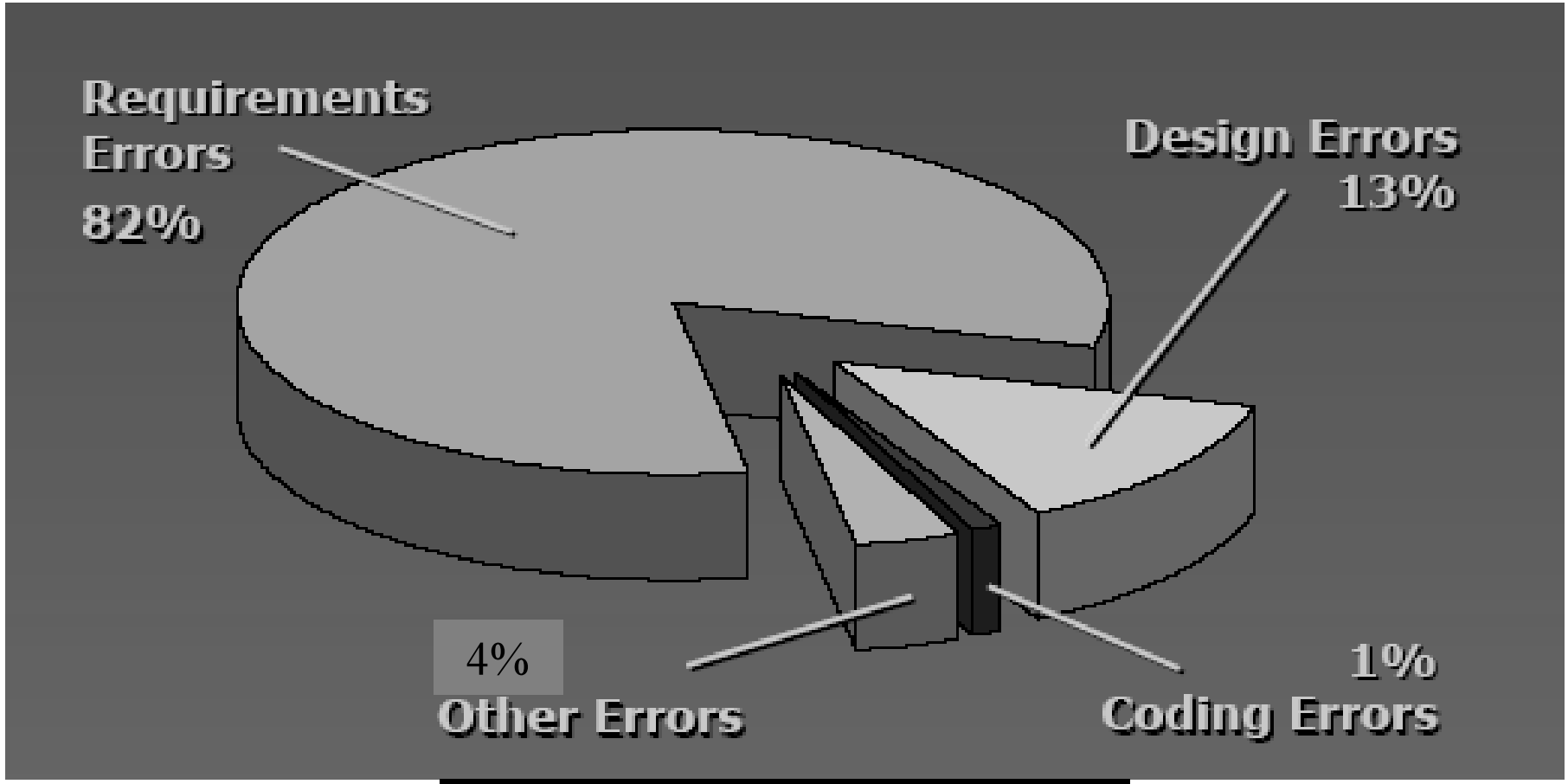
56% chyb vzniká ve specifikacích, stojí více než 80% nákladů a času na odstranění, stojí za většinou neúspěchů projektů a za nespokojeností při provozu



Zdroj: An information system manifesto, James Martin

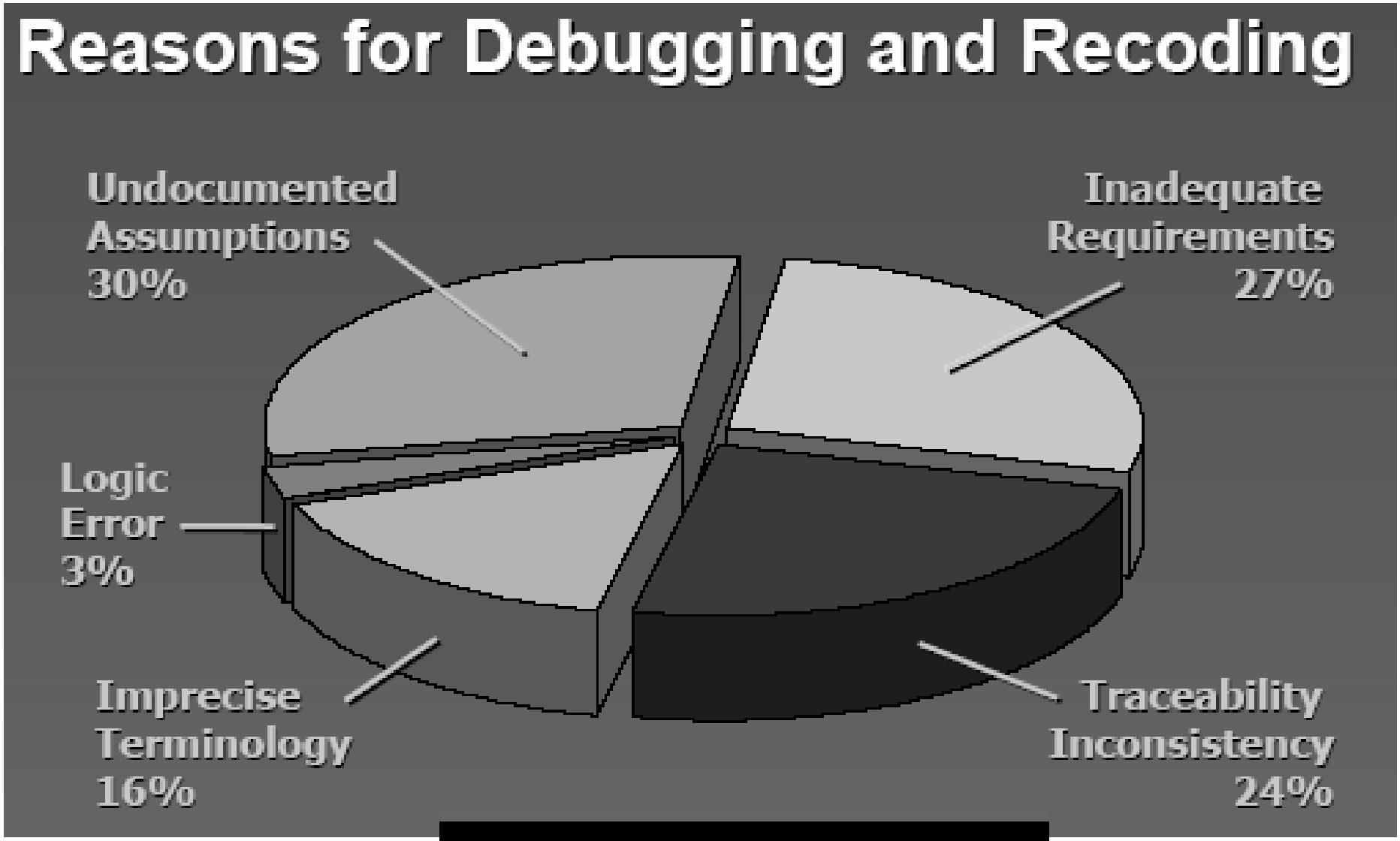
Cena odstranění chyb

82% ze specifikace požadavků, 1% kódování



Source: “An Information Systems Manifesto”, James Martin

Jak dochází k chybám při specifikacích



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

Vlastní zkušenost přednášejícího

Hlavním zdrojem problémů jsou nedokumentované a hlavně opomenuté předpoklady, opomenuté požadavky a požadavky u nichž se neví, jak se k nim došlo (nelze je vystopovat). To vše vede k neúplným a neadekvátním požadavkům a k nekonzistencím a ztrátě znalostí a tom, proč se věci řeší tak a ne jinak (traceablity). Tj. podobně jako v předchozím slajdu spoluodpovídají za 80% problémů ve specifikacích.

Příklad zamlčených předpokladů

Tunel Mrázovka ulehčí dopravě, protože auta rychleji projedou centrem a nebudou překážet (u Pisáreckého tunelu v Brně je to podobné).

Zamlčený předpoklad – v okolí tunelu je průjezd volný a to neplatí.

Obchvat pomůže, protože odvede transitní dopravu

Zamlčený předpoklad – transit sice tvoří značnou část pražské dopravy, ale lidé jezdí do centra, protože jen tak mohou projet Prahou, nebo ve středu pracují a jezdí tam z pohodlnosti nebo z prestiže.

Příklad zamlčených předpokladů

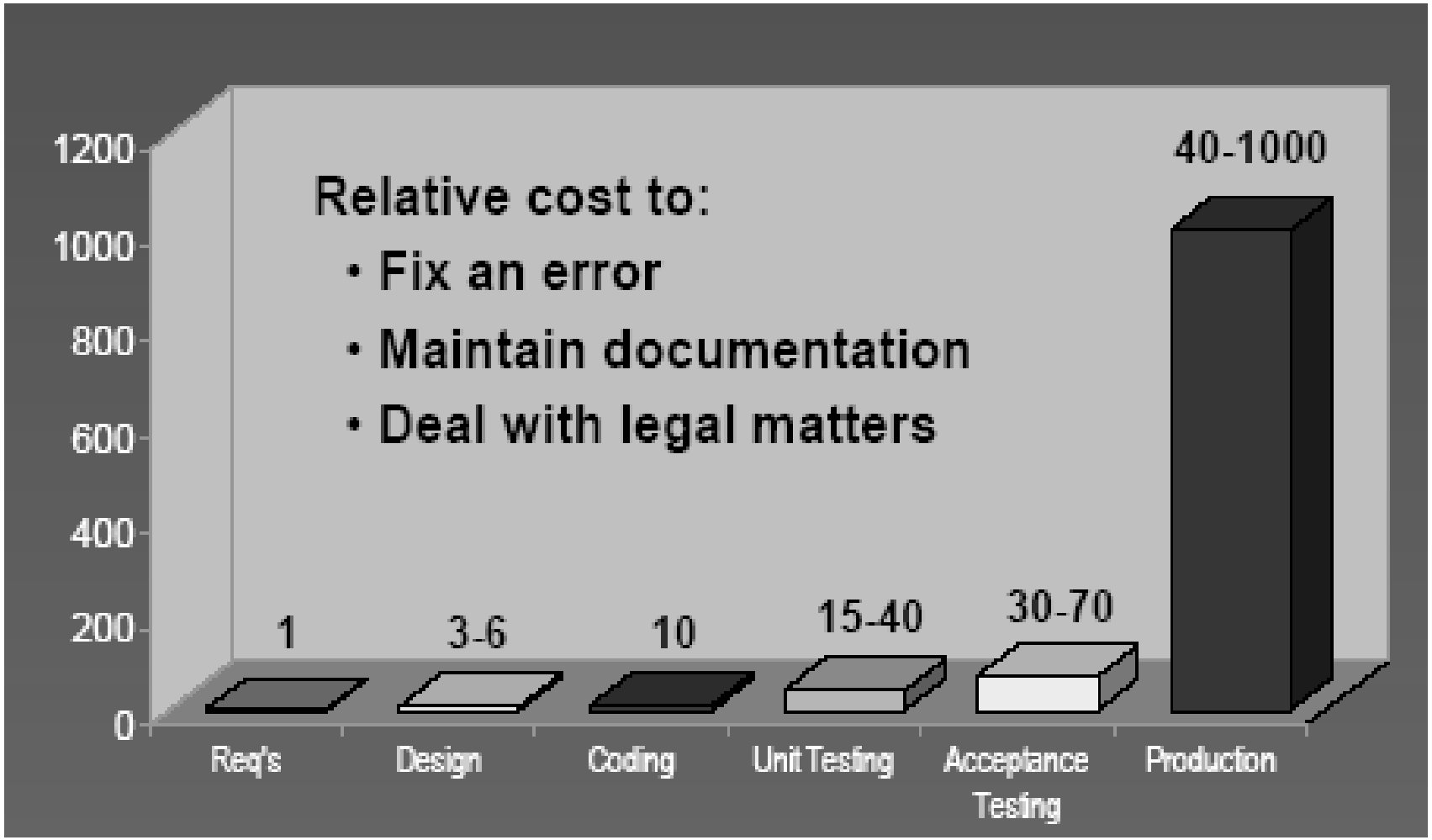
Skrytý základní předpoklad – Zvýším-li kapacitu parkovišť, věc se vyřeší. Předpoklad – problém není v kapacitě tras. To neplatí. Zahltí se ulice. Rozšířené ulice zhorší kvalitu života a opět se zahltí

Řešení: nepřipustit nebo omezit příjezd do centra:

Kodaň: Omezení parkovacích míst v centru

Londýn: Placené propustky

Cena odstranění defektu se zněkolikanásobí na každou etapu, kterou projde (kvocient je tři až pět).



Růst cenu opravy

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

Varianty testování

1. Testy připravuje a provádí programátor
   - Po napsání programů (kromě testů částí se to nedoporučuje)
   - Před napsáním programů (agilní vývoj, vhodné)
2. Testy připravují a provádí testéři (s možnou výjimkou testů částí). Nutné a velkých a u kritických systémů
   - Je žádoucí opakování testů automatizovat. Automatizace testování je podle teorie složitosti (např. test na mrtvý kód, Churchova téze) nereálná
   - Testování černých skříněk
     - Pracné ale účinné
   - Testování bílých skříněk

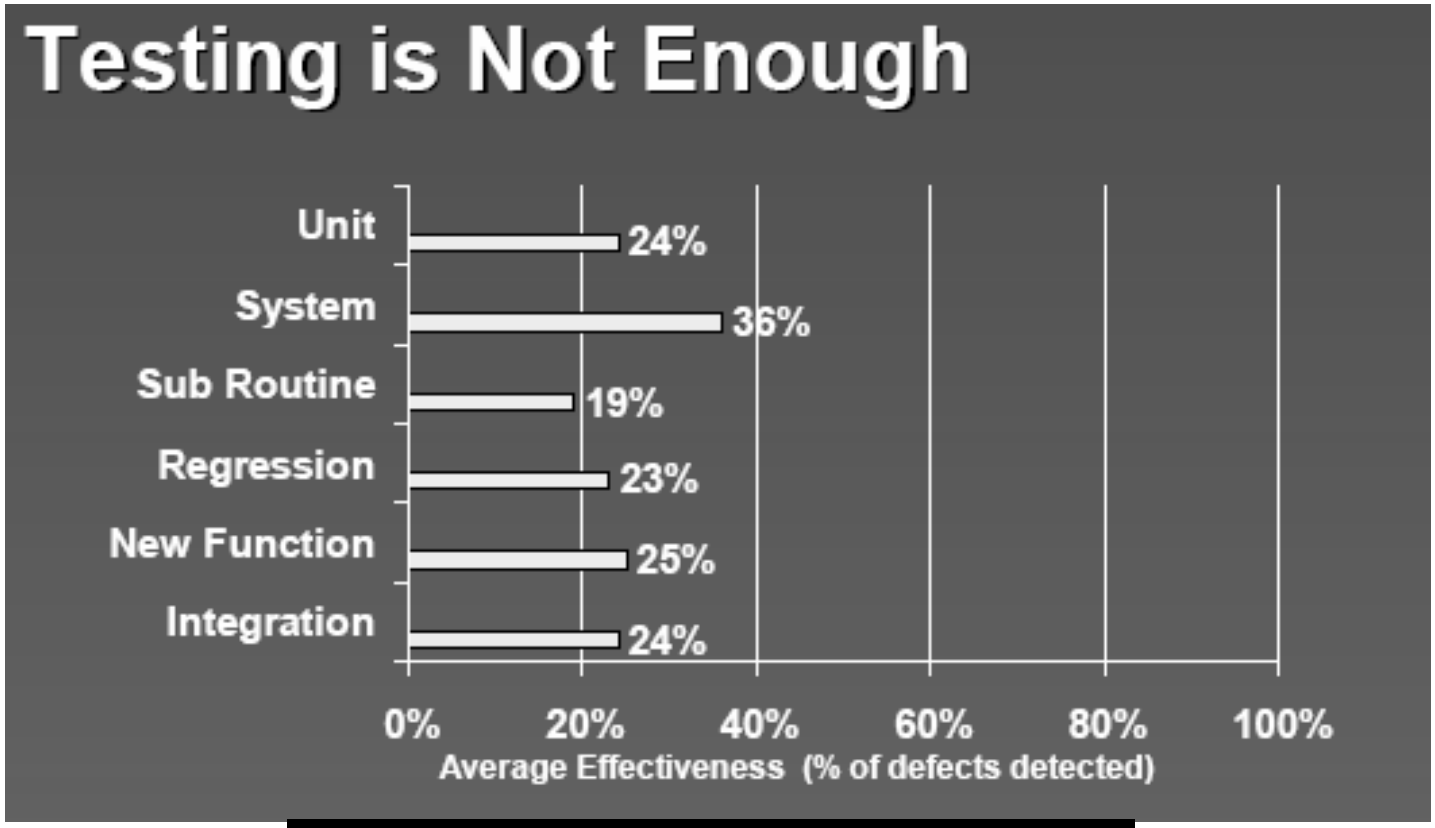
Hlavní problém testů: dokazovat sám sobě, že jsem chyboval – proto testéři.

Být dobrým testérem vyžaduje specifický talent.

Testování černých skříněk je důkladnější, ale pracnější

- Je zaměřeno jednoznačně na rozhraní
- Nevadí osobní vazby mezi kódery a testéry,
  - není tendence krýt kamarády
  - Nepřebírají se myšlenkové chyby vývojářů
- Testéři nemají tendenci navrhovat opravy což podle zkušeností zhoršuje jejich kapacitu detekovat chyby

Účinnost testování



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

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

Odvození účinnosti testování

- Doplněk jevu *J*, že žádný test nezachytí
- Pravděpodobnost *J* = součin pravděpodobností jednotlivých jevů

Praha 18.10.

Trochu terminologie

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

V obou případech obdobné techniky.

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

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

Trochu terminologie

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

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

Inspekce: Přísně formalizované oponentní řízení pro menší dokumenty s řadou činností a rolí.

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

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

Vyžaduje to ale individuální nasazení, disciplinu a koncentraci, které se obtížně kontrolují a ne všichni jsou jich schopni. Špatně se kontroluje

Trochu terminologie

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

Trochu terminologie (norma IEEE std. 1044)

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

Vztahy mezi typy chyb

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

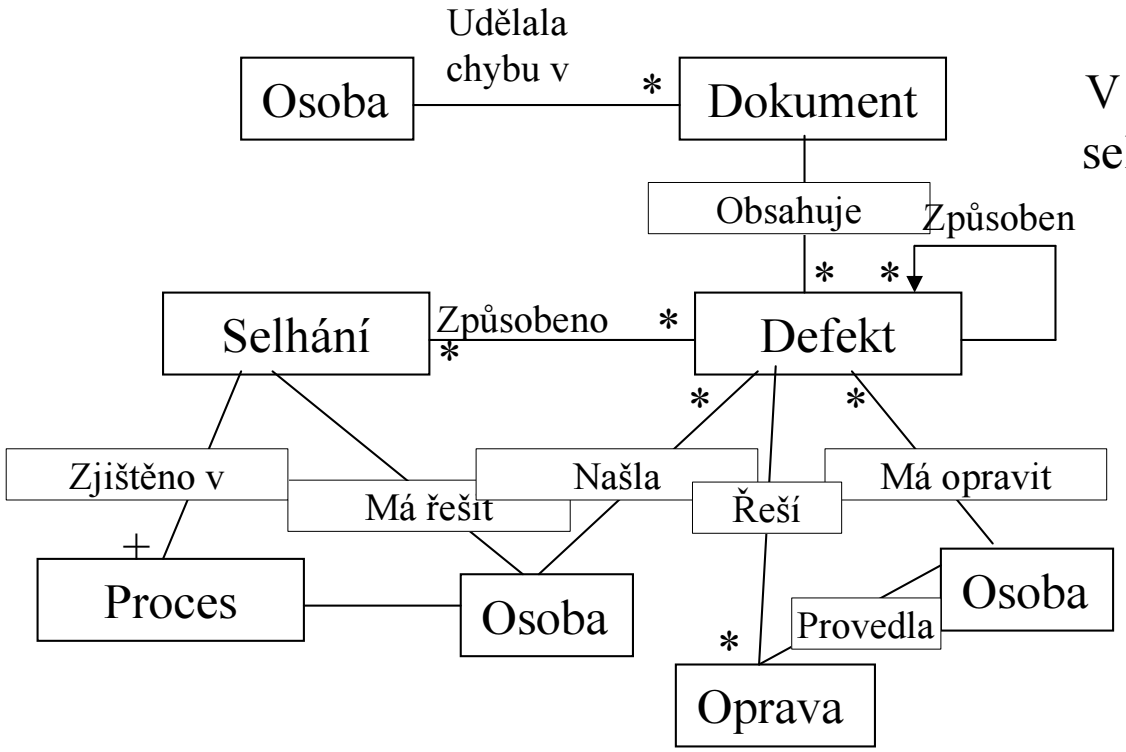
Výstupy verifikací a validací

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

Tyto zásady zvětší účinnost příslušných procesů, mj. tím, že lze vytvořit kvalitní IS pro kontrolu vývoje systému

Konceptuální schéma

Osoba



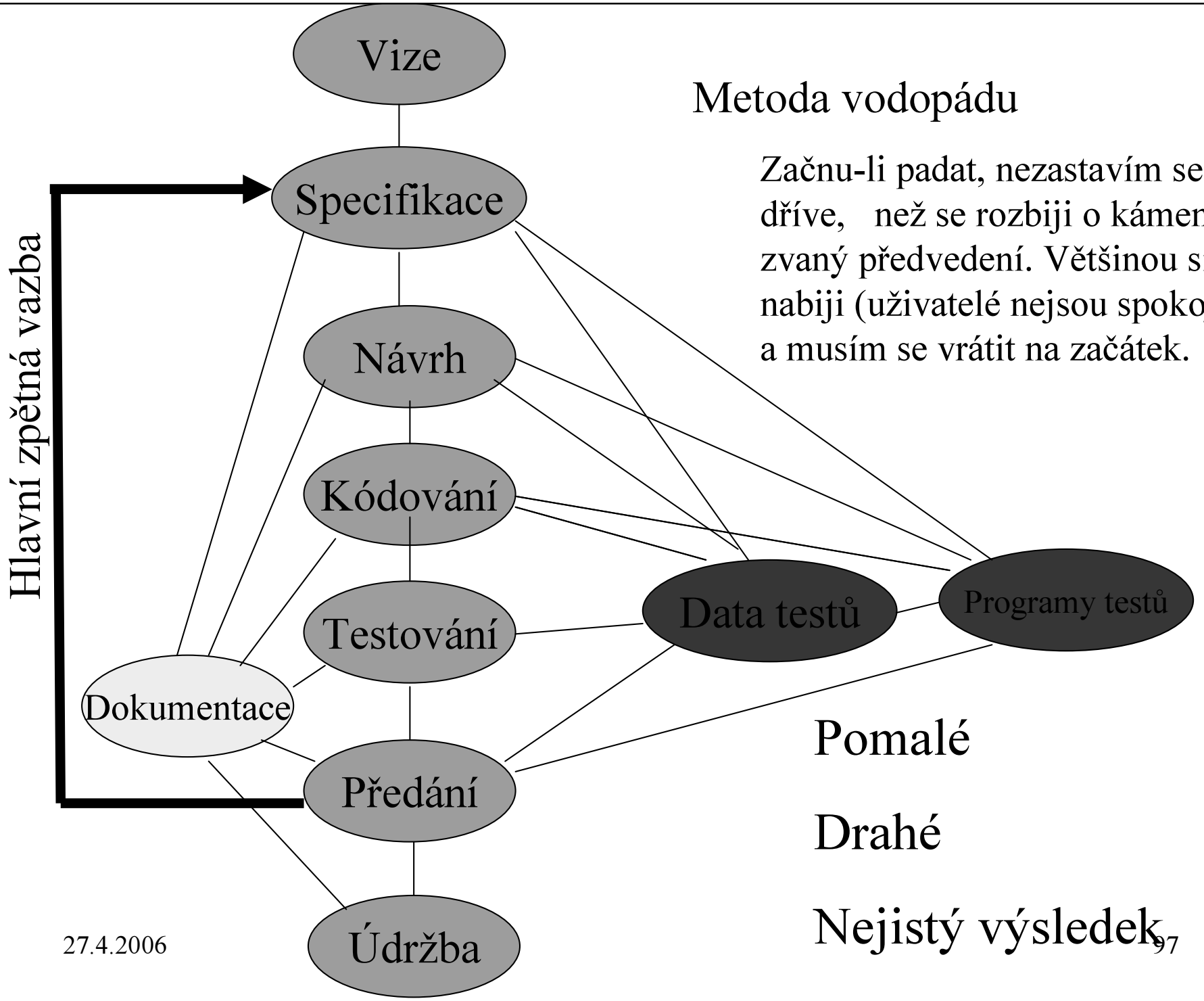
V procesu odstraňování selhání se * změní na +

Proces je oponentura, test, nebo stížnost uživatele

Jak se dá zjistit, kdo udělal opomenutí?

Úkol

Dá se dohledat prapříčina defektu?



Metoda vodopádu

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

Pomalé

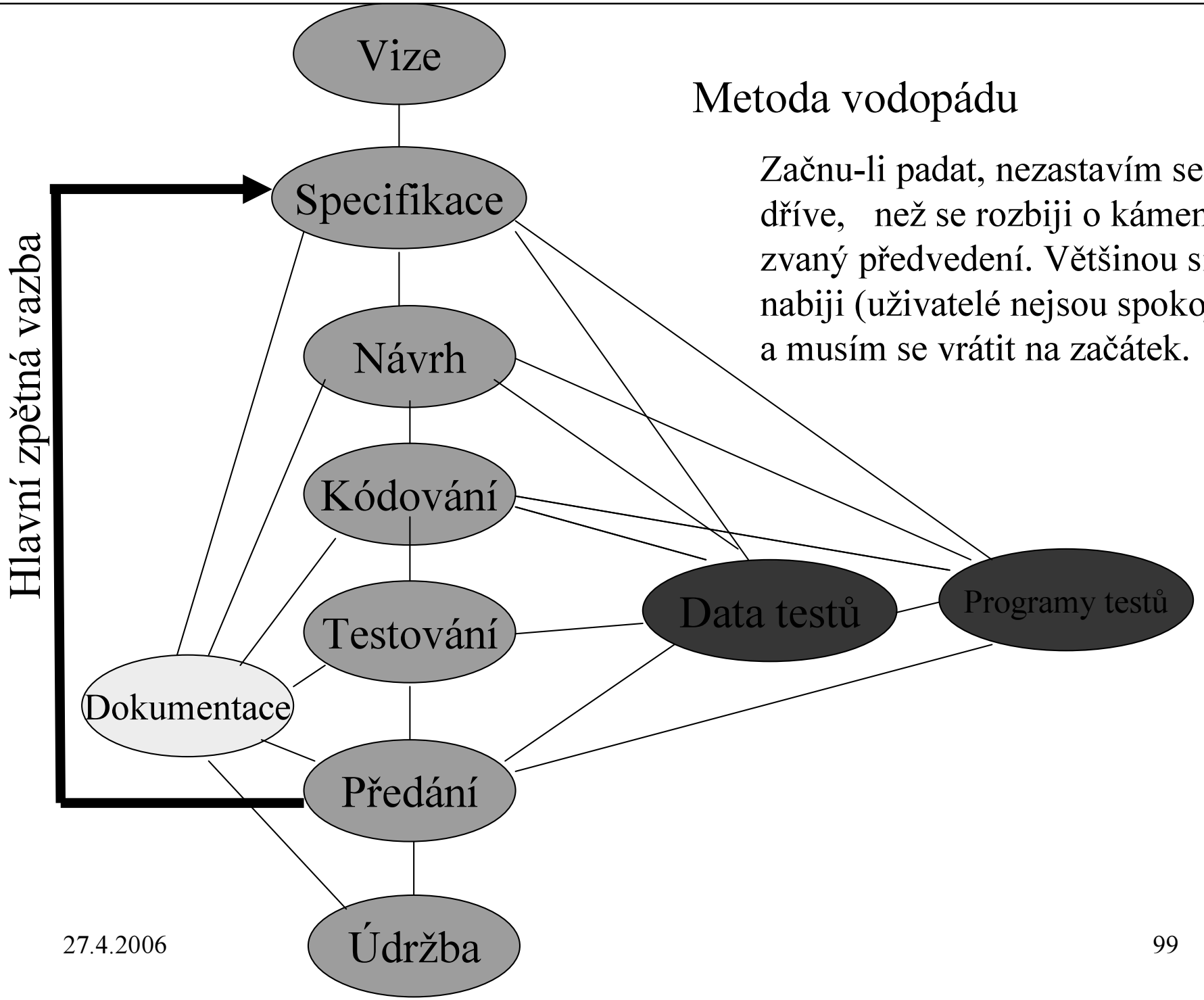
Drahé

Nejistý výsledek

Léčba

Oponentury (verifikace)

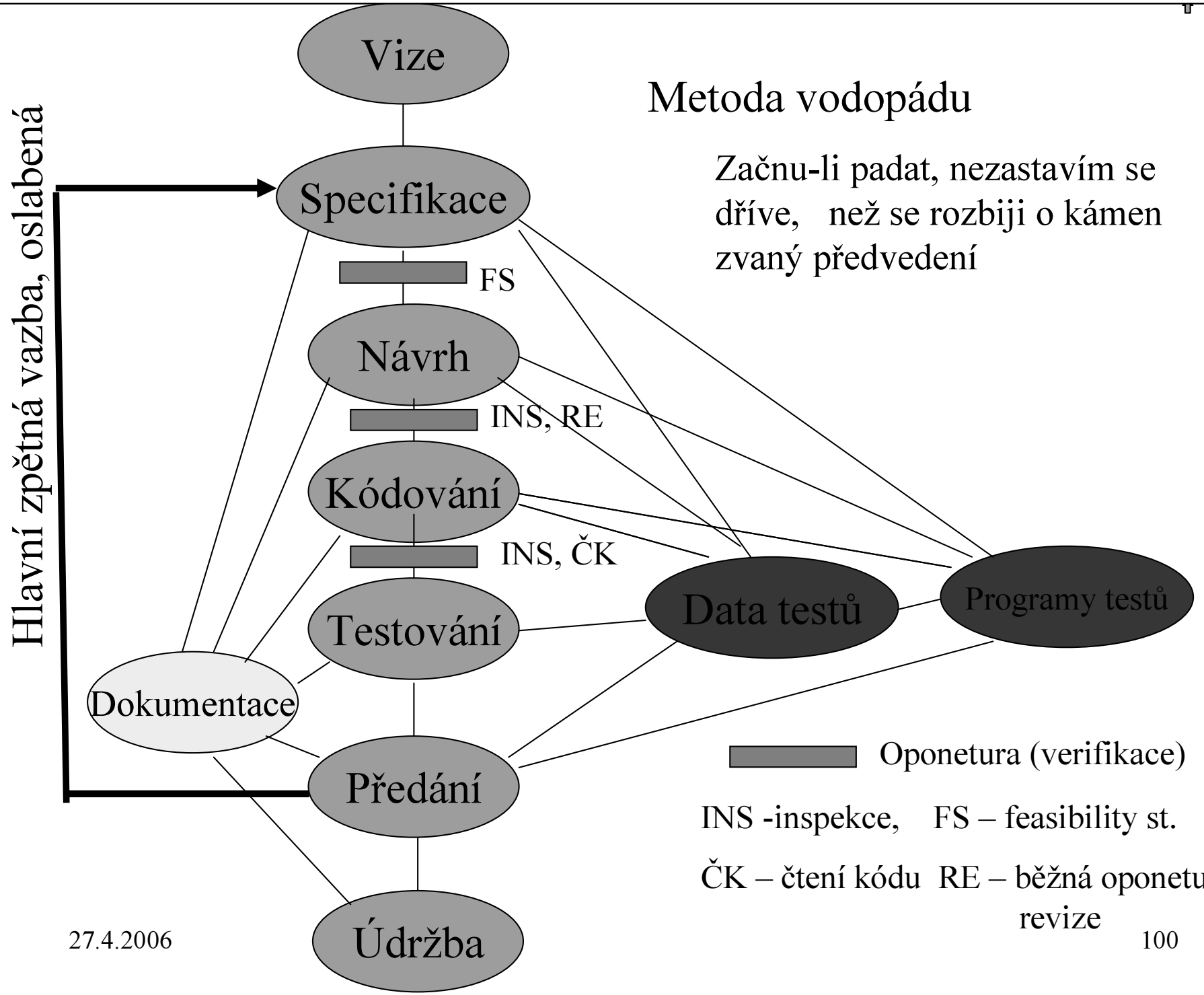
Metoda vodopádu



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

Metoda vodopádu

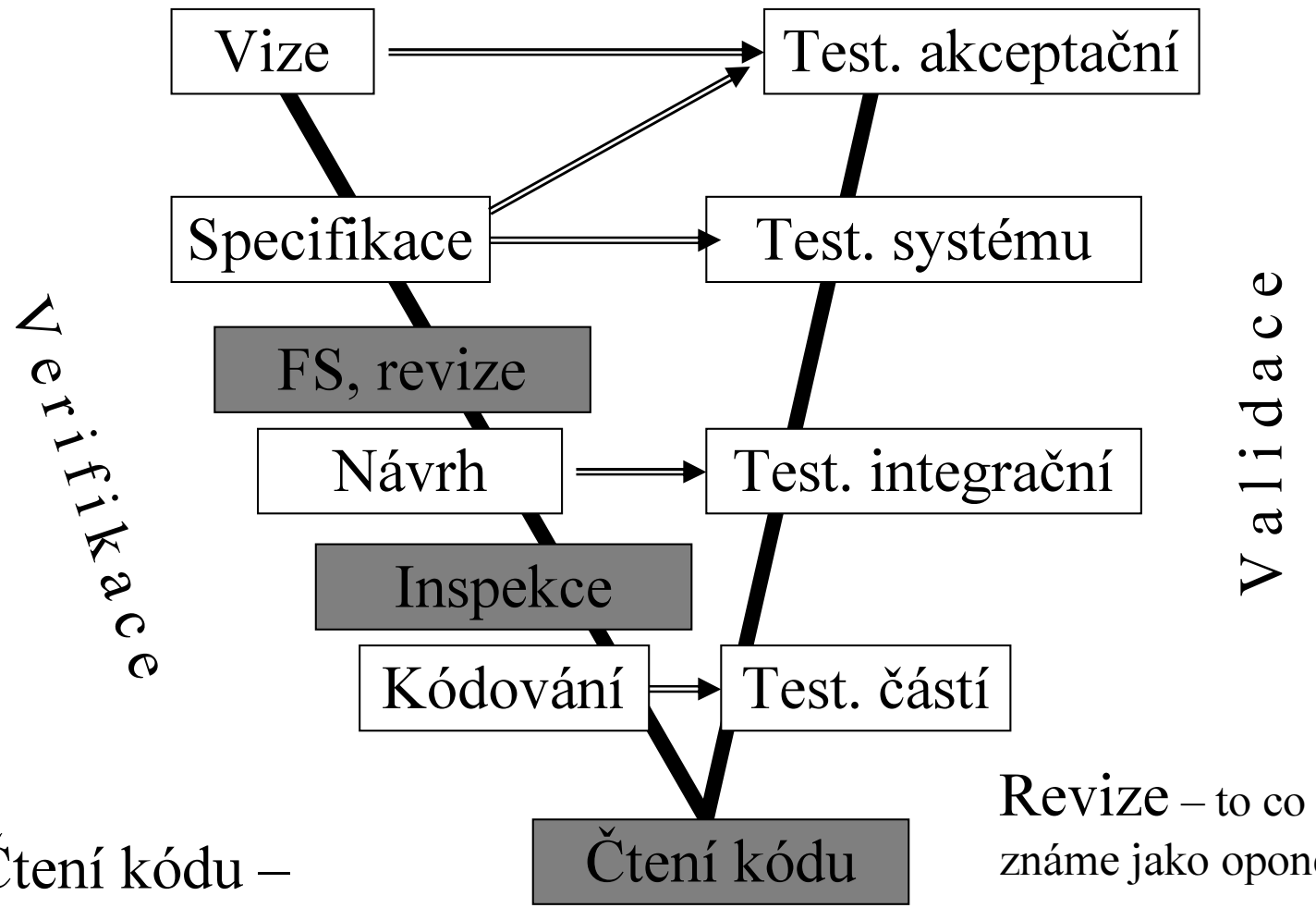
Začnu-li padat, nezastavím se dříve, než se rozbiji o kámen zvaný předvedení



INS -inspekce, FS – feasibility st.

ČK – čtení kódu RE – běžná oponetura, revize

V-schema



Čtení kódu –

procházení textu ve dvou

Revize – to co obvykle známe jako oponenturu

Inspekce – vysoce formalizovaná oponentura

Efekt verifikace

Pokud je dobře provedena je stejně účinná jako testování. Může ale snáze odhalit některé typy chyb.Tvrdí se že účinnost je až 80%, stačí ale i 25%. Účinnost ladění tří etap prováděním verifikací i validací se zvýší až na 85%. To většinou vyhovuje, protože zůstávají skryty defekty, které jen výjimečně vyvolají selhání

1-(0.75)**3* (1-0.24)*(1-0.24)* (1-0.36)=0.84

Problém je v kvalitě provedení verifikace. Verifikace se obtížně kontrolují a závisí na kvalitě nasazení a na pozornosti účastníků sezení.

Další způsoby léčby

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

- Validovat specifikace předvedením SW prototypů (částečně funkčních modelů systému)
- Validovat i prototypy (spirálová metoda)
- Postupně systém jako monolit rozšiřovat (iterativní metoda)
- Systém inkrementálně rozšiřovat integrací autonomních aplikací (inkrementální vývoj).

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

Další způsoby léčby

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

Začni programovat co nejpozději!

Použij agilní metody!!

Agile development prefers:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

Architektura

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

Architektura, k čemu

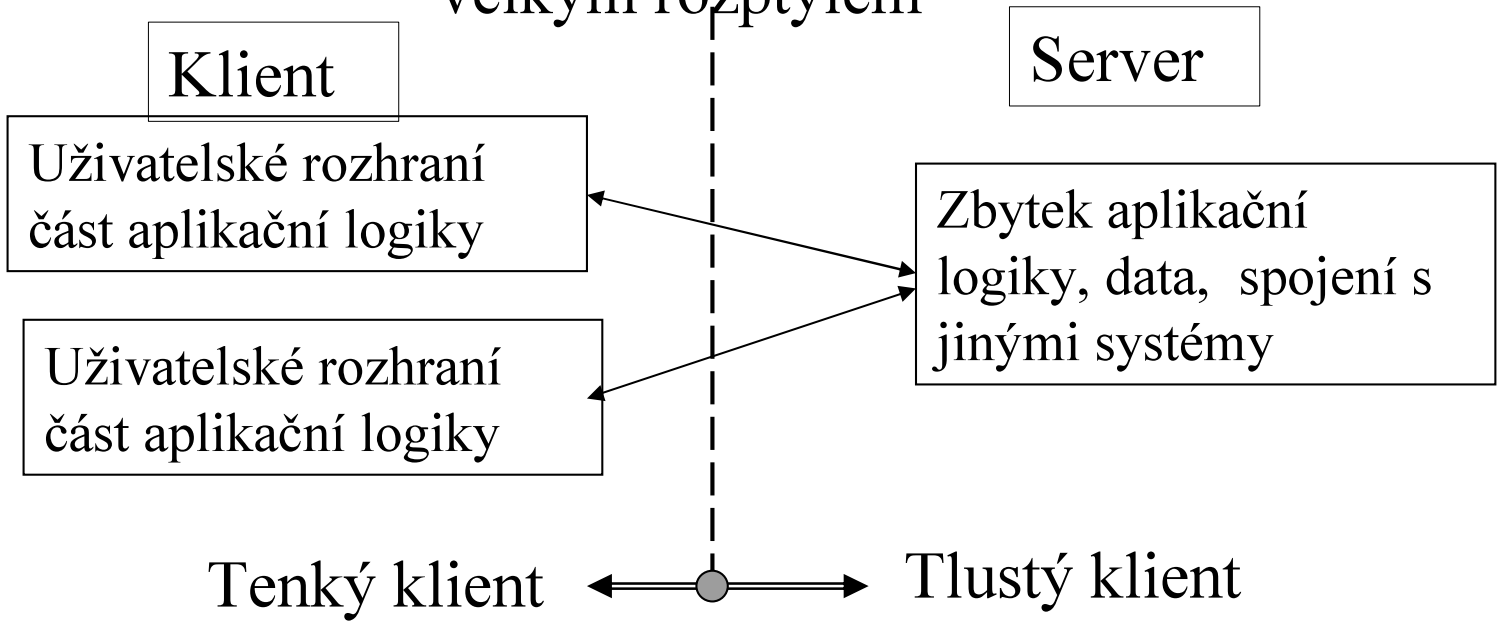
- Celková koncepce a návrh, jednotící idea, porozumění celku
- Usnadňuje zobrazení celku (dílenský výkres)
- Většinou prostředek dekompozice, porozumění, analýzy (např. konzistentnost a kvalita návrhu)
- Důležité pro
  - Zvládnutí vývoje
  - Znovupoužitelnost částí
  - Údržbu
  - Dostupnost některých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

Architektura klient – server

Společná správa dat a aplikací,

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

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



Výhody tenkého kleinta

- Úspory nákladů na hardware klientů
- Snazší správa
  - Bezpečnost
  - Méně záplat
  - Méně příležitostí k flákání hraním her
  - Lepší kontrola práce (nesmí se ale přehánět)

Úspora výkonu

- *Zátěž klienta se chová jako náhodná veličina s velmi malou střední hodnotou M a poměrně velkým rozptylem D.* Pokud klient málo využívá server (nebo je to samostatná aplikace) musí mít výkon zajišťující, že se jeho kapacita překročí jen v 1% případů. Výkon klienta by tedy neměl být menší než $M+3\sqrt{D}$ *(hranice konfidenčního intervalu)*

Úspora výkonu

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

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

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

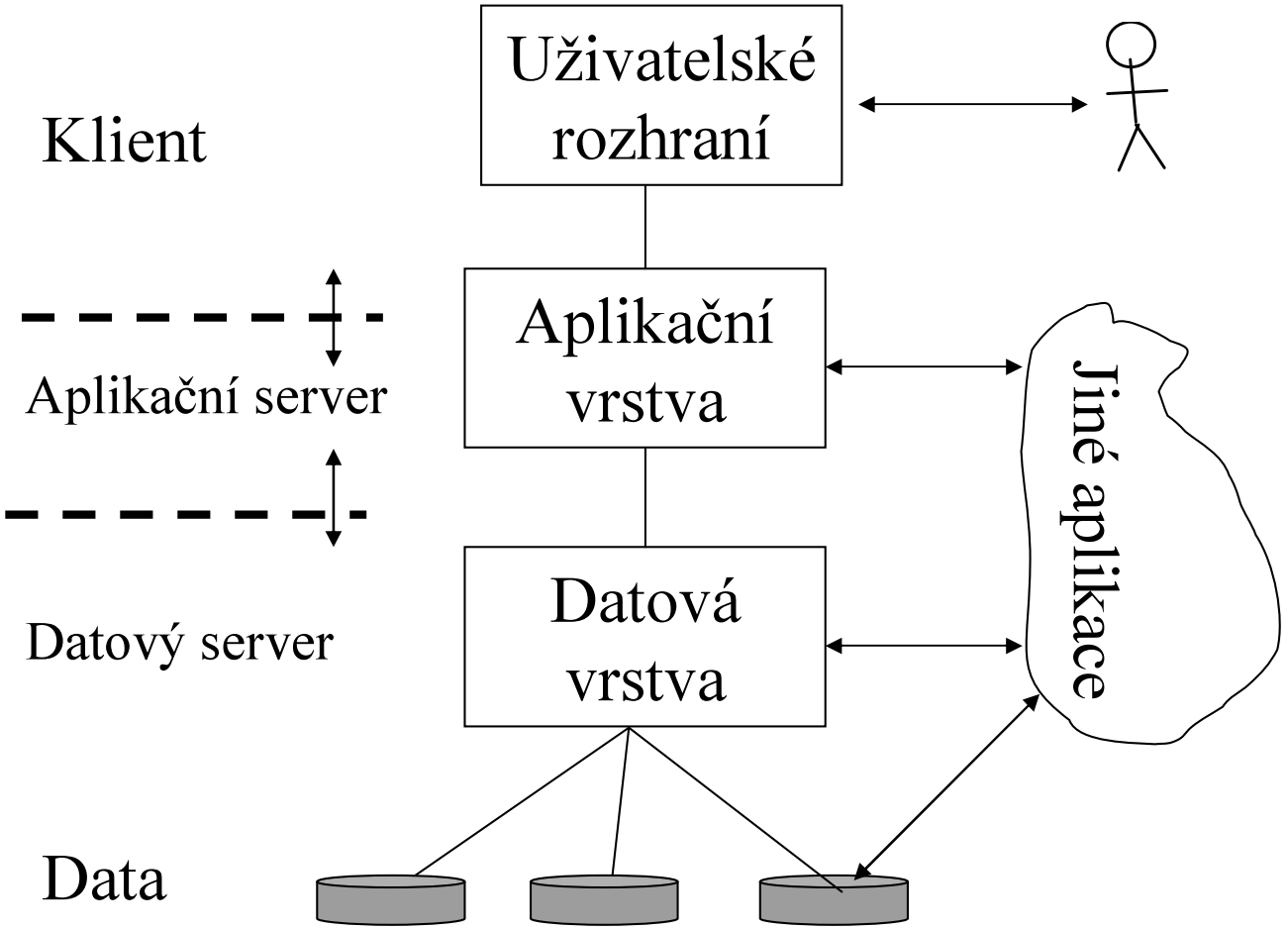
Úspora výkonu

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

Nevýhody tenkého klienta

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

Třívrstvá architektura



Vliv objektové orientace

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

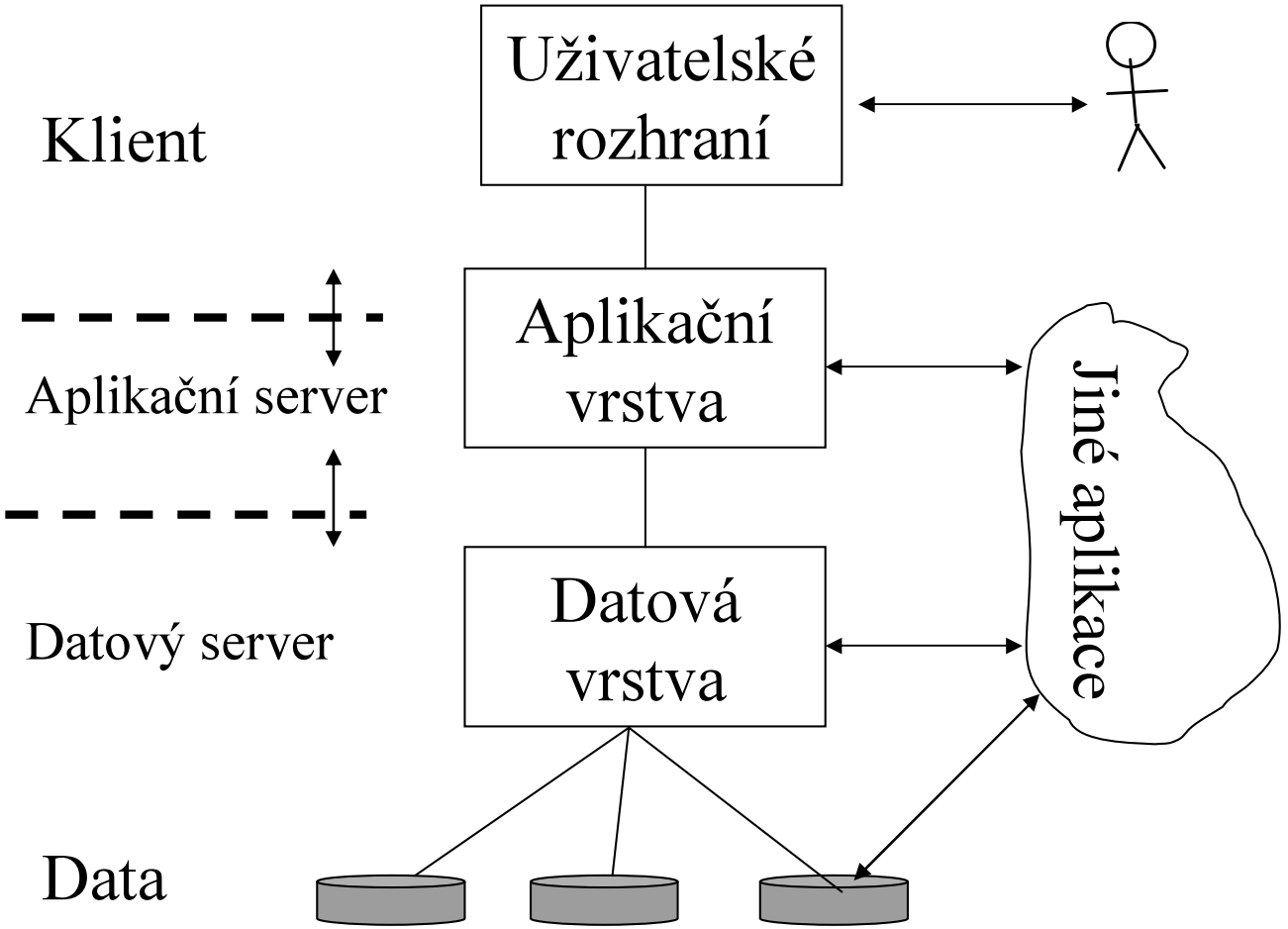
Spoje s jinými systémy, diskuse variant

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

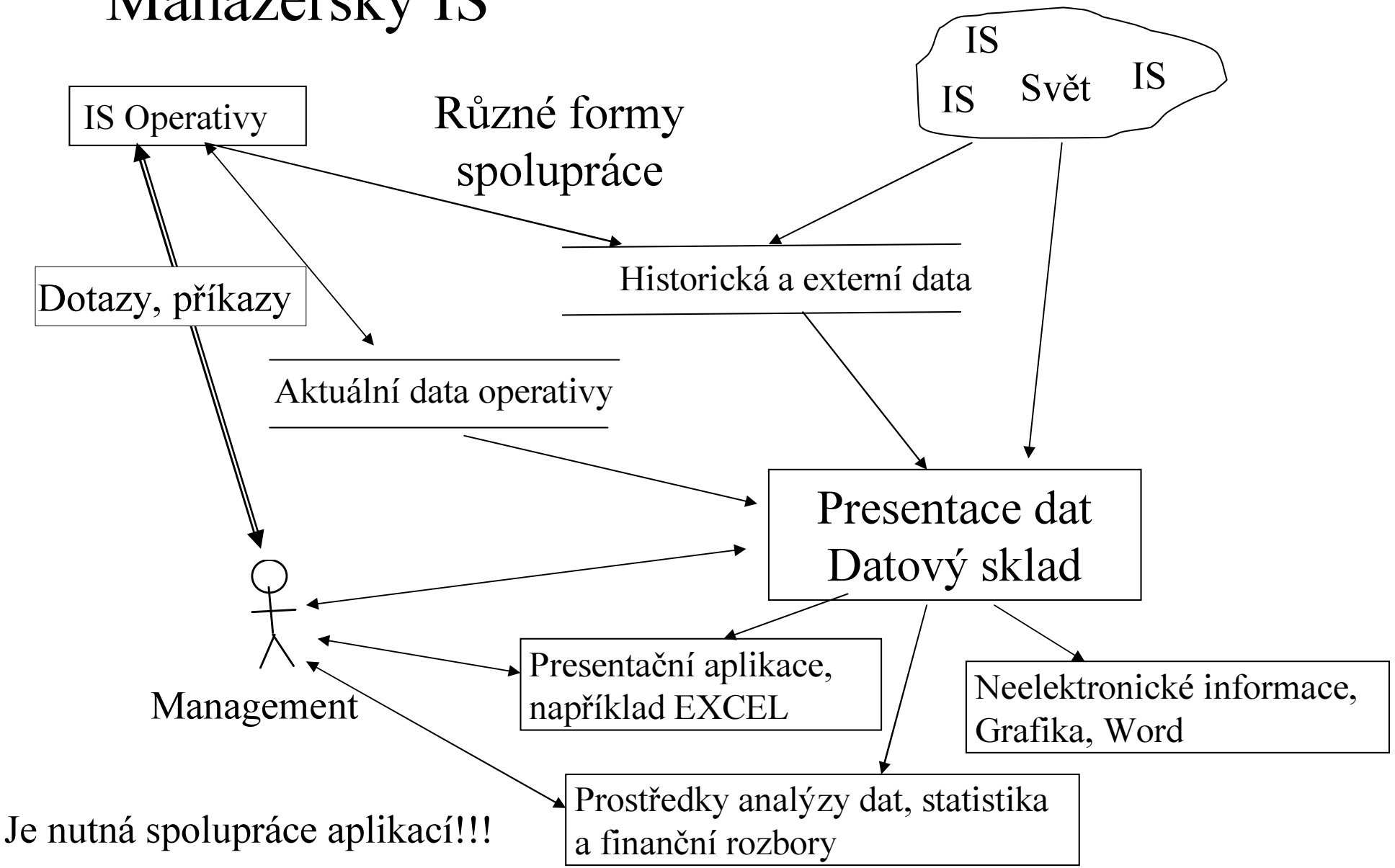
Přístup k datům customizovaných systémů

- Velké systémy, jako je R/3 od SAP nebo Oracle financial mají v podstatě třívrstvou architekturu. Data jsou uložena ve standardních databázích
- Je výhodné si sjednat přístup k datům pro čtení pro řešení ad-hoc potřeb managementu. To co se potřebuje často zobecnit a dát implementovat dodavateli
  - Ten takový postup z obchodních důvodů nemiluje

Třívrstvá architektura



Manažerský IS



Je nutná spolupráce aplikací!!!

Formy spolupráce podle rychlosti

1. Dávka
2. Datové úložiště
3. Aktivní DB
4. Přímý přenos zpráv
   - Přímý přenos zpráv může být „málo inteligentní“

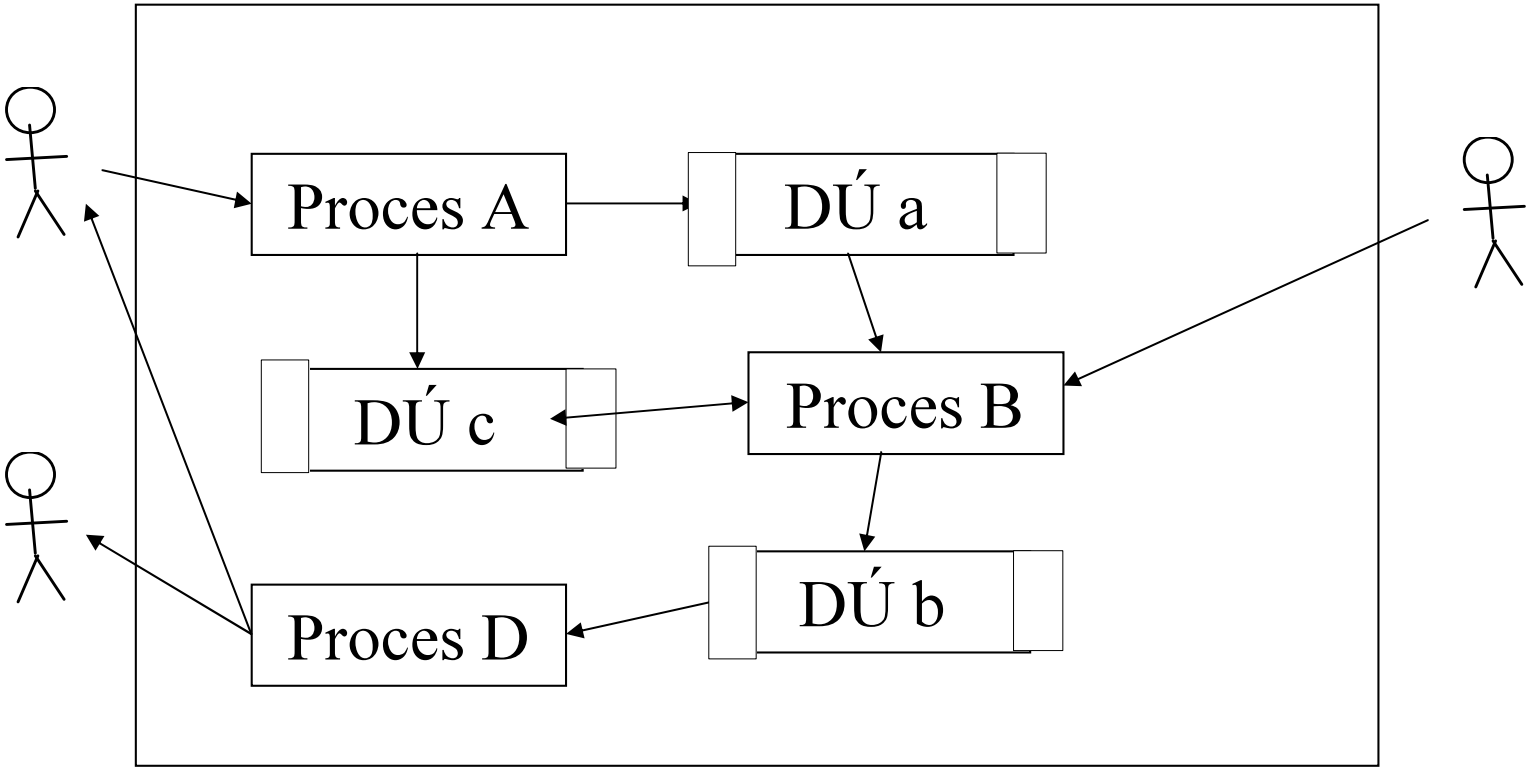
Pozorování

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

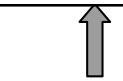
Pozorování

- Obchodní procesy závisí skrytě na kulturních zvláštnostech
  - U nás na rozdíl od USA se ponechává prostor pro změny a intuici
  - Vyžaduje to ale jiné vzdělání (učňáky)
  - Obě řešení mají své výhody a nevýhody.

Offline komunikace: data-flow diagram



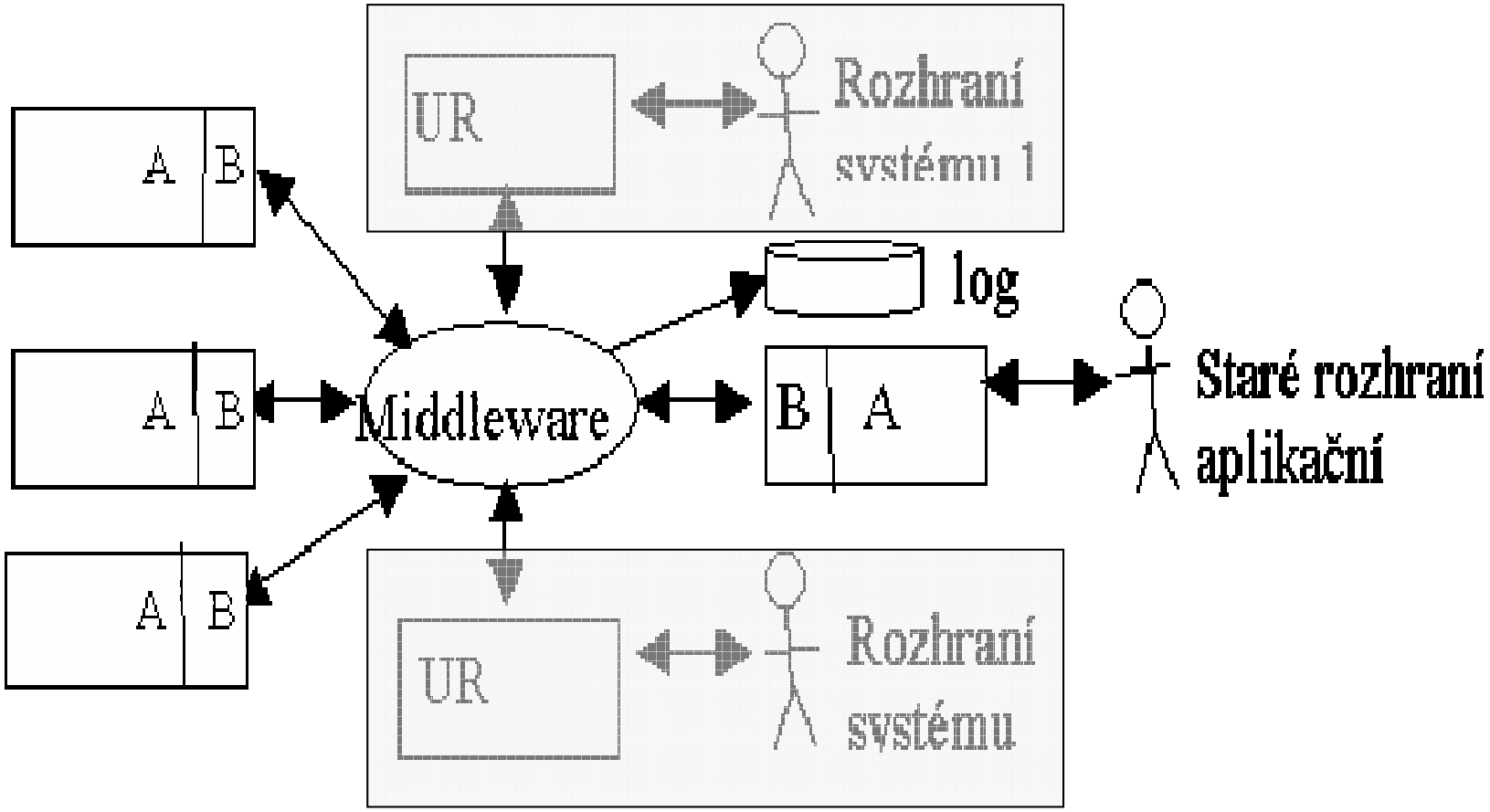
Autonomie procesů Zavrhováno v OO



Varianty vývoje

- Případ Y2K ukázal velkou stabilitu DFD
  - Systémy bez údržby po mnoho let, takže nebyli k dispozici žádní programátoři schopní program upravit
- Typické pro dávkové zpracování
- Snadný autonomní vývoj
- Procesy se chovají jako peers ve virtuální síti

Architektura aliance a konfederace



A – aplikační služba, B – její rozhraní (primární brána)
UR je uživatelské rozhraní, např. portál

Staré aplikační rozhraní je u komplexnějších služeb nutností (viz IS úřadů), usnadňuje podstatně vývoj.

XML a uživatelské rozhraní