

INFORMAČNÍ SYSTÉMY

Společná část, okruh 1

(vypracovala „Vev“ Lenka Horáčková,
materiály vyhledal „Kumpa“ Marek Kumpošt,
obrázky většinou spáchala „Pucík“ Lucinka Leistnerová)

Seznam materiálů a předmětů:

ANANAS – skripta

OO IS = Obj. metody návrhu IS – skripta

VTP = Vedení týmového projektu - skripta

TIS = Technologie IS I, Technologie IS II – přednášky

kniha: Jaroslav Král, Informační systémy – specifikace, realizace, provoz. (největší zdroj)

1. Modely životního cyklu SW

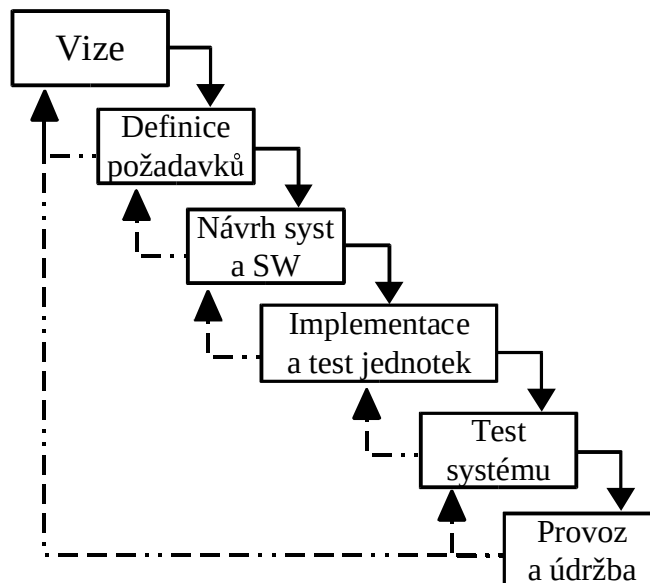
(kde: kniha od str. 97, TIS II, ANANAS 1. přednáška)

Vodopád (klasický)

Problémy:

- chyby, ke kterým je potřeba se vracet – zvyšují se náklady

- uživatelé nedokáží hned na začátku formulovat požadavky jasně a zřetelně

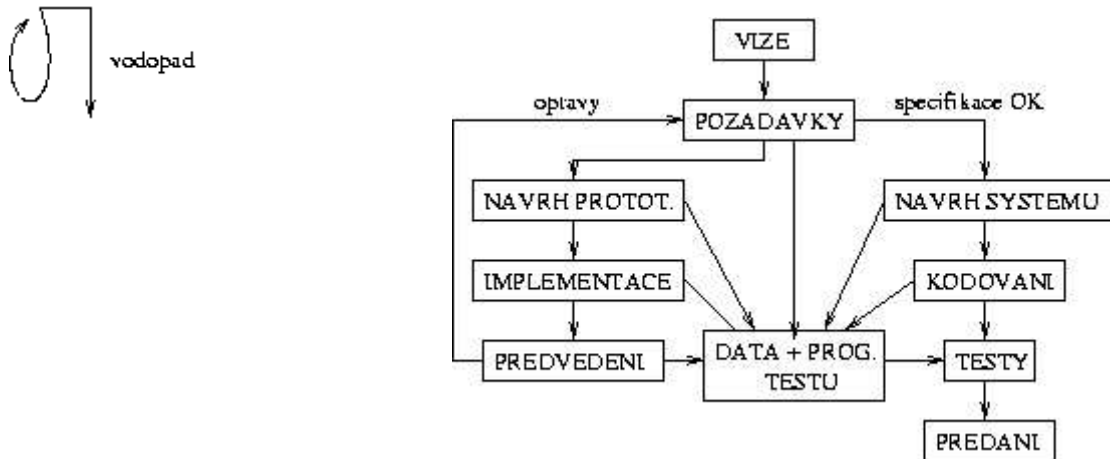


Prototypování:

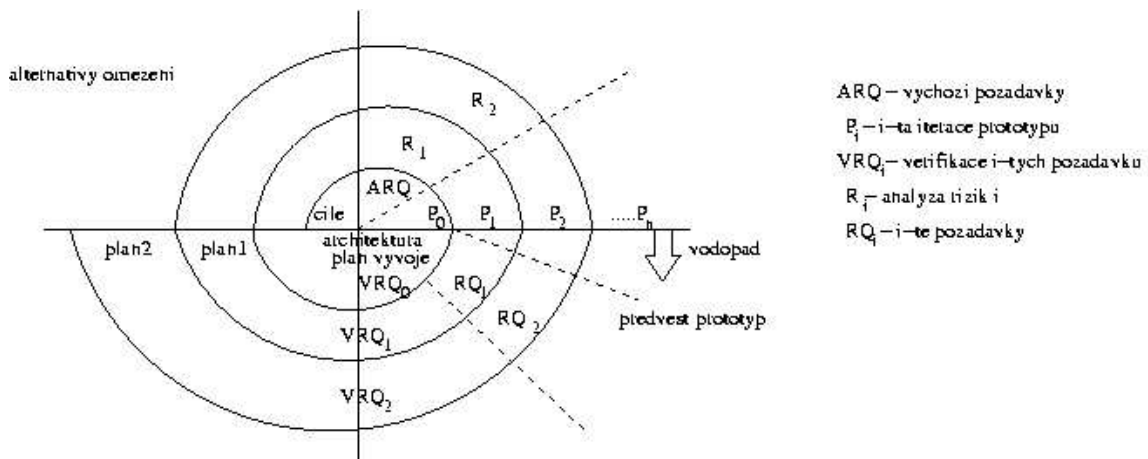
Prototyp – částečně funkční model systému, který realizuje nebo simuluje některé vlastnosti systému. Systém se pak vyvíjí postupně rozšiřováním jistého jádra o relativně samostatně vyvíjené přírůstky (inkrementy), nebo zpřehňováním a doplňováním fcí (iterativní vývoj).

Vývoj s prototypy:

- opakované zdokonalení prototypu + vodopád



- spirálový model



- součástí každého cyklu je analýza rizik
- zpřesňování požadavků, verifikace požadavků, plán akcí, analýza rizik, návrh a realizace prototypů, předvedení a analýza funkcí prototypů, ...
- vhodné pro IS, kde je značná míra nejistoty ve stanovení požadavků a pro vývoj od počátku
- vhodné pro velké systémy

- iterační model

- výsledkem každého cyklu je větší a lépe fungující část cílového systému.



Výhody:

- umožňuje dříve dospět k SW (po částech)
- lze vyvíjet i v menším týmu
- vede k modif. a rozšíř. Systému
- snadněji se integrují produkty třetích stran a existující aplikace

Nevýhody:

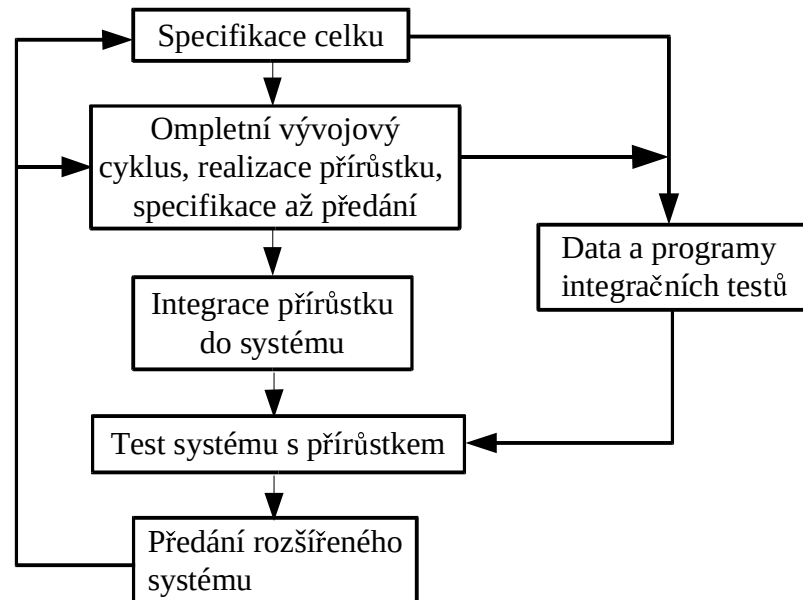
- celý systém až za delší dobu
- u některých systémů lze jen těžko použít
- obtížně se zkracují termíny realizace

• **inkrementální vývoj**

- postupně se z jádra buduje, rozšíření o přírůstky
- vhodný i pro vývoj velmi rozsáhlých systémů
- každý přírůstek prochází komplet vývojovým cyklem

Výhody:

- ve více týmech
- existující aplikace + třetí strany
- snadno lze modifikovat, modernizovat



2. Ná vaznosti a produkty jednotlivých etap

(kde: kniha od str. 84, je toho ale docela málo, možná zkusit pohledat jinde :-()

- každá etapa ukončena vnitřní oponenturou. vnitřní oponentura se většinou týká dokumentů právě ukončené etapy
- realizace se uskutečňuje jako řada etap. Při zjištění problému v etapě n se musíme vrátit k některé z předchozích etap.

- správnost etapy n se prověřuje vzhledem k etapě $n-1$ a $n+1$
- každé etapě odpovídá vlastní dokumentace
- někdy je nutné nechat některé relativně samostatné části projektu otevřené
- při testech často jen intuice
- důležité je, abychom se na vyšších úrovních nemuseli zabývat nižšími úrovněmi (detaily)
- vystopovatelnost požadavků

3. Aplikace CASE v životním cyklu

(kde: kniha od str. 297)

Použití: při specifikaci požadavků, návrhu, kódování a údržbě

Z hlediska životního cyklu SW se nástroje CASE dělí do následujících skupin:

1. Horní (upper) CASE

- použití pro formulaci cílů projektu, analýze a specifikaci požadavků. Nástroje pro plánování a vedení projektů, procesní inženýrství.
- hl. úkolem je analýza organizace, v níž se má systém používat, zobrazení procesů v organizaci, definice klíčových informačních toků, atp.

Hlavní nástroje:

- diagramy toků dat a jejich varianty
- ER diagramy bez podrobné specifikace všech atributů
- prostředky pro řízení projektů

2. Střední (middle) CASE

- použití pro podrobné specifikace požadavků, návrh systému, dokumentace a vizualizace systému
- formalizace specifikace a návrhu s cílem usnadnění změn a snazší komunikace se zákazníky
- vytvoření modelů usnadňujících, případně umožňujících generaci návrhu
- je jádrem komerčně dodávaných CASE systémů
- především u větších firem
- zahrnují prostředky pro podrobnou specifikaci požadavků a návrh systému
- nejúspěšnější, neboť pokrývá činnosti, kt. nejsou předmětem činnosti konzultačních firem, a firem zabývajících se tvorbou modelů podniků a řízením projektů
- nejsou dosud ve větší míře integrovány do moderních prostředí pro vývoj SW

Hlavní nástroje:

- diagramy toků dat včetně možnosti podrobnějšího popisu procesů, dat. úložišť a datových toků
- ER diagramy s možností detailní specifikace atributů,
- pro objektovou metodologii diagramy OO technik – diagramy tříd a jejich vztahů, diagramy instancí, přechodové dia., atp.
- systém správy dokumentů a správy konfigurace

- systémy vyhodnocování metrik souvisejících s návrhem systému a specifikacemi pož.
- vývoj prototypů, většinou potěmkinovských, návrh rozhraní, generátory obrazovek a sestav
- generátory definic dat. Někdy pouze kostry definic, někdy se doplňují ručně.

3. Dolní (lower) CASE

- obsahují nástroje na podporu kódování, testování a údržby a reverzního inženýrství

Hlavní nástroje:

- generátory kódu
- prostředky reverse engineering. Nástroje umožňující rekonstrukci dokumentace z existujícího SW nebo alespoň detekci míst, kde již existující dokumentace neodpovídá aktuálnímu stavu.
- prostředky sledování a vyhodnocování metrik kódu
- prostředky a nástroje plánování a zajišťování kvality SW (sběr info o průběhu testování, řízení testování, sběr a vyhodnocení dat, inspekce a výsledků testů, pravidla přijímání prvků konfigurace, podpora plánování opatření na zajišťování kvality)
- správa konfigurace
- prostředky sledování a vyhodnocování práce systému

(zkušenosti s CASE a volba CASE nástrojů dále na přečtení z knihy od str. 299)

4. Specifikace požadavků

(kde: kniha od str. 87, TIS II, ANANAS 2. přednáška)

Odpovídá na otázku CO má být realizováno. Chyby v této etapě mají fatální následky. Je potřeba zajistit správné a úplné zjištění požadavků na systém.

Typy:

1. **Interview:** dobře připravený pohovor o tom, co pracovník uživatele dělá, co by mohl IS zlepšit či přinést.
2. **Strukturované interview:** interview, kde se postupně odpovídá na otázky dle předem připraveného dotazníku.
3. **Dotazníky:** rozesílají se, uživatelé vyplní sami. Levné, ale nezjistí se vše.
4. **Studium dokumentů:** používaných zákazníkem
5. **Společ. vývoj požadavků:** formulace požadavků skupinou pracovníků uživatele a konzultantů dodavatele IS
6. **Pozorování chodu prací u zákazníka:** drahé, zdlouhavé, ruší při práci.
7. **Účast na prac. procesu:** totéž co výše, někdy se neodchytí výjimečné případy
8. **Analýza existujícího IS:** nebezpečí převzetí chyb původního IS

(vše je velmi podrobně popsáno v knize i TIS II, nejdůležitější je asi umět interview)

5. Prototypy a oponentury

(kde: kniha od str. 97 (prototypy), 103 (oponentury), TIS II,

PROTOTYPY

SW prototyp = částečně funkční model systému, který realizuje či simuluje některé vlastnosti systému.

Prototypování umožňuje odhalit nedostatky již v začátku projektu.

Důvody pro vytvoření prototypů:

- ověření správnosti a úplnosti specifikace požadavků
- ověření správnosti a úplnosti funkcí a návrhu struktury systému
- předběžný odhad nákladů a rizik realizace

Typy prototypů:

- a) **Potěmkin:** model cílového systému, který simuluje uživatelské rozhraní, tedy obrazovky dialogů a tvar tiskových sestav. Výkonná část systému téměř či úplně chybí.
- b) **Neúplný:** modeluje pouze některé funkce
- c) **Jiný kůň:** téměř úplný, ale funguje na jiném HW nebo nad jiným základním SW
- d) **Hlemýžď:** neefektivní, pomalý
- e) **Nepříjemný:** má nepříjemné uživ. Rozhraní, je nestabilní
- f) **Lajdák:** nereaguje správně na chyby v datech a chyby obsluhy

Prototyp není určen k cílovému řešení, pouze k ověření specifikace funkcí, zpřesnění požadavků. Bohužel zřídka otestuje vlastnosti, jež se ukážou až v provozu.

Výhoda – poměrně brzy se testují fce programů

Nevýhoda – dělat prototypy je práce navíc

OPONENTURY

- nejefektivnější detekce anomálií
- snížení rizika neúspěchu
- ale pracné

Oponentury jsou **vnitřní (dohled)** a **vnější (audit)**.

Dohled: kontrolní činnost prováděná vedením firmy formou kontrolních dnů. Prověrka skutečností důležitých z manažerského hlediska. (Další dělení viz níže.)

Audit: dohled prováděný nezávislou organizací. Prověřuje dodržování podmínek smlouvy, funkce systému. Provádí auditor.

Dokonce i etapa **specifikace požadavků** by měla být zakončena oponenturou. Kontroluje se splnitelnost požadavků, dodržení záměrů cílů projektu, návrh a zhodnocení plánu dalšího postupu, správnost požadavků, bezespornost, atd.

Výstupní dokument je **Feasibility study = studie splnitelnosti**. Její součástí mohou být výstupy částí vnitřních oponentur. Nejpozději se vypracovává před zahájením kódování, nejdříve však po dokončení spec. požadavků.

Vnitřní oponentury:

- kontrolní akce prováděné členy řešitelského týmu
- druhy oponentur se liší úrovní formalizace a počtem účastníků.

Společné rysy:

- účastní se řešitelé, možno i spoluřešitelé ze strany budoucích uživatelů
- cílem je detekce chyb, chyby se neodstraňují, jen zaznamenávají
- detekce chyb nesmí být důvodem postihu jejich původců
- neměly by trvat déle než 2 hodiny

Typy oponentur:

INSPEKCE

- oponentury prováděné podle přesných pravidel v 1 nebo více fázích ve skupině

a) Jednofázové inspekce

Tým (3-6 lidí), vedoucí (moderátor), 1-3 oponenti, předčítatel a zapisovatel.

- pročítá se specifikace části nebo dokument nebo program, ostatní hledají chyby
- materiály mají několik dní předem, o problémech se dělá zápis
- méně než 2 hodiny, řídí moderátor, nemá se toho účastnit vedení
- problémy se neřeší, jen detekují (výjimka - uživatelská dokumentace)
- inspekce od celku k částem
- odhalení až 80 % chyb

Etapy inspekce:

- plánování (moderátor), příprava materiálů, výběr členů, termín,...
- úvodní studium (účastníci se školí, stanoví si role)
- příprava (materiály pár dní dopředu, studium materiálů)
- inspekce (pod vedením moderátora, zjišťují se chyby)
- vypracuje se zápis (data do dbs)
- přepracování (oprava materiálů)
- kontrola (kontrola, zda byly chyby opraveny)

Aktivní inspekce = metoda “zasetých chyb”, sleduje kvalitu inspekce

- vhodné pro kontrolu programů a návrh dat
- zjišťuje se, kolik bylo nalezeno zasetých chyb a skutečných chyb

$$\frac{c}{C} = \frac{z}{Z}$$

c ... počet nalezených chyb

C ... celkový počet dosud neznámých chyb

z ... počet nalezených zasetých chyb

Z ... celkový počet zasetých chyb

b) Vícefázové inspekce

- některé činnosti se při inspekci provádějí separátně

Etapy vícef. inspekce:

1. **provádějí jednotlivci:** kontroly formálních vlastností (ty lze provést v principu počítačem) Například se kontroluje celková struktura dokumentu, mnemotechnika zkratk, identifikátorů, index a úplnost odkazů, programovací standardy, typ org. rozložení, ...
2. **skupinové inspekce:**
 - oponenti dostanou materiály předem spolu s kontrolními otázkami jak co funguje
 - oponenti individuálně pročítají program či dokument a řídí se seznamem otázek a pokyny, co mají sledovat
 - ve skupině se výsledky z výše uvedených bodů jednotlivých respondentů porovnávají, zjistí se nejasnosti
 - provede se oponentura
 - vše se zanesse do zápisu

REVIZE

méně formální než inspekce, lze použít na větší celky

Etapy revize:

1. určí se moderátor, ten si vybere oponenty
2. každý oponent dostane k analýze určitou část op. Materiálu a snaží se nalézt problematická místa
3. provede se vlastní revize (stručně se spec. úkol, uvedou se a prodiskutují zjištěné nedostatky, zhodnocení dodržení plánu práce, lze vypracovat i doporučení, zhodnocení kvality materiálu)
4. výstupem revize je souhrnné hodnocení s přílohami obsahující seznam problémů

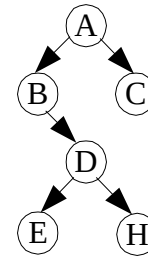
Výhody: větší flexibilita, možnost oponovat rozsáhlejší materiálu, menší nároky na kvalitu členů oponent. týmu.

Nevýhody: menší účinnost, menší možnosti měření kvality provedení.

DALŠÍ TECHNIKY OPONENTUR

- **Ve dvojicích:** dvojice (extrémní programování), vedoucí týmu vs. jeho zástupce. Jeden vymýšlí a píše a druhý kontroluje. Je to efektivní, zlepšují se znalosti, možnost převzetí.
- **Procházení nebo strukt. procházení:** dvojice až trojice, jeden vysvětluje ostatním, často sám je schopen pak chybu odhalit. Podmínkou je dobrý vztah mezi členy, vysoké pracovní nasazení.
- **Simulace textů (čtení kódu):** u programů se simuluje chování programů; prevence, obtíže detekce defektů, dnes již částečně řeší ladící programy, ale neřeší vše.
- **Cleanroom:** formalizovaná metoda, zahrnuje formální důkazy správnosti, při vývoji IS ne příliš efektivní, lépe tam, kde netřeba úzce jednat se zadavateli.

- **Týdenní posezení u kafe:** pravidelné schůzky, neformální diskuse, podpora dobrých vztahů, odchycení vznikajících problémů.
- **Oponentury zdroj. textů programů:**
 - shora/zdola/efektivně
 - systém je hierarchie daná vztahem A potřebuje B: $A \rightarrow B$



- pokud je dobře navržen, bývá to hierarchický strom
- lze se rozhodnout, že začneme od A po směru šípek (shora od kořene), vš. Potomky (do šířky) nebo
- zdola od listů (raději do šířky) nebo
- selektivně shora/zdola se snažíme co nejdříve dostat k oponentuře problematických komponent

ne/výhody:

- shora: oponují/testují v prostředí, které se bude skutečně používat, ale menší obecnost
 - zdola: větší pracnost, obecnost, náročnost na data a předpoklady.
- U oponentur lépe postupovat shora, u testů to není tak jednoznačné.

6. Strukturovaná analýza

(kde: ANANAS 7. až 10. přednáška, OO IS 2. přednáška, kniha 154)

Strukturované metody obecně:

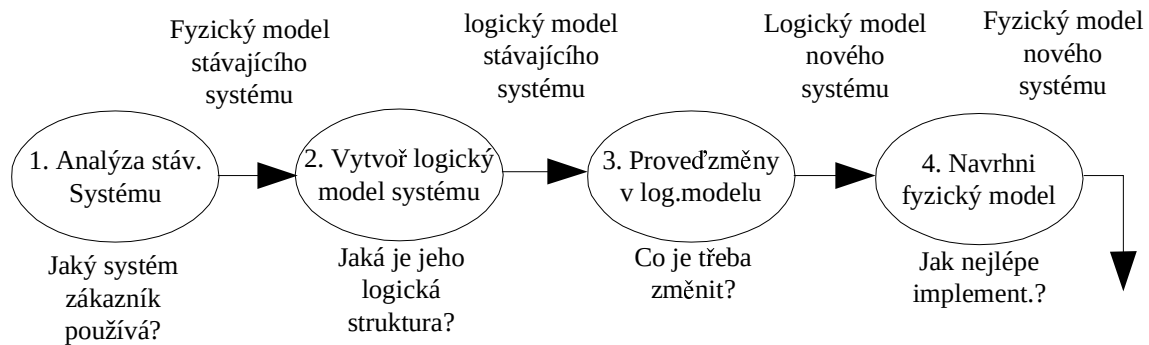
- člení projekt na malé, dobře definované aktivity a určují posloupnost a interakci těchto aktivit
- používají diagramatické a další modelovací techniky, pomocí nichž je tvořena přesnější a úplnější strukturovaná specifikace, které rozumí uživatelé i návrháři
- efektivně využívá zkušené i méně zkušené pracovníky
- k dispozici je třeba mít různé modely systému popisující jednotlivé aspekty jeho činnosti. Slouží ke komunikaci mezi účastníky vývoje systému.

Ověřené modely:

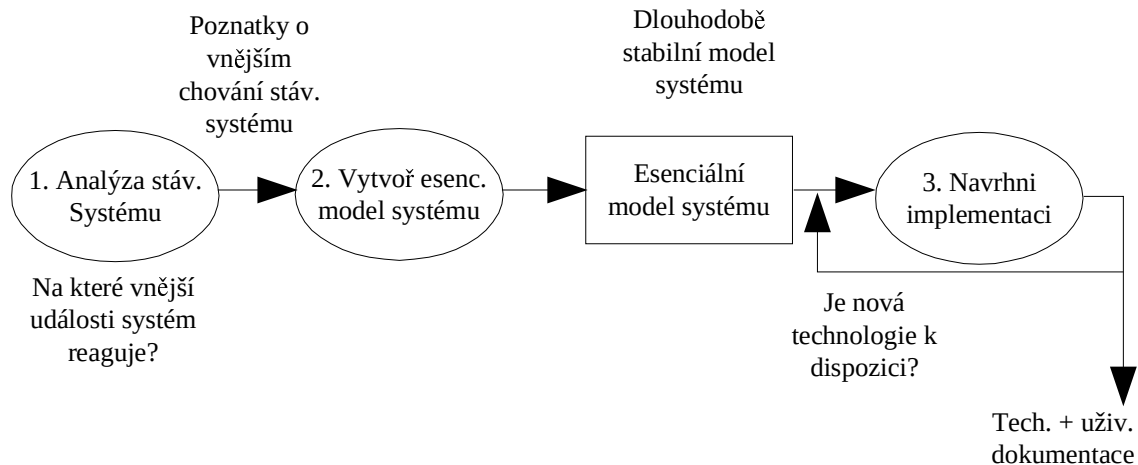
- účel systému (prvotní analýza)
- seznam událostí a reakcí na ně (Yourdon – YMSA)
- procesní (fční) model – návaznosti transformace dat
- datový model – atributy a struktura dat
- stavový model – diagram přechodů mezi různými stavy systému
- obsah/formáty výstupů

Yourdonova moderní strukturovaná analýza (YMSA)

Nejdříve obecně: analýza 4 modelů:



Analýza s esenciálním modelem:



Model okolí popisuje rozhraní mezi systémem a okolním světem.

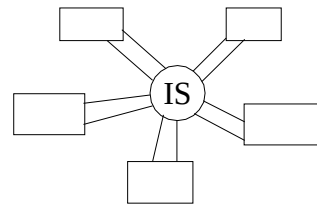
Model chování popisuje vnitřní části.

Úkolem analýzy je zjistit co je a co není součástí systému.

Esenciální model = model okolí + model chování

A) Model okolí obsahuje:

1. Kontextový diagram (zvláštní případ DFD) – terminátory, systém, data přijímaná, data vysílaná (produkovaná), datové oaměti, hranice.
2. Seznam událostí – textový výčet stimulů, které se objeví ve vnějším světě a na něž musí systém odpovědět. (toková událost – flow, časová událost (temporal), řídicí povel (control))
3. dokument o účelu systému
4. tvorba datového slovníku



B) Model chování:

1. dekompozice systému na jednotlivé procesy, toky a paměti, uspořádané v sadě DFD
2. pro každou odezvu na událost ze seznamu událostí zakreslíme do prvotního DFD 1 proces
3. proces pojmemujeme podle očekávané odezvy na tuto událost
4. zakreslíme datové paměti, které modelují data nezbytná pro zpracování asynchronně probíhajících událostí
5. doplníme odpovídající vstupní a výstupní toky
6. DFD ověříme proti kontextovému diagramu

Vyvažování:

1. **Nahoru:** hledáme seskupení vzájemně souvisejících procesů do agreg. procesu na diagramy vyšší úrovně
Seskupení procesů má zahrnout blízké, obdobné odpovědi (obdobně pojmenované procesy).
Paměti zakrýváme, pokud paměť není používána pouze skupinou procesů na nižší úrovni (každý jiný proces vně této skupiny paměť nepoužívá)
2. **Dolů:** čistá funkční dekompozice: u procesu se složitou funkcí jsou zřetelné dílčí funkce. Tyto funkce vyjádříme jako procesy na nižší úrovni DFD
Funkční dekompozice nezřetelná: pokusíme se odvodit dekompozici na základě vstupních a výstupních toků. Data určují směr dekompozice

Dokončení modelu chování:

Dokončení ERD a datového modelu jako celku:

- ERD vyvíjen jako DFD a souběžně s DFD
- kontrola proti DFD

Dokončení stavového modelu:

- definice všech možných stavů
- dosažitelnost stavů
- cesta z nekoncových stavů
- reakce na všechny možné podmínky v každém stavu

Dokončení esenciálního modelu obsahuje:

kontextový diagram, úplná sada vyvážených diagramů dat. toků, úplná sada entitně vztahových diagramů, úplná sada vztahových diagramů, úplný datový slovník (analytická etapa), úplná sada procesní specifikace.

Dokončení uživatelské implem. modelu:

- určení UI
- doporučení pro návrh rozhraní
- návrh formulářů
- identifikace manuál. podp. aktivit
- opatření pro chyb. technologie
- spec. operač. omezení

7. Objektová analýza a návrh, UML

(kde: ANANAS 11. a 12. přednáška, OO IS 1. přednáška (OO) 4. a 5. přednáška (UML), kniha 181 (OO), 194 (UML.)

- modeluje požadavky na systém prostřednictvím objektů a služeb, které poskytují
- objekty se oproti funkcím (příliš) nemění, tedy OO model tolik nezestárne a proto je lépe udržovatelný

Principy zvládnutí složitosti:

- abstrakce procedurální a datová
- zapouzdření
- dědičnost
- sdružování
- komunikace pomocí zpráv
- postup metody organizace
- měřítko
- kategorie chování

Abstrakce: principem abstrakce je ignorovat ty aspekty subjektu, které nejsou významné pro současný účel, abychom se mohli víc soustředit na ty, které významné jsou.

Procedurální abstrakce: každá operace, která docílí jasně definovaný výsledek, může být považována a používána jako jednoduchá entita, ikdyž ve skutečnosti je realizována nějakou sekvencí operací nižší úrovně.

Datová abstrakce: princip definice datového typu pomocí operací, které jsou aplikovány na objekty příslušného typu s tím omezením, že hodnoty těchto objektů mohou být modifikovány a pozorovány pouze pomocí operací.

Abstrakce vymezuje podstatné znaky objektu, které jej odlišují od ostatních druhů objektů, a tak poskytuje ostře definované koncept. hranice relativně podle perspektivy pozorovatele.

Zapouzdření, ukrytí informace – princip použitý při vývoji celé prog. struktury. Spočívá v tom, že každá komponenta programu by měla zapouzdřit a ukrýt jediné návrhové rozhodnutí.

Rozhraní každého modulu je definováno tak, aby odhalilo co nejméně o vnitřním dění v modulu.

Objekt má stav, chování a identitu. Struktura a chování podobných objektů jsou definovány v jejich společné třídě. Pojmy instance a objekt jsou vzájemně zaměnitelné.

Stav objektu: zahrnuje všechny (obvykle statické) vlastnosti objektu plus současné (obvykle dynamické) hodnoty těchto vlastností.

Chování: vyjadřuje, jak objekt koná a reaguje, v pojmech změn stavu a předávání zpráv.
Třída: je množina objektů, které sdílejí společnou strukturu a shodné chování. Objekt je instancí třídy.

Hierarchie – znamená hodnotní zařazení nebo uspořádání abstrakcí

1. „is – a” hierarchie („je něčím”) = dědičnost
2. „part – of” hierarchie („součást něčeho”) = sestava vnitřní struktury objektu

Dědičnost: mechanismus, který vyjadřuje podobnost mezi třídami, zjednod. definici třídy pomocí již dříve defin. tříd(y). vyjadřuje generalizaci a specializaci tím, že v hierarchii tříd explicitně určuje společné atributy a služby.

Propojení (vazba): fyzické nebo konceptuální spojení mezi objekty. Označuje speciální asociaci, pomocí níž 1 objekt (klient) používá služby jiného objektu (poskytovatel, dodavatel) nebopomocí níž může jeden objekt navigovat druhý objekt.

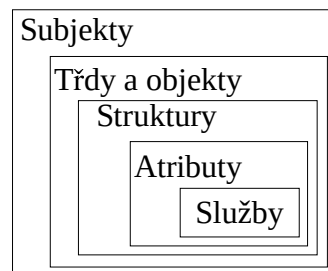
Role účastníků propojení:

1. **Aktor** (aktivní objekt) – může operovat s jinými objekty, ale sám není nikdy takto operován
2. **Server** – nikdy neoperuje s jinými objekty, a sám je operován jinými
3. **Agent** – operuje s jinými objekty a sám je operován jinými objekty (agenty či aktory)

Coad – Yourdon: 5-vrstvový model

5-etapový proces

1. identifikace tříd a objektů (dědičnost)
2. identifikace struktur („part – of” vztahů)
3. definice subjektů
4. definice atributů
5. definice služeb



UML (unified modeling language)

- standardní jazyk pro zobrazení, specifikaci, konstrukci a dokumentaci artef. systémů s převážně SW charakterem
- může být použit při všech procesech životního cyklu vývoje a pro různé technologie a implementace

Modelovací techniky UML:

- specifické vyjadřovací prostředky (jazyky) pro popis konceptů na vysoké úrovni abstrakcí
- grafické vyjádření – pro pochopení, dokumentaci a vysvětlení problému
- zápis myšlenek strukturovaným způsobem napomáhá ke zvládnutí složitosti a multidimenzionality SW

Modely v UML:

1. modely ukazující **statickou strukturu** systému
 - diagramy tříd a objektové diagramy
 - implementační diagramy (diagram komponent, rozmístění)
2. modely ukazující **dynamické chování** systému
 - model případů užití – externí pohled na systém
 - diagram aktivit – externě/interní pohled na systém
 - interakční diagram: dia. sekvencí a dia. spolupráce – interní pohled na systém
3. modely ukazující **dynamické chování jedné třídy**
 - stavové diagramy, diagramy aktivit

Diagram tříd – grafický pohled na statickou strukturu

Objektový diagram – instancí diagr. tříd, ukazuje snímek systému v určitém bodě.

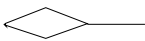
Třída - „příhrádka se jménem“, Kolekce objektů se shodnou strukturou, chováním, vztahy a sémantikou

Operace – chování třídy reprez. jejími operacemi. Ty jsou nalezeny po přezkoumání diagramu interakcí.

Atributy – reprezentují strukturu třídy. Jsou nalezeny přezkoumáním def. třídy, požad. na problémy a použ. znalostí z předmětové oblasti.

Vztahy – cesta po komunikaci mezi objekty.

- **asociace** – obousměrné propojení mezi třídami
- **agregace** – vztah mezi celkem a jeho částmi (silnější forma vztahu)

agregace: 

kompozice: 

- část agregace se silnějším vlastnictvím a spol. životem částí v celku. Části neexistují vně celku, každá část patří do jediného celku.

- **závislost** – slabší forma vztahu, mezi klientem a poskytovatelem, klient nemá žádnou znalost o poskytovateli

Násobnost – definuje, kolik objektů se účastní vztahů (počet instancí jedné třídy vztažený k jedné instanci druhé třídy). Pro asociaci a agregaci je násobnost 2.

Navigace – určuje směr propojení a směr návěstí.

Dědičnost – vztah mezi nadtřídou a jejími podtřídami

1. **generalizace (zobecnění)** – taxonomický vztah mezi obecným a specif. elementem, který je plně konzistentní s obecným elementem. Přidává dodatečné info.
2. **Specializace** – rozdělení do podtříd (asi)

Událost – z pohledu stavového diagramu je to výskyt jevu, který může spustit přechod mezi stavy.

8. Nástroje a modely datové, funkční a časové dimenze systému

9. Vývoj uživatelského rozhraní

(kde: kniha od str. 217, TIS II,

UI = User Interface = Uživatelské rozhraní

- návrh a testování
- kvalita UI silně ovlivňuje náklady na provoz IS
- ovlivňuje počet chyb, jichž se uživatel dopouští
- vhodné vyvíjet jako nezávislýb subsystém

UI musí:

- být vhodný pro daný účel a musí plnit zvolené funkce
- být snadno použitelný
- mít aplikovány zásady SW ergonomie

Etapy vývoje UI:

1. analýza požadavků
2. návrh UI
3. realizace prototypu + test s uživateli
4. integrace UI se zbytkem systému + test UI s uživateli
5. sledování a vylepšování

Testování UI:

Systém testují uživatelé, aleduje se jejich práce se systémem, celý proces se opakuje (iterativní vývoj), dokud uživatelé nejsou spokojeni.

- Sledování uživatelů
- scénáře (zaznamenávání ucelených postupů uživatelů při práci)
- myšlení nahlas
- heuristická evaluace (hodnotí se UI neformálně, podle sady doporučení – horké klávesy, jwdnoduchost dialogů, zpětná vazba, konzistentnost, kvalitní hlášení chyb, prevence chyb, nápověda a dokumentace, ...)

Akceptovatelnost SW:

1. sociální a společenská
2. praktická
 - užitečnost (usefulness): funkčnost, použitelnost (usability)- snadnost učení, efektivita, dobře se pamatuje, dobrá ergonomie, příjemnost, ...
 - cena

- kompatibilita
- modifikovatelnost
- spolehlivost
- dostupnost pro vš. uživatele

Životní cyklus UI:

1. Analýza a spec. požadavků

- poznání uživatele (koncového), analýza úkolů, fcí, požadavků uživ.
- analýza UI podobných nebo konkurenčních IS
- stanovení kontrolovatelných cílů
- stanovení finančního vlivu UI

2. Návrh UI

- paralelní návrh
- spoluúčast zákazníka
- koordinace návrhu
- heuristická evaluace

3. Implementace a test UI

- realizace prototypů
- předvedení prototypů UI (Potěmkin)
- zlepšení návrhů a odstranění nedostatků

4. test a provoz UI

- sledování metrik UI
- úpravy UI

(Více o testování UI s uživateli viz kniha str. 222, Grafické UI – kniha str. 224)

10.Problém testování

(kde: kniha od str. 203, TIS II)

- **Cílem** testování je nalezení chyb nebo ověření, že systém funguje jak má. Zjišťují se selhání, příčiny selhání apod.
- Testy bývají koncipovány tak, aby odhalily chybu, neboli tak přísně, jakoby měly dokázat, že systém není správný.
- Využívají se též výsledky oponentur všech etap vývoje SW, především oponentury programů, čtení kódu apod.
- Programy se testují a ladí tak dlouho, dokud frekvence selhání neklesne pod určitou mez.
- Programy by měly být navrženy tak, aby byly snadno testovatelné.

Testování probíhá v etapách:

- testování částí
- testování při integraci
- testování při předávání

Testují se tedy u velkých systémů nejprve malé části, posléze se spojují ve větší celky a ty se pak znova testují (integrační testování).

(strana 203-205 popisuje konkrétní činnosti při testování – příliš podrobné na opisování)

Testování mají provádět specialisté – testeři:

- **testování částí** – provádějí **kodéři**
- **integrační testy** – provádějí **testeři** (nejsou kodéři, většinou jsou organ. oddělení od programátorů) – výstupem: popis, co nefunguje, ne jak by to šlo dělat.

Integrace

- testuje se ještě v době, kdy systém není oživen
- po testech částí se části spojí a testuje se celek
- spojení naráz – nevhodné, pozdě se odhalí nedostatky v rozhraních
- lépe metoda „postupného nabalování“ - části se poměrně brzy spojují do vyšších funkčních celků.

- **Integrace zdola** – začíná od modulů, které nepotřebují jiné moduly, a postupně se přechází na moduly, které potřebují pouze integrované moduly. Vyžaduje to mnoho pomocných programů, které generují data pro prověřené moduly. Pozdě se ale ověřuje celý systém.
- **Integrace shora** – začíná od modulů, které nejsou potřeba v jiných modulech, a moduly, které jsou potřeba v daném modulu, se simulují. Podřízené moduly se naprogramují později a připojí do vzniklého systému.
- **Modifikovaná metoda integrace shora** – při postupu shora se může stát, že modul, který je kritický (může obsahovat mnoho chyb, není jasné, zda jej můžeme naprogramovat), bude integrován a zkoušen příliš pozdě. Pak lze postupovat shora, avšak tak, že v každé úrovni integrujeme pouze moduly nadřazené kritickému modulu.
- **Metoda sendviče** – systém se rozdělí vyšší část se integruje shora, nižší zdola. Oto je vhodné použít při programování operačních systémů nebo souborů programů. Moduly vyšší úrovně se mohou testovat nejprve každý zvlášť, ale v horším případě je možné, že uprostřed se to „nějak nepotká“...

Předávací testy – regression testing (celkové testování při údržbě) – soubor testů systému, vypracovaných obvykle ve spolupráci s uživatelem, nutných pro převzetí systému uživatelem.

Kritéria ukončení testování

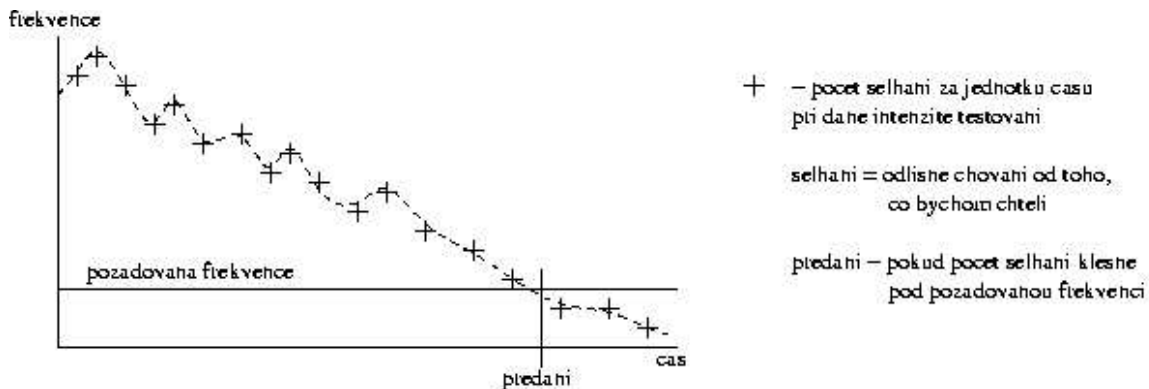
- jak dlouho a jak důkladně testovat?

- ukončí se, až je frekvence selhání dostatečně nízká (ale co konkurence? naše rozhodnutí by na tom nemělo záviset)

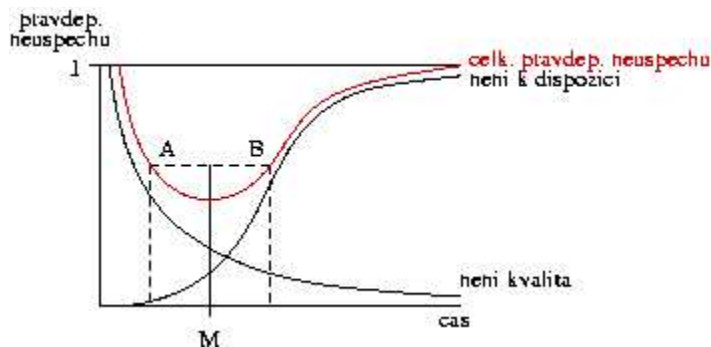
(pozn: pst = pravděpodobnost (pro zkrácení textu))

Rizika:

1. Příliš brzké předání nedoladěného systému zvyšuje pst, že systém neuspěje, tedy že zákazník buď systém nepřevzme, nebo s ním není spokojen, atp.
2. Prodlužování termínů zvyšuje nespokojenost zákazníka, má ztráty v době kdy systém není použitelný. U produktů dodávaných na trh opakovaně roste nebezpečí, že uspěje konkurence.



Celková pst neúspěchu je dána „superpozicí“ (většinou součtem) pstí obou jevů.



- úkolem managementu je ovlivnit pst.
- ukončit blízko minima M
- A vyžaduje při stejné psti méně nákladů (testuje se kratší čas, tedy je tendence se co nejvíce přiblížit minimu M, ale nepřeskočit)
- A menší spotřeba práce než B

Testování je nejobtížnější a nejpracnější část vývoje. Důvod: s použitím churchovy teze (žádný algorit. systém není výkonnější než turingův stroj) problém testování zahrnuje neřešitelné problémy:

1. problém zastavení
2. problém otestování všech částí
3. problém kombinací větví, dělení nulou, ...

Důsledek:

- testování je tvůrčí problém = v systému můžou zůstat chyby, neboť testování nelze algoritmizovat (plně, dost velké systémy)
- musíme dělat ad-hoc řešení – heuristiky - nelze otestovat dokonale (obecně nelze + cena testování)

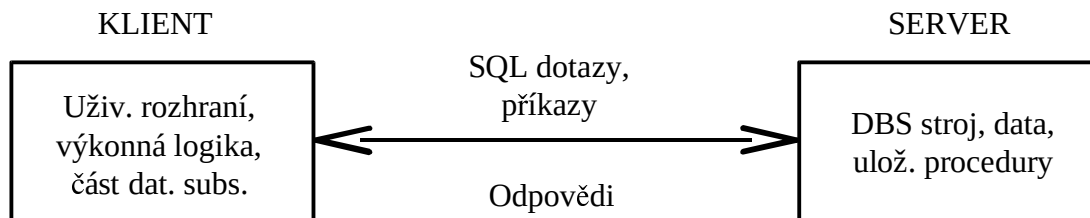
Zkušenosti z testování (data a programy) je dobré uschovat a znovu použít.

11. Architektura klient-server

(kde: kniha od str. 146)

Aplikace se rozdělí na 2 části:

1. **část klientová** (WIN, OS2, MACINTOSH)
2. **část běžící na výkonném centrálním počítači** (server či síť serverů, nejčastěji UNIX, AS/400 nebo MVS)



Klasické schéma spolupráce server – klient (tlustý klient)

Varianty dělby aplikace:

A: a) klient:

- vrstva styku s operátorem
- vrstva výkonné logiky
- část vrstvy správy dat (generace SQL příkazů / interpretace odpovědí)

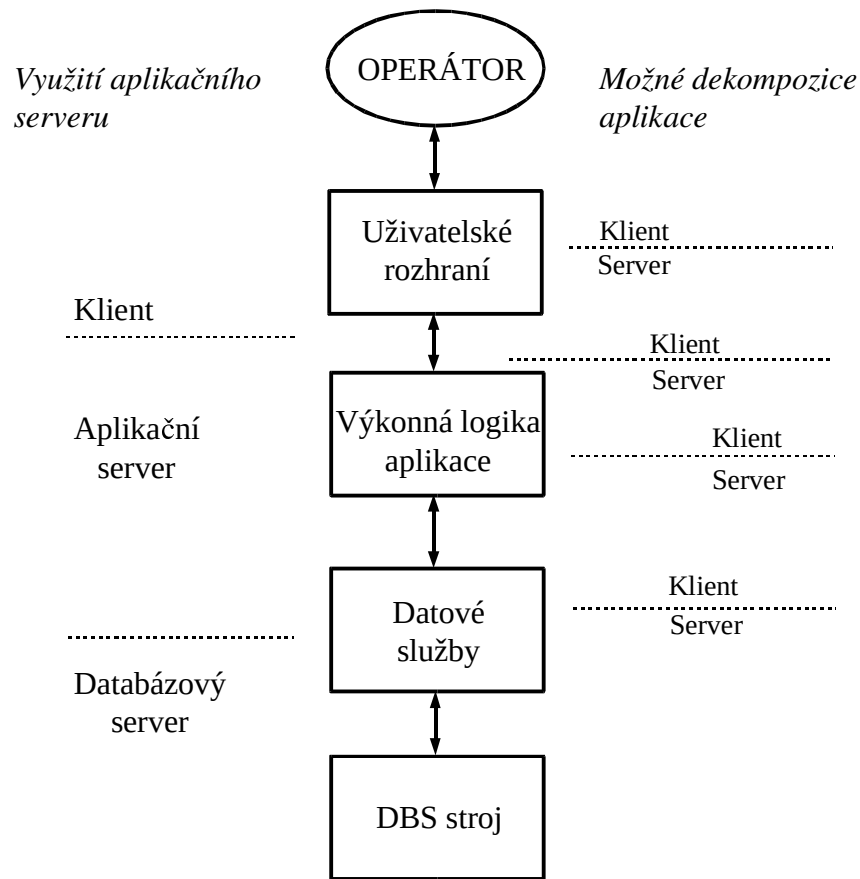
b) server:

- interpretace SQL příkazů (DBS stroj)
- vlastní data

- tlustý klient (*thick*) – viz obrázek
- spolupráce probíhá pomocí nástrojů, které nabízí výrobce databáze
- tlustý klient není vhodný pro IS s mnoha prac. místy.

B: částečné řešení: přesun části výkonné logiky pomocí tzv. ukládacích procedur na server s tím, že SQL dotazy jsou rozšířeny o správu triggerů. Neřeší ale problém tlustého klienta, není vhodný pro datové sklady, vhodný pro středně velké systémy.
- tlustý klient

C: optimální řešení: s využitím technik OO návrhu a prog. lze aplikaci rozdělit na jiném místě, například mezi vrstvou GUI a výkonnou logikou
- optimalizace zatížení klienta, serveru i sítě.
- tenký klient (thin) – neobsahuje ani rozsáhlá data ani programy

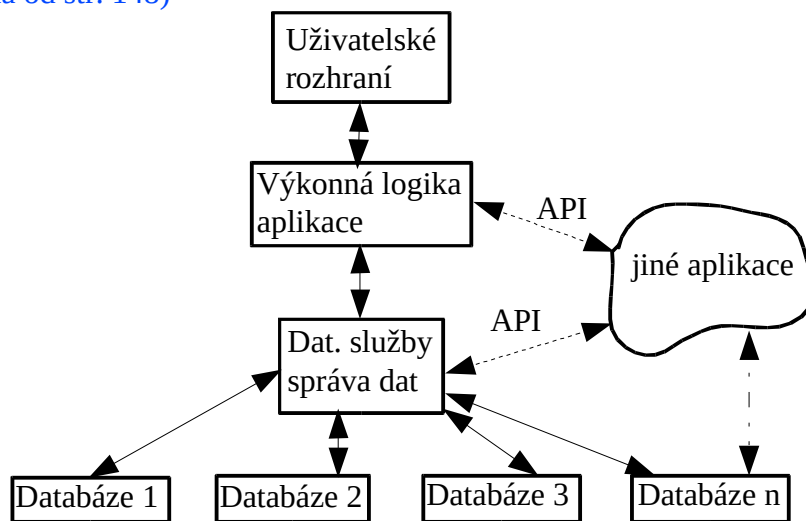


D: třívrstvá architektura: pro opravdu velké systémy se aplikace rozdělí na více částí:

1. klientovou
2. část logiky, která pracuje na dedik. počítačích (aplik. serverech)
3. část, která bude na serveru nebo serverech zajišťujících správu dat
dále viz další otázka:

12. Třívrstvá architektura

(kde: kniha od str. 148)



13. Konfederativní systémy

(kde: použit soubor ze stránky http://objekty.pef.czu.cz/2000/minikurz_kral_zemlicka.pdf TIS I

14. Procesní pohled na tvorbu SW

(kde: kniha od str. 287) – lépe číst z knížky, tohle je jen lehký souhrn...

Procesní model – zahrnuje prostředky řízení projektů a jejich integrace s vysloveně SW aktivitami, jako je správa konfigurace, sběr a vyhodnocení SW metrik atd.

SW proces – síť kroků nutných k dosažení stanoveného cíle (vytvoření či zdokonalení SW či customizace SW)

Krok procesu – je z hlediska procesního nedělitelná akce procesu, která není dále strukturována. Například: Návrh tvořený kroky: předběžný návrh a podrobný návrh.

Agent – aktivní entita (obvykle člověk), nebo počítač, nebo SW systém, provádějící daný krok procesu.

Zdroje – prostředky nutné pro provedení určitého prvku procesu.

Process Engineering zahrnuje následující činnosti:

- reprezentace procesního modelu
- analýza procesu
- instanciací procesu
- provedení procesu

Životní cyklus SW procesu:

- **analýza požadavků na SW proces** (volba modelu, např. inkrementální)
- **vývoj modelu** (tvorba provedení SW procesu)
- **přizpůsobení** (adaptace modelu na konkrétní projekt)
- **instanciace** (dopl. údajů o agentech – kdo co provede, a o zdrojích (co je třeba))
- **evoluce** (SW procesy jsou navrhovány jako schéma či plán budoucích činností)
- **provedení** (dle modelu se uskuteční vývoj SW)

Provádění SW procesu

- lze chápat jako síť spolupracujících a na sebe navazujících aktivit parujících buď:

- synchronně – aktivita čeká na jí vyvolané dokončení aktivity
- asynchronně – ostatní případy

Vlastnosti modelů SW procesů:

1. **věrnost** (fidelity) – vyjadřuje do jaké míry se provádí to, co je stanoveno v modelu procesu.
2. **vhodnost** (fitness) – vyjadřuje, do jaké míry lze při přesném provádění procesu s rozumným úsilím dosáhnout plánovaného cíle.
3. **přesnost** – vyjadřuje správnost, úplnost a jednoznačnost modelu
4. **redundance** – vyjadřuje počet takových kroků, které lze vynechat nebo nahradit jinými kroky.
5. **škálovatelnost** – vyjadřuje rozsah velikosti projektů pro něž je daný model procesu použitelný.
6. **udržovatelnost** – vyjadřuje, jak snadno lze model a jeho instance modifikovat

Dále vlastnosti známé z operačních systémů:

- **životnost** – vyjadřuje, že při provádění nedojde k uváznutí (deadlock)
- **robustnost** – vyjadřuje stupeň ochrany před neautorizovanými změnami a stability při vzniku neočekávaných akcí.
- **stabilita** – stupeň ochrany vůči nesprávným změnám a celkové spolehlivosti
- **interakce s okolím** – stupeň autonomi a rozsah interakce s jinými procesy.

Modelování procesů

- grafickými prostředky
- jazykově

(dále na pročetění od strany 291...)

15.Zásady tvorby dokumentace

(kde: kniha od str. 277, TIS II)

Dokumentace: technická, administrativní, ekonomická.

Musí:

- být aktuální, přehledná, srozumitelná, tak aby byly možné opravy a modifikace programů bez rizika zavlečení dalších chyb

- obsahovat info o tom, jak systém používat, instalovat, obsluhovat, udržovat, popis realizace, testů, atd.
- při předání dokumentace obsahuje manuály a uživatelskou dokumentaci, dok. pro údržbu (zdrojáky s komentáři, systémovou dok., dok. o SW, ...)

Uživatelská dokumentace:

- poskytuje představu o práci se systémem (uživatel by měl najít právě jen to, co potřebuje ke své práci)
- obsahuje návod k instalaci, úvod do systému, popis funkcí, referenční manuál, návod k použití, příručka operátora.

Dokumentace pro údržbu:

- účinná pomůcka je slovník dat, jeho položka obvykle obsahuje:
- identifikátor objektu, význam objektu, typ hodnot, oblast platnosti, křížové odkazy, ...

Při psaní dokumentace:

- uplatňovat procesy kontroly kvality
- tvar dokumentace standardizován
- jazyk by měl být jazykem odborných článků

Zásady:

- kratší věty
- u odkazů nejen čísla, ale i názvy kapitol
- maximální stručnost
- ale někde nešetřit slovy, když je potřeba něco vysvětlit
- jednoznačnost použitých termínů
- dokument do odstavců a paragrafů
- dobrá grafická úprava
- hierarchické členění

Nástroje:

editační systémy poměrně vysoké kvality; jsou vhodné full text databáze.

Údržba dokumentace: nutná spolu se systémem.

Kvalita: srozumitelnost, úplnost, modifikovatelnost, vystopovatelnost, jednoznačnost, použitelnost, dostupnost, aktuálnost.

16. Softwarové metriky

(kde: kniha od str. 227-262, TIS II, ANANAS 15. přednáška,

sledování kvantitativní charakteristiky (počet řádků, doba řešení, spotřeba práce, ...)

- základ zajištění kvality SW
- hodnoty metrik jsou cosi jako paměť firmy
- metrika = „kvalifikovaný“ atribut SW



Druhy metrik:

External – explicitní – zjistí se z artefaktů produktů systému (například délka programu, počet podprog., počet metod, rozsah a výskyt operandů, rozsah a výskyt operací, počet fcí, počet datových tabulek, ...)

Internal – implicitní – z procesů během vývoje (např doba řešení, velikost týmu v čase, spotřeba práce, produktivita, počet selhání systému v čase, defekty, spokojenost zákazníka, ...)

Potíže se metrikami:

- rozptyl hodnot
- produktivita práce programátorů, jejich kvalita
- druh SW, obtížně splnitelné termíny realizace
- moderní projekční techniky
- kvalita zúčastněných
- omezení SW a HW

Datové typy metrik:

- příslušnost ke třídě (číslo tramvaje), vztah: =
- fuzzy metrika – slabý, dobrý, velmi dobrý, vynikající, vztah: =, ≤
- fuzzy číselná – známky ve škole, vztah: =, ≤, (aritm. operace)
- intervalová (teplota)
- čas, vztah: =, ≤, interval, omezující operace
- plná data (délka)

17.CMM (Capability Maturity Model)

(kde: kniha od str. 292-295)

- cílem je zvýšení spokojenosti uživatelů Swsyst., zlepšení kvality SW a omezení rizik spojených s vývojem SW
- nástroj na zlepšení efektivity práce firmy, obsahuje postupy na snižování nákladů, zkrácení termínů, eliminaci rizik spojených s migrací pracovníků.
- CMM hodnotí vyspělost organizací podle stupně a kvality využívání SW procesů- SWP

Pět úrovní využívání SWP:

1. **Počáteční úroveň (initial level)** – SWP v neformální formě, a definuje se případ od případu od počátku, nejsou pevná pravidla plánování a řízení projektů. Výsledky záleží spíše na kvalitě jednotlivce než organizaci práce, zkušenosti se nevyužívají, po odchodu pracovníka ztraceny.
2. **Úroveň zajišťující opakovatelnost (repeated level)** – zavedena pravidla pro řízení projektu, plánování a řízení založeno na zkušenostech, jednotné zásady pro celou firmu, SWP nejsou standardizovány, plány realizace se sledují, náprava při odchylkách.

3. **Úroveň definovaných procesů (defined level)** – SWP standardizováno (jak procesy SW – inženýrské, tak manažerské), součástí norem jsou nástroje kontroly a zvýšení efektivity práce, zkušenosti a osvědčené metody a postupy, součástí standardů jsou procedury přizpůsobení SWP na konkrétní projekt, zajištěna kontrola dodržování požadavků, nákladů a termínů, SWP založeno na odb. zázemí a znalostech pracovníků firmy, pravidelná školení
4. **Úroveň řízení procesů (controled level)** – definovány metriky kvality pro SWP i vývoj SW, systém sběru, sledování a vyhodnocování metrik jednotným způsobem v rámci celé organizace, firma schopna vyhodnocovat trendy a odhad hodnoty důležitých metrik, schopna odhadnout přesnost odhadů přes kontingenční interval.
5. **Úroveň optimalizace procesů (optimized level)** – zavedeny procedury neustálého vylepšování SWP, vytvořen tým hodnotící kvalitu procesů a navrhuující vylepšení včetně zavádění nejnovějších metod, postupů a nástrojů, tým analyzuje příčiny úspěchů i neúspěchů, pak modifikuje SWP

CMM doporučuje procházení postupně jednotlivými úrovněmi 4 a 5 spíše u velkých firem.

18.COCOMO a funkční body

(kde: kniha od str. 230, 264-266, ANANAS 15. přednáška,

COCOMO

- vychází z odhadu délky programu v tisících nově napsaných řádků

Typy projektů:

1. Organický typ

velikost projektu = málo set tisíc řádků, malé problémy pro malé týmy, typické jsou mírné normy a malá omezení na specifikaci rozhraní a možnost ovlivnit požadavky. Algoritmy a postupy jsou dobře známy, podmínky realizace relativně stabilní, nejsou ostré podmínky ani termíny. (Příklad: zpracování dat v dávkovém provozu, úpravy známého operačního systému nebo kompilátoru)

2. Přechodný typ

typ mezi organickým a vázaným, velikost projektu < 400 tisíc řádků, pokud není kód generován automaticky. V týmu jsou zkušení i méně zkušení pracovníci, tým má ne moc velké zkušenosti z předchozích obdobných realizací, úloha poměrně složitá (Příklad: menší dedikované oper. systémy, běžné transakční systémy, systém řízení výroby, jednoduchý systém řízení vojsk a zbraní)

3. Vázaný typ

SW pracuje za velmi ostrých omezení na výkonnost a dobu odezvy, velikost jsou miliony řádků, vyžaduje práci s komplikovanými SW a HW systémy za ostrých předpokladů na předepsané fce, spolehlivost, termíny, přenositelnost, modifikovatelnost. Požadavky je obtížné měnit, řeší se nové problémy, se kterými se pracovníci dosud nesetkali. Obvykle: specifikaci požadavků + návrh dělá malá

skupina, vývoj částí – velký tým. Programování a testy – souběžně. (Příklad: nové rozsáhlé oper. Systémy, , kosmické lodě, letadla, atomové elektrárny, RT aplikace)

Atributy SW produktu:

RELY – požadovaná spolehlivost

DATA – velikost databáze

CPLX – složitost produkt

Atributy HW produktu:

TIME – omezení času výpočtu

STOR – využití paměti/disku

VIRT - spolehlivost virtuálních strojů

TURN – míra rychlosti oběhu úlohy počítačem

Atributy vývojového týmu:

ACAP – analytické schopnosti

PCAP – programovací schopnosti

AEXP – zkušenosti s podobnými aplikacemi

VEXP – zkušenosti se spec. virt. strojem

LEXP – zkušenosti se spec. prog. Jazykem

Atributy projektu:

MODP – použití moderních prg. technik

TOOL – použití SW nástrojů

SCED – přesné plánování

Postup při stanovení odhadu pracnosti:

1. určí se typ SW
2. určí se hodnocení vlivu faktorů reprezent. jednotl. atributy (velmi málo, málo, standard, mnoho, velmi mnoho, extrémně mnoho)
3. určí se číselné hodnocení atributů z tabulek
4. určí se hodnota K jako součin takto získaných hodnot
5. podle typu SW se vypočtou hodnoty odhadu
hodnocení: nepohodlí, malé ztráty, nahraditelné ztráty, velké ztráty, ohrožení lidí

Závislost na metrice Del (délka prg. v řádcích):

Organický typ: $Prac \approx 2,4 \cdot K \cdot Del^{1,05}$ $D \approx 2,5 \cdot Prac^{0,38}$

Přechodný typ: $Prac \approx 3,0 \cdot K \cdot Del^{1,12}$ $D \approx 2,5 \cdot Prac^{0,35}$

Vázaný typ: $Prac \approx 3,6 \cdot K \cdot Del^{1,20}$ $D \approx 2,5 \cdot Prac^{0,32}$

FUNKČNÍ BODY

- měří aplik. oblast, nezkoumá technickou
- zkoumá aplik. fce a data, ne kód
- měří se vstupy, výstupy, dotazy, vnitřní paměti, vnější paměti

odhad = velikost projektu x složitost x rizikové faktory

Typy fčních bodů (vztahené k transakčním fcím):

EI – externí vstupy

EO – externí výstupy

EQ – externí dotazy

Typy fčních bodů (vztahené k datovým fcím):

ILF – vnitřní logické soubory

EIF – soubory vnějšího rozhraní

Neupravené fční body:

Váhy:	nízká	průměrná	vysoká	celkově
EI	... × 3 +	... × 4 +	... × 6 =	...
EO	... × 4 +	... × 5 +	... × 7 =	...
EQ	... × 3 +	... × 4 +	... × 6 =	...
ILF	... × 7 +	... × 10 +	... × 15 =	...
EIF	... × 5 +	... × 7 +	... × 16 =	...

Neupravené fční body celkem: = ...

Obecné charakteristiky systému při úpravě fčních bodů: (například)

- jsou vyžadovány datové komunikace?
- je výkonnost kritická?
- Požaduje systém on-line vstup dat?
- Je kód navrhován s cílem znovupoužití?
- Je systém navrhován pro násobné instalace u různých organizací?
- Jsou konverze a instalace zahrnuty v návrhu?
- Rychlost zpracování transakcí
- složitost zpracování
- on-line oprava
- atd (celkem 14 charakteristik, viz kniha)

Vlivy:

bez vlivu	0
bezvýznamný/náhodný	1
mírný vliv	2
průměrný vliv	3
významný vliv	4
podstatný vliv	5

Součet 14 hodnocení dá tzv. FAKTOR TECHNICKÉ SLOŽITOSTI = TFC

Faktor přizpůsobení = $0,65 + (0,01 \times TCF)$

Výsledný počet fčních bodů systému:

Upravené fční body = $0,65 + (0,01 \times TCF) \times [\text{počet nepřizp. fčních bodů}]$

19. Počítačová ergonomie

(kde: kniha od str. 49-57, TIS I)

zdravotní a sociální důsledky používání PC

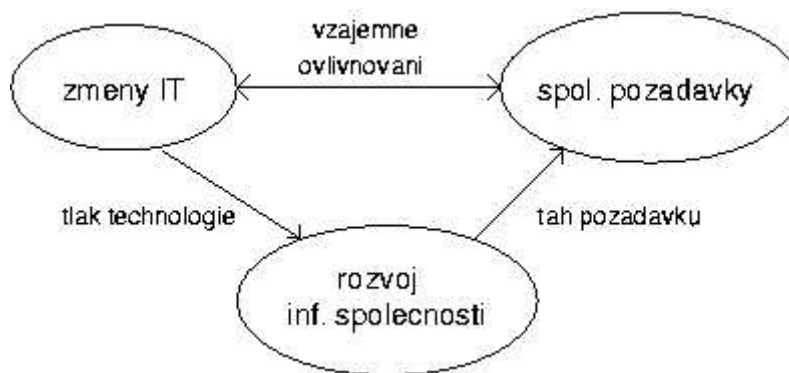
analogie: nová technologie = nové průšvihy

- fyziologické (nejvíce postižené ruce)
- psychologické/sociální

Sociální problémy (sociální deviace)

- stálé změny zaměstnání, kvalifik. Předpokladů, sociálního postavení, nutnost vzdělávání
- ohrožení z vnějšku (globalizace) – stále více lidí se cítí být ohroženo okolím (- různá opatření)
- černobílý přístup informatiků (zkratové myšlení – př. nenávidíme mat. Statistiku)

motor změn:



! ne všichni mají schopnost se účastnit změn (zůstávají na okraji společnosti)

Počítačové nemoci z povolání:

- je jich řada
- snižují výkonnost
- regres? (žalovat o náhradu za újmu na zdraví)
- expoziční doba (inkubační) = průměrná doba působení škodlivého faktoru až do projevení nemoci (je několik let)
- objektivní potíže – dají se přímo změřit, pozorovat (například změna dioptrií)
- subjektivní potíže – většinou postihuje mozek – např. nespavost
- poč. nemoci většinou z: špatné návyky + přetěžování – vede k poškození (projevuje se plíživě)

Objektivní:

1. RSI (GUI) Repetitive Strain Injuries

- poškození z opakované zátěže (práce s grafickým prostředím)
- zátěž: práce s myší a klávesnicí
- poškození rukou:
 - a) záněty + změny loket. kloubu („tenisový loket“) - bolesti kloubu, snížená pohyblivost – důvody: nevhodná poloha klávesnice a myši (stůl), dlouhá práce (více jak 6 hodin bez přestávek)
 - b) záněty kůstek v ruce, kloubů – bolesti, ztráta schopnosti jemného stisku (psaní) – důvod: chybná poloha klávesnice, nacvičen nesprávný způsob psaní
 - c) ramenní kloub
 - d) otoky, záněty nervů v paži a ruce, záněty šlach – celková bolest, ztráta hmatu a schopnosti jemné motoriky
- obrana: méně než 6 hodin práce, přestávky (10 minut/hodinu), nápravná cvičení, střídání činností (klávesnice – myš – práce mimo PC)
- přehled příznaků:
 - bolesti (při pohybu, v klidu) – v paži, v ruce
 - ztráta citlivosti hmatu
 - neschopnost jemné motoriky
 - třes
- podle všech znalostí jsou tyto změny VRATNÉ
- otázka: sázka na grafické rozhraní – zatěžuje uživatele více?

2. Nemoci ze sezení

- a) poškození krční páteře (otoky, bolesti), „nemoc pokladních“ - krk, svaly ramenní – důvod: dlouhá práce, nevhodná výška umístění monitoru a podkladů
! POZOR ohrožení vegetativního nervstva (ohrožuje řízení vnitřních orgánů – souběžně s míchou, neovladatelné vůlí) = žaludeční neurózy, potíže s trávením, problémy se srdcem, těhotenské potíže (silnější) až ztráta vědomí.
- b) poškození bederní páteře (ischias)

Subjektivní:

- způsobené špatným rozdělením práce za den
- noční můry (nespavost) – práce dlouho do noci
- poruchy soustředění
- večerní ztráta ostrosti vidění (extrémně – ztráta vidění - na čas)
- neurózy, psychická labilita, nespokojenost (projevy stresu)
- zhoršení zažívání

Neověřeno / pravděpodobné:

- zhoršení zraku – není prokázáno statisticky (existuje spousta dalších činností, při kterých se kazí oči, překrývají se, nedá se statisticky prokázat) – nemáme lepší metodu

Souvislost práce s počítačem se sociálně patologickými jevy:

- určitě existuje (hackeri)
- sociabilita dětí (nenaučí se jednat s lidmi)

Ergonomie SW (jak dobře navrhnout SW)

- co nejméně automatizovat (co nejvíce se ptát uživatele) – inkrementální, iterativní vývoj (pouze to, co přinese efekt – postupně přidáváme)

! začínat od nejlepšího (největší efekt)

- střídat činnosti: datový vstup – myš – papíry
- organizace práce (6 hodin, přestávky)
- ergonomie rozhraní (jak navrhne obrazovky)
- ergonomie místa a prac. prostředí

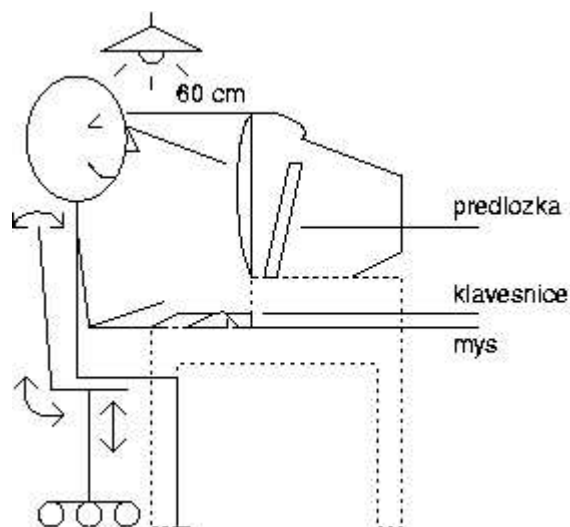
- **rozhraní (obrazovky) Usability Engineering**

- maximálně 12 logicky rozdílných údajů (co nejméně věcí, kterých si musí člověk všimnout)
- vhodné členění
- barvy, logicky blízké – blízko sebe, nestřídat rychle

- **pracovní místo**

kvalitní židle (nastavitelný sedák, pružný sedák + opěradlo, nastavitelná výška, zvlášť podloketní opěrky)

- myš a klávesnice tak, aby mohl být loket u těla, předloktí mírně zvednuté a ramenní část paže svisle dolů, úhel mezi předloktím a ramenní částí je cca 90 stupňů.
- monitor – vršek ve výšce očí (neboli střed asi 15 stupňů pod úroveň očí), vzdálenost od očí aspoň 50-60 cm, frekvence individuální, alespoň však 70-80 Hz, sousední monitory (zadní strany) cca 1.5 metrů od osoby.
- okna z boku, nejlépe zprava
- individuální osvětlení s proměnnou intenzitou
- dobrý výhled nejlépe do přírody (uleví očím)
- předlohy na podložce hned vedle monitoru



- **pracovní prostředí** (běžný vývoj, nevhodné pro více různých činností)
 - 2 varianty:
 - a) ratejna (hosting) – velká hala, desítky pracovníků, zábrany pohlcující zvuk, pořád stejné pracovní místo nebo pokaždé jiné
 - b) menší pracoviště, 2-3 lidé v místnosti
 - vždy: společenské místnosti (kuchyňka, konfer. místnost)
 - výhled do přírody, okna ne na J, JV, JZ
 - estetika (květiny, část ženy)
 - společenská zařízení (tiskárna, ...)

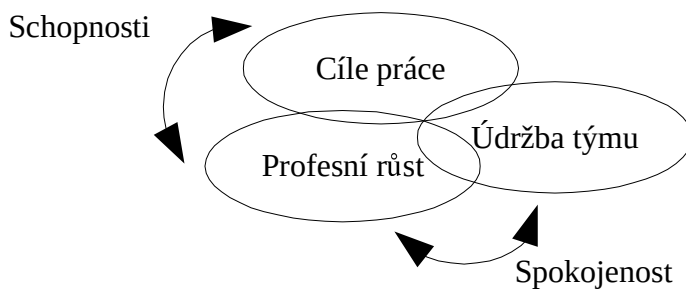
20.Práce v týmu

(kde: kniha od str. 121, TIS II)

- na dobře fungujícím týmu závisí úspěch projektu
- pouze v týmu jsou řešitelné velké úkoly
- vlastnosti a schopnosti členů týmu se musí vzájemně doplňovat
- umět pracovat v týmu! Vzájemná podpora a stimulace lidí v týmu, využití talentů
- profesní růst členů týmu

Nevýhody: menší produktivita, občas byrokracie, pro mnohé horší práce, potřeba speciálních talentů

Činnosti týmu:



- výkonnost lidí závisí na vzdělání, vedení, „poměrech“ v týmu a v podniku (spokojenost)
- důvěra – menší tendence odejít, a tedy se neztratí vzdělání do kterého jsme investovali.
- rizika vyškolení:
 - naučí se a odejdou – obrana: kariérní perspektiva, práce, poměry
 - učí se nevhodné (opONENTI, poradci)
 - nevhodní lektori (univerzity?) - nejen úzké znalosti
- důležité: používat ke hledání talentů, důležité jsou personalisté!

Práceschopnost:

1. dřeva – malá píle, malý talent – vyhodit
2. dřiči – velká píle, malý talent – motivovat, profesní růst, odměny
3. zlobivé děti – malá píle, velký talent – kontrola, autonomní úkoly
4. hvězdy – velká píle, velký talent – snažit se udržet, nechat dělat to, na co ostatní nestačí, nepřetěžovat!

Snadnost výměny:

		Výměna	
		<i>snadná</i>	<i>těžká</i>
Vliv na hosp. Výsledky	<i>Malý</i>	Linioví pracovníci	Experti
	<i>Velký</i>	Manažeři	Vizionáři, vývojáři

Osobnostní typy:

- workoholik** – motivován prací (zlobivé děti)
- kamarádi** – baví je spolupracovat (často ženy)
- sobci** – motivací je vlastní kariéra

Zásada: max. 1 sobec, 2-3 workoholici, zbytek kamarádi, lépe koedukovat

Důležité: nepřítomnost sociálních podrazů! Fungující tým = dobré klima = týmová solidarita (identifikace týmu, obrana týmu), pozor na týmový šovinismus (nekritická obrana týmu + animozita k cizím týmům)

Neegoistické jednání – brát to tak, že chyby jsou nutné zlo, programy a dokumentace jsou společné dílo, ne jen jednotlivce; umět volit řešení, které je optimální pro celý tým, i když pro jednotlivce může být dočasně nevhodné.

Komunikace v týmu:

1. každý s každým – náročné na čas, nepřehledné
2. vše přes společného vedoucího – nárůst administrativy

Pracovní procesy týmu:

1. výběr úkolu
2. formování týmu

- určení vedoucího + jádra
 - analýza hl. rysů úkolu
 - další členové
3. krystalizace úkolů (zpřesňování úkolů, diskuse o způsobech řešení, převzetí rolí, formování podtýmů)
 4. vyjasnění úkolů (přijetí zásad řešení, návrh struktury týmu, souhlas s cíly, volba norem a pravidel práce)
 5. realizace (definitivní stanovení struktury týmu a podtýmů, defin. přijetí rolí, vlastní provedení úkolu)

Role = postavení jedince ve struktuře týmu, souhrn činností, které může provádět jeden pracovník. Role musí být správně stanovena. Problémem bývá konflikt rolí, nekompatibilní představy o roli, moc velký nebo malý rozsah úkolů pro roli, role neodpovídá profesnímu zaměření apod.

Souhlasy s rozhodnutím:

- hotová věc
- klika
- dohoda menšiny
- dohoda většiny
- všeobecný konsensus
- zdánlivý konsensus

Vedoucí týmu (+ jeho zástupce)

Vlastnosti: kompetentnost, znalost vlastních + a -, sebedůvěra, formulační schopnosti, personalistika, management, dělat příklad, vychovat nástupce, detekce průšvihů, loyality k podniku i k týmu, podpora růstu týmu, umět podržet tým při neúspěchu, nepanikařit a nezmatkovat, umět jednat s lidmi, psychická odolnost.

Činnosti: plánování, vysvětlování, kontrolní akce, podpora, informování, regulace, hodnocení, delegování pravomocí, udržování a rozvoj pozitivních postojů členů týmu.

Neformální role v týmu:

1. **kladné**: iniciátor, inovátor (hledač info.), encyklopedista (má velké znalosti nebo ví, kde co najít), cizelér (dokončuje jemné práce, zahlučuje), koordinátor, navigátor, šťoural (v dobrém smyslu – hledá chyby, které ostatní přehlédli), provozář, hecíř (dokáže vyhecovat k lepším výkonům), harmonizátor (harmonizuje a uklidňuje vztahy v týmu), moderátor, normovač.
2. **záporné**: agresor (agresivní vůči ostatním), negativista (negativní názor za každou cenu), exhibicionista (předvádí se), playboy (svět je pro něj jen sexuální loviště, obtěžuje spolupracovníky apod.), kecal (do všeho kecá, nic nedělá), vládce (snaží se dělat vedoucího, ovládat ostatní), populista (vystupuje a štví proti vedoucímu), kanadčan (dělá hloupé žertíky na úkor ostatních)

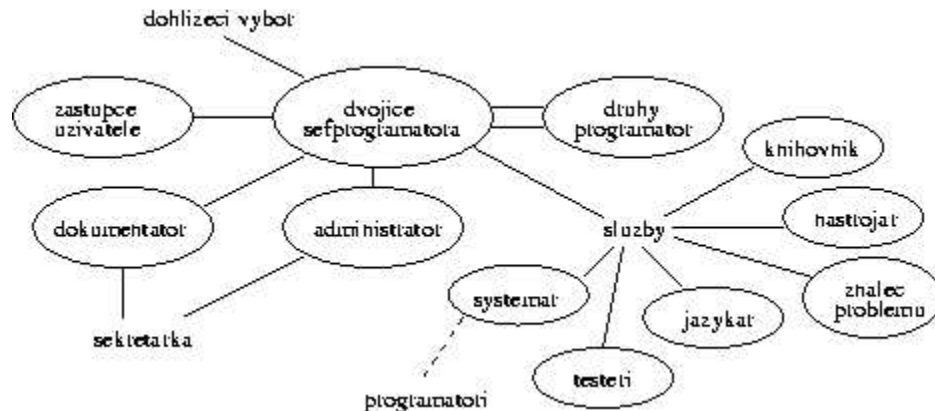
Psychohry:

- „Alkoholik“ - účastní se „alkoholik“, karatel, utěšovatelka, „kumpáni“. Osoba, která se nějak provinila, nebo je to nevýkonný pracovník (alkoholik), je kárána vedoucím, který se jej snaží napravit (karatel), utěšovatelka ho utěšuje, ať si z toho nic nedělá, a že pro něj nemají pochopní, čímž negativně ovlivňuje efekt kárání na provinilce, a ten se pak nesnaží o nápravu svého jednání, totéž kumpáni, tj. spolupracovníci, kteří říkají „nám taky vynadal, vybodni se na to ...“ a všemožně jej přesvědčují, že o nic vlastně nejde.
- „Beru vše“ - pracovník si nabírá spoustu úkolů, pak nic nestíhá a vymlouvá se na přetížení.
- „To máte z toho, že jste chtěli ...“ - výmluva. Odmítání kritiky s využitím toho, že se musel podřídit nějakým normám, nebo rozhodnutím.
- „Ty máš taky máslo na hlavě, tak si nevyskakuj...“ - využívá slabosti partnerů i tehdy, mají-li oprávněné výhrady. Kritizovaný odvrací oprávněnou kritiku poukazováním na dřívější selhání těch, kteří kritizují.
- „Ano, ale ...“ - odmítání disciplíny pod různými záminkami
- „Kanaďan“ - ruší tím, že neomaleně provádí kanadské žertíky

Druhy týmů:

- **Osamělý vlk (netým)**
 - jeden člověk si udělá vše
 - menší úkoly (zlobivé děti) - konfederace
- **Supervlk**
 - vedoucí + programátoři (náznak týmu) třeba po celém světě, využívá se internet. Vhodné pro střední úkoly, supervlk vymyslí a nechá naprogramovat části.
- **(Super)Dvojice**
 - střední až velké úkoly, závislost na vedoucím není absolutní (narozdíl od vlků)
- **Samoorganizující se týmy**
 - 1) **Horda** – bez hlubšího dělení rolí se jde hned na věc, každý dělá na svém úkolu vše, od analýzy až po konec. Nepoužívat, velmi nevhodné!!
 - 2) **Demokratický tým** – jednoduchá profesní ešlba práce, dohodou na základě analýzy se rozdělí role, zahrnují i služby, vhodné pro stálý tým i větší a nové systémy, podmínkou je vysoká úroveň členů týmu, nevhodné pro moc velké projekty, tvrdé termíny, kritické aplikace apod.
- **Tým šéfprogramátora**
 - několik vynikajících odborníků + poměrně mnoho relativně nezkušených pracovníků, spousta práce úřednické, práce s dokumenty.
 - *šéfprogramátor* – nejschopnější člen týmu, dekompozice problému, návrh řešení, algoritmů, dokumentace, je schopen realizovat celý projekt sám, ale nesmí být rozptylován. Velice talentovaný.
 - *druhý programátor* – dvojníkem šéfprogramátora, je schopen mu oponovat, taky talentovaný, zastupuje tým vůči okolí.

- *dokumentátor* udržuje dokumentaci (zda je úplná, ...)
- *administrátor* - admin. věci (peníze, kdo nastoupil, ...)
- *služby* - knihovník, nástrojář, systémák, ...
- *systémák* - stará se o chod systému, udržuje datové báze, ...
- *jazykář* má přehled o tom, jak nejlépe používat zvolené SW nástroje
- *nástrojář* rozšiřuje paletu SW nástrojů



• **Supertým – tým hlavního programátora**

- koordinuje více týmů (velké úkoly)
- týmy zajišťují: jednotlivé služby (testy, péče o systém a tvorbu nástrojů, ...), samostatnou realizaci subsystémů, koordinaci prací, ...
- hlavní programátor vede tým, přijímá řešení, nemusí se zabývat administrativou
- druhý programátor je ideový partner hlavního programátora, vazba je volnější než u šéfpřipravitelů.
- programátorské skupiny – nejnižší úroveň organizace, 2-3 členové, demokratická skupina.



- **Multitýmová organizace (konfederace)**

- má svou strukturu (vedoucí skupina, členy jsou jednotliví vedoucí řešitelských týmů, týmy mají vlastní strukturu)
- větší decentralizace prací a odpovědností.

- **Maticové týmy:**

		Služby					
		P_0	1	2	3	4	5
Projekty	1		P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
	2						
	3						
	4						

P_{ij} Lidé služby i pro projekt j = dvojí podřízenost – problém (ale můžeme zajistit, aby jednotlivé služby byly vyvíjeny specialisty)

- osvědčuje se u velkých firem

- P_0 jsou kmenoví pracovníci projektu – patří k jádru projektu

Co kdy použít:

		<i>velikost</i>		
		malá	střední	velká
<i>novost</i>	malá	Jeden/dvojice	Samoorg/hierar	Hierar/samoorg
	střední	samoorg	Samoorg/hierar	hierar
	velká	samoorg	samoorg	Nelze dělat jako monolit

- hierarch – přísné dělení rolí. Výhoda je, že i slabší členové jsou užiteční.

- velké systémy - nedají se dobře kopírovat, hodně peněz, menší konkurence, snažit se rozdělit na středně velké části

Druh úkolů:

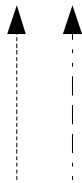
velmi rozsáhlé a komplikované
 rozsáhlé a komplikované
 rozsáhlé
 střední složitosti



požadavky na spolehlivost části —————▶
střední úroveň kvalifikace členů týmu▶

Typ organizace:

multitýmová organizace
tým hlavního programátora
tým šéfprogramátora
demokratická skupina



doba řešení úlohy dané složitosti▶
požadavky na kvalitu organizace - - - - -▶

1. XML a odvozené jazyky