1. Popište smysl pyramidy vitality a význam jejích jednotlivých pater

Teorie vitality – aplikace zdravého rozumu při vedení firem k úspěšnému fungování, analogie mezi fungováním moderní firmy a přírodními systémy

1. ***Užitečnost*** – firma musí mýt smysl, poskytovat užitek, produkovat něco natolik chtěného, že někdo
   - za to zaplatí – zákazník
   - investuje čas, energii, práci – zaměstnanec
   - investuje nápady, peníze – majitel
2. ***Efektivita*** -firma má pořádek v procesech – nesmí se příliš vysílit, musí být efektivní alespoň jako konkurence, nad popsanými procesy vybuduje organizační struktura a v procesech hospodárně využívá zdroje
3. ***Stabilita*** – schopnost nacházet rovnováhu v měnících se podmínkách, je nutné vybudovat systém zpětných vazeb, individuálně se věnovat jednotlivým zaměstnancům a zapojovat je do firemních dějů
4. ***Dynamika*** – vyvolávání změn, firma se stává původcem změn, systém dopředných vazeb, ovlivňování mezilidských vztahů ve firmě, skupinách, týmech

2. Popište smysl Maslowovy pyramidy potřeb a význam jejích jednotlivých pater

Popisuje univerzální lidské potřeby

Smysl - jsem užitečný sobě a svému okolí? Seberealizace, osobní rozvoj, užitečnost pro jiné

Výlučnost - uznávají ostatní mou výjímečnost? nejen někam patřit, ale i v daném prostředí vynikat

Příslušnost - Mohu počítat s tím, že někam patřím? někam patřit - rodina, škola, přátelé

Bezpečí - budou mé základní potřeby zajištěny i nadále? zajištění přežití v budoucnosti

Přežití - jsou pokryty mé základní potřeby? Pokrytí základních potřeb - metabolismus, reprodukce

3. . Vysvětlete základní schéma zajišťování efektivity: Procesy-> Zdroje-> Struktury

- máme definované produkty, tyto musíme získat v dostatečné míře a kvalitě s minimálními náklady (peníze, čas, úsilí)
- sledujeme cenu vstupů, hospodaření s nimi během firemních procesů, hodnotu výstupů (první podmínka: hodnota výstupů je větší než cena vstupů)
- hodnota výstupů záleží na užitečnosti.
- zajišťování efektivity probíhá dle schématu *Procesy -> Zdroje -> Struktury*

1. ***Procesy***

– opakovaně probíhající transformace vstupu na výstup
– skládají se z jasně stanovených posloupností aktivit
– jedinečné – komparativní výhoda firmy na trhu
– dobře organizovaná cesta od vstupu k výstupu = popis a řízení firemních procesů (nejprve definice transformace, poté pořadí aktivit a celková podoba procesu)
– i malé zlepšení se může významně projevit

Řízení procesu – tj. jen způsobu transformace – dvoustupňové
– řízení celého procesu - vlastník procesu
– výkon jednotlivých aktivit - vykonavatelé aktivit
– střední úrovně řízení často neefektivní

Řízení zdrojů - zajištění veškerého zázemí včetně péče o lidi
– individuálně se lze věnovat jen omezenému množství lidí
– specializovat se lze pouze na určité typy zdrojů
– počet úroveň řízení dle situace

Tři odlišné druhy procesů
- Ortoprocesy – hlavní procesy, výstupy (produkty) určené vnějším zákazníkům
- Paraprocesy – paralelní podpora a monitoring ortoprocesů, vytvářejí interní produkty (typicky účetnictví, marketing), interní produkty tvoří zdroje a vstupy do dalších procesů
- Metaprocesy – výsledky monitoringu vnitřního a

2. ***Zdroje****:* nespotřebovávají se (na rozdíl od vstupů), pro zajištění procesu

– tvrdé zdroje- materiál, energie, informační systém
– lidské zdroje – vlastnosti, schopnosti, postoje lidí - nositelé jsou konkrétní lidé
– specifické zdroje - systém firemních myšlenek)

Požadavky na vstupy
– z tvrdých zdrojů – většinou jednoznačné a dobře srozumitelné (v IT typicky požadavky na hardware), odvozeny z popisu procesu
– z lidských zdrojů – obtížnější, chování lze předpovědět jen zčásti

Ve fázi zajištění efektivity
– je zásadní definice „tvrdých zdrojů" - lze je organizovat „shora", direktivně - krizový management
– investice do lidských zdrojů nepřináší viditelný účinek (naopak ve fázi budování stability a dynamiky význam lidských zdrojů prudce roste)

Definice lidských zdrojů: *Činnost->Nároky->Požadavky->Člověk*
– činnost – z popisu aktivity (úlohy) – autorem odborník na věcnou stránku procesu (vlastník procesu)
– nároky – co musí člověk, který aktivitu vykonává, zvládnout (definování parametrů – např. znalost programovacího jazyka Java)
– požadavky – určují míru, do jaké je nutné nároky zvládnout (hodnoty parametrů – např. znalost práce s databázemi s využitím rozhraní JDBC) – obecně definuje odborník na lidské zdroje, v IT komplikovanější
– člověk – který konkrétní člověk vyhovuje požadavkům (je nositelem požadovaných vlastností, schopností a postojů) – klíčový vstup pro vlastní výkon dané činnosti – práci, hledáme vhodného člověka pro roli (ne roli pro člověka)

3. ***Struktury***
- odvozeny od jedinečných procesů ve firmě – těžko obecně definovatelné
- odpovědnost za běh procesů, kvalitu zdrojů, informací o procesech, zdrojích ad. -> strukturovanost – definice organizační struktury firmy

4. . Popište princip dvou nezbytných podmínek stability: cyklického řízení a podpory lidí

1. ***Cyklické řízení***
- učení se z vlastních výsledků, zavedení zpětných vazeb, určujeme, čeho chceme dosáhnout a jak, hodnotíme, zda jsme/nejsme úspěšní, provádíme případné korekce
- nutnost zavedení nových prvků do řízení

stanovování cílů (smyslu aktivit) – úkol vedení firmy a cest (způsobů, jak cílů dosáhnout) – úkol řízení firmy, odvozeny od *strategického rámce* (shrnuje klíčové myšlenky, na nichž je postaveno podnikání firmy)

monitorování výsledků – posuzování výsledků aktivit (člověk nebo počítačový systém), porovnávání skutečných a předpokládaných výsledků, poskytování informací o zjištěných odchylkách (pozitivní a negativní zpětná vazba)

Korekční systém – vyhodnocuje důsledky z výsledku aktivit (významnou roli hrají lidé), pozitivní odchylka – potvrzení cílů a cest, negativní odchylka – korekce cílů (cest)

2. ***Podpora lidí*** – cyklický model potřebuje pochopení a podporu ze strany lidí, musí být srozumitelný a přijatelný pro většinu lidí
- Firemní kultura (množina vztahů ve firmě- mezi lidmi, k firemním myšlenkám)

Firma vedená lidmi – nadřízení řeší všechny nerutinní problémy, mají věci pod kontrolou, bývají zavaleni rutinními záležitostmi, převažuje v podmínkách krizového managementu

Firma vedená myšlenkami - - lidé řeší problémy v závislosti na firemních, myšlenkách a cílech, na nadřízené se obracejí méně, vhodnější v rozvojové etapě řízení

5. . Popište princip dvou nezbytných podmínek dynamiky: proaktivního cyklického řízení a aktivity lidí

tvrdá podmínka – cyklické řízení (stabilita) je reaktivní, potřebujeme proaktivní cyklické řízení, doplníme jej o dopředné vazby
- prognózování vývoje – důležitost zdravého rozumu
- ovlivňování vývoje – jeho rychlosti a směru – typicky vyvolávání umělých potřeb

o schopnost učit sama sebe – vrchol dynamiky - tvůrčí charakter procesů, změna vlastního systému zpětných a dopředných vazeb, atakování hranic poznání, nelze se spolehnout na rutinu – nutnost změny firemní kultury

- ***měkká podmínka***

o vyžadována spontánní aktivita lidí, rychlost, pružnost, tvůrčí práce, dobré nápady, generování nápadů není jen věc lídrů či manažerů, postupné zapojování lidí do řešení firemních problémů a rozhodování

o **!** za rozhodování je stále zodpovědný manažer, synergické řízení – směs rozhodovacích stylů užívaných dle povahy problému a vlastností lidí, kteří je řeší, tedy i autoritativní řízení, nejasná pravidla, nejistota, nenaplněná očekávání jsou horší než autoritativní přístup

6. . Popište význam, důležitost a možnost změny následujících lidských zdrojů: vlastností, schopností a postojů

lidské zdroje z pohledu manažera:

- schopnosti – znalosti (co teoreticky zvládnete) a dovednosti (co prakticky umíte) – lidský potenciál, který lze rozvíjet (výsledek vzdělávacích programů)
- postoje – míra snahy a ochoty pracovat, loajalita příslušného člověka, úzce souvisí s motivací, míra disproporce mezi "vím, umím" a "skutečně to dělám" (viz váš týmový projekt), příčiny postojů – zájmy a hodnoty – důležité je, že jsou měnitelné (typicky motivační programy)
- vlastnosti - „temperament“, spíše zděděné rysy osobnosti člověka spojené s biologickou a psychologickou podstatou – takřka nezměnitelné (úzce se týká i vhodného rozdělení rolí v týmu) – člověka s určitými vlastnostmi nemůžete „vychovávat“, ale získat již „hotového“

- změna schopností

o zaměstnanci jsou motivováni, schopnosti jsou však menší než nároky – můžeme snížit nároky, nebo rozvíjet chopnosti (přesně specifikovat cíl a formu vzdělávání) - **habilitace**

o orientace, motivace i individuální schopnosti lidí jsou dostatečné, spolupráce vázne, chybí kvalitní vztahy (kurz týmové spolupráce, změna hodnocení, vliv chování managementu) - **synergetizace**

o konfliktní osobnost člověka a její začlenění do systému, systém nedovoluje člověku se rozvíjet – **integrace**

- příčiny jsou nezávislé, vyplatí se je řešit odspoda – pyramida kultury (efektivní strategie vedení lidí)- podmínka zvládnutí stability a dynamiky firmy

- lidské zdroje jsou nejcennější hodnotou, kterou úspěšná firma disponuje

- změna postojů:

o lidé nemají co sdílet, firemní myšlenky neexistují nebo nejsou dovedeny na úroveň jednotlivce (neexistují úkoly pro jednotlivce) – **definice** - je nutné definovat myšlenky, z nich pro konkrétní lidi konkrétní úlohy

o firemní myšlenky existují, ale lidé je neznají nebo nechápou, lidem je nutné vysvětlit význam jejich práce, a co se od nich očekává – **orientace** – kontext úkolu (smysl) a zadání úkolu

- § co, kdy, jak udělat – míra a způsob zadání závisí na povaze úkolu i úkolovaného – vztah k motivačním typům lidí
- § co, kdy, jak hodnoceno – zpravidla tři témata hodnocení (tzv. 3V)
  - výsledky – výkon (zejména pohyblivá složka mzdy)
  - vývoj – přínos a rozvoj lidských zdrojů (zejména pevná složka mzdy)
  - vztahy – lidé potřebují vědět, že nejsou jenom „jednotka v systému“

o firemní myšlenky a konkrétní úkoly jsou pochopeny, ale nejsou akceptovány – skloubení zájmů firmy a konkrétního zaměstnance (na zájmy lidí je dobré myslet již při definici firemních myšlenek) – motivace

7. . Popište smysl pyramidy kultury a význam jejích jednotlivých pater (základní obrázek budete mít k dispozici)

integrace - zvládnutí konfliktních lidí (Roste?)
Lidé mají prostor k osobnímu rozvoji

synergetizace - rozvoj vztahových dovedností (Sdílí?)
Lidé mají kvalitní vztahy

habilitace - harmonizace skutečených a požadovaných individuálních schopností (Umí?)
Lidé mají odpovídající schopnosti

motivace - harmonizace potřeb lidí a zájmů firmy (Chce?)
Lidé podporují firemní myšlenky
orientace - šíření firemních myšlenek a vysvětlování rolí jednotlivcům (Ví?)
Lidé znají firemní myšlenky
definice - formulace firemních myšlenek a stanovení rolí a úloh jednotlivcům při jejich uskutečňování (Může?)
Firemní myšlenky jsou definovány

8. . Popište princip soutěživého vztahového chování; jakým způsobem dokážete soutěž vyvolat, či naopak potlačit

Vznik, využívání a regulace soutěže (směs ryzího egoismu a ryzího útlaku)
- vzniká při nedostatku a pocitu ohrožení zevnitř skupiny – vyvolán relativním hodnocením – lidé jsou srovnáváni navzájem (odměna patří jen nejlepším)
- přirozená lidská potřeba vítězit, dokazovat vlastní výlučnost – je třeba hledat vhodné příležitosti (týmové vítězství chutná za určitých podmínek lépe než vítězství individuální), zajištění vnějšího tlaku – důležité pro vybití nevyužité ompetitivní energie, aby se spontánně
neprojevovala v rámci skupiny
- vzniká, když lidé plní stejné úkoly – v rámci týmu definujeme různé role – oslabíme soutěžení
- absence nebo nepřijetí společných cílů – roste význam cílů individuálních (v určitých případech může být motorem vývoje skupiny)
- soutěžení je dlouhodobě udržitelnou taktikou mezilidských vztahů, protože nositelé nezapomínají na své zájmy, provázeno napjatými vztahy, vylučuje synergický efekt, při nejasných, a nepochopených pravidlech může přecházet do destrukce
- soutěžení nelze zcela vypudit, ale je možné jej přesměrovat mimo skupinu (soutěžení s jinou skupinou, soutěžení mezi firmami), dlouhodobá úspěšná taktika vnějších vztahů mezi konkurenty a krátkodobá taktika vnitřních vztahů u skupiny, kde chybí dynamika.

9. . Popište princip spolupráce; jakým způsobem dokážete spolupráci podpořit

přesvědčení, že hodnot je dost pro všechny – absolutní hodnocení odměnu mají všichni, kteří dosáhnou výsledku
- zadávání úkolů tak, že uspět mohou jedině všichni – lidé si pak musí pomáhat (opatrně s tím)
- rozlišení rolí ve skupině – více disciplín, ve kterých je možné uspět, zvýší se vzájemná závislost lidí ve skupině
- vliv vnějšího tlaku – spolupracují i ti, kdo se normálně chovají spíše soutěživě
- pocit sdílení – společné cíle přijatelné pro celou skupinu
- spolupráce je dlouhodobě udržitelný vzorec vztahového chování (za předpokladu stejného chování partnera)
- Spolupráce je dlouhodobá žádoucí taktika vnitřních vztahů ve skupinách a vůči vnějším subjektům, se kterými nejste v přímé konkurenci
- spolupráce má obecně menší potenciál než soutěž (spolupracujeme, jen když je to výhodné), při pocitu, že více uspějete v soutěži, volíte zpravidla soutěž, dokud nezvítězíte nebo nezjistíte, že soupeř je těžký a je výhodné spolupracovat

10. Proč se zabýváme požadavky na software? Pro koho jsou sesbírané a zdokumentované požadavky na software užitečné?

Požadavek

- Vlastnosti a parametry softwaru, které definují jeho užitečnost pro zúčastněné subjekty
- Popis toho, co všechno by se mělo implementovat, popisují žádané chování systému a jeho vlastnosti, mohou představovat nějaká omezení procesu vývoje systému

Užitečné zejména pro zákazníky(investoři - projekt financují, chtějí dostat systém, který pokryje jejich podnikatelské potřeby, uživatelé) a zaměstnance(

- Analytik požadavků – sepisuje požadavky na systém a tlumočí je vývojářům (definuje produkt a orientuje vedoucího projektu, vývojáře, testery, dokumentátory,...)
- Vývojáři – navrhují, implementují, udržují systém

11. Vysvětlete pojmy podnikatelské a uživatelské požadavky.

podnikatelské požadavky

- formulují strategický rámec organizace (zákazníka), říkají, proč organizace systém chce (čeho zavedením systému dosáhne)
- Často samostatný dokument – tzv. charta projektu – popisuje vize a rozsah projektu

uživatelské požadavky

- popisují cíle uživatelů a úkoly, které musí být uživatelé se systémem provést (způsob zápisu – případy užití, scénáře, tabulky), např. převést peníze z účtu na účet
- odpovídají procesům z pohledu managementu a vykonavatelů
- mohou být v rozporu s podnikatelskými (pak je nutné komunikovat o cílech projektu a omezeních)

12. Vysvětlete pojmy systémové a funkční požadavky.

funkční požadavky

- procesy z pohledu vývojáře
- popisují softwarovou funkcionalitu – úkol pro vývojáře, co mají naprogramovat

systémové požadavky

- celkové požadavky na systém složený z podsystémů (podsystémy mohou být softwarové i hardwarové, součástí systému i lidé)
- odpovídají zejména definicím zdrojů a struktur

13. Vysvětlete pojmy podnikatelská pravidla a parametrické (mimofunkční) požadavky.

podnikatelská pravidla

- firemní předpisy, státní nařízení, průmyslové standardy, atd. (tato pravidla existují i sama o sobě)
- omezují počet možných uživatelů systému, někdy z nich plynou kvalitativní parametry

parametrické požadavky (často nazývané také jako mimofunkční požadavky)

- požadavky na výkonnost a kvalitativní parametry – použitelnost, přenositelnost, integrita,...

14. Vysvětlete význam podpisu dokumentu specifikace požadavků.

- podpis není jen formální záležitost (zákazníka přesvědčíte, ať si specifikaci opravdu přečte)
- podpis neznamená zmrazení požadavků, ale vytvoření tzv. směrné verze specifikace – verze, která v daném čase představuje závaznou podobu požadavků

15. Uveďte alespoň tři dobré zvyky, které jste použili při psaní specifikace požadavků během vašeho týmového projektu. Vysvětlete význam těchto dobrých zvyků.

- Úplnost – požadavek musí úplně popisovat funkcionalitu
- Správnost – požadavek funkcionalitu popisuje přesně (rozhodnout může jen zdroj požadavku, tj. uživatel, nebo nějaký obecný systémový požadavek), požadavek nesmí být v rozporu se svým nadřazeným systémovým požadavkem
- Proveditelnost – požadavek se musí nechat zrealizovat (zhodnotí vývojář, udělá se specializovaný prototyp)
- Nepostradatelnost – požadavek musí vyžadovat uživatel, vnější systém nebo standard
- Priorita – každý požadavek má svoji implementační prioritu
- Jednoznačnost – jednoduchý, stručný jazyk, omezení možnosti různých interpretací
- Ověřitelnost – testem, prohlídkou

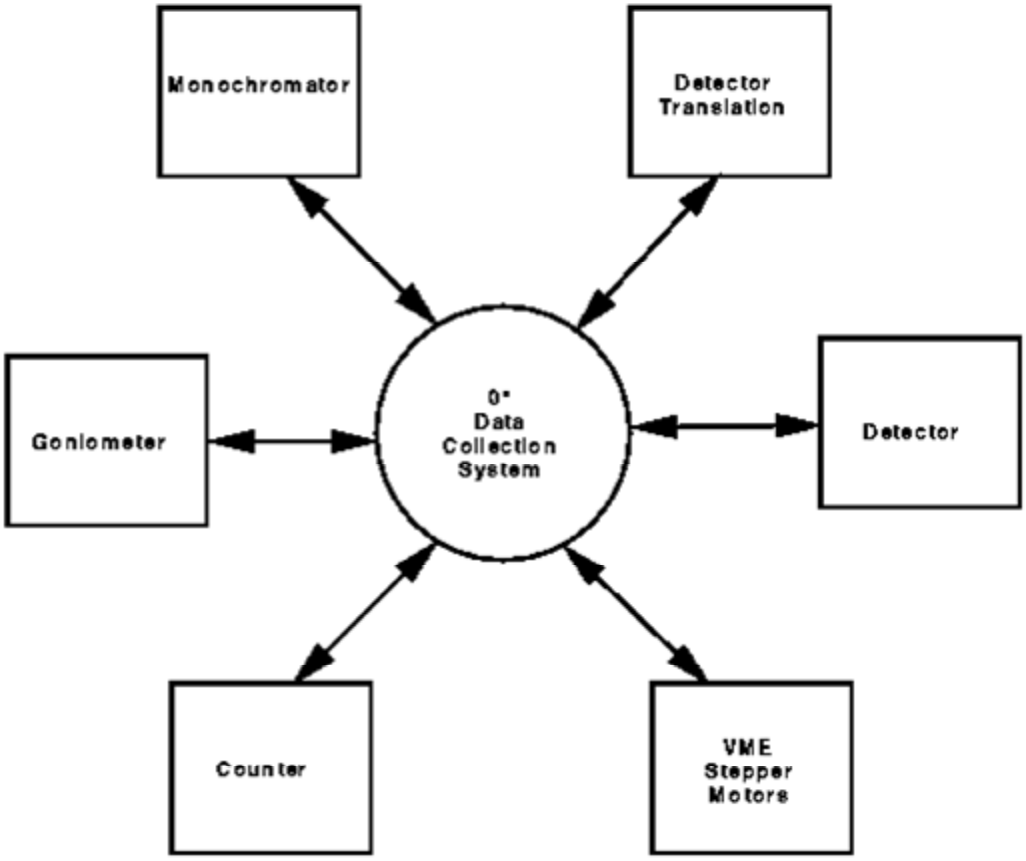
16. Vysvětlete význam uvedení vize a stanovení rozsahu v dokumentu specifikace požadavků

- produktová vize – společný směr všem investorům – k čemu software je a co by se z něj v budoucnu mělo stát vize se mění poměrně zvolna, rozsah se v čase upravuje dle termínů,rozpočtu, zdrojů a kvalitativních omezení – rozsah - každého projektu řádně definován a zároveň podmnožinou vize
- rozsah – která část dlouhodobé vize bude zpracovávána aktuálním projektem, co projekt bude řešit a co už ne (rozsah zároveň definuje omezení)

17. Vysvětlete princip a použití kontextového diagramu a diagramu případu užití. Nakreslete jednoduché příkladové obrázky
    - Kontextový diagram - používá se pro grafický popis rozsahu projektu - hranici a spojení mezi vyvíjeným systémem a okolím

Princip:

uvnitř kruhu celý systém
koncové prvky - obdélníky představují třídy uživatelů, organizace, jiné systémy nebo hardwarová zařízení
šipky - představují datové toky nebo přesuny skutečných předmětů mezi systémem a koncovými prvky



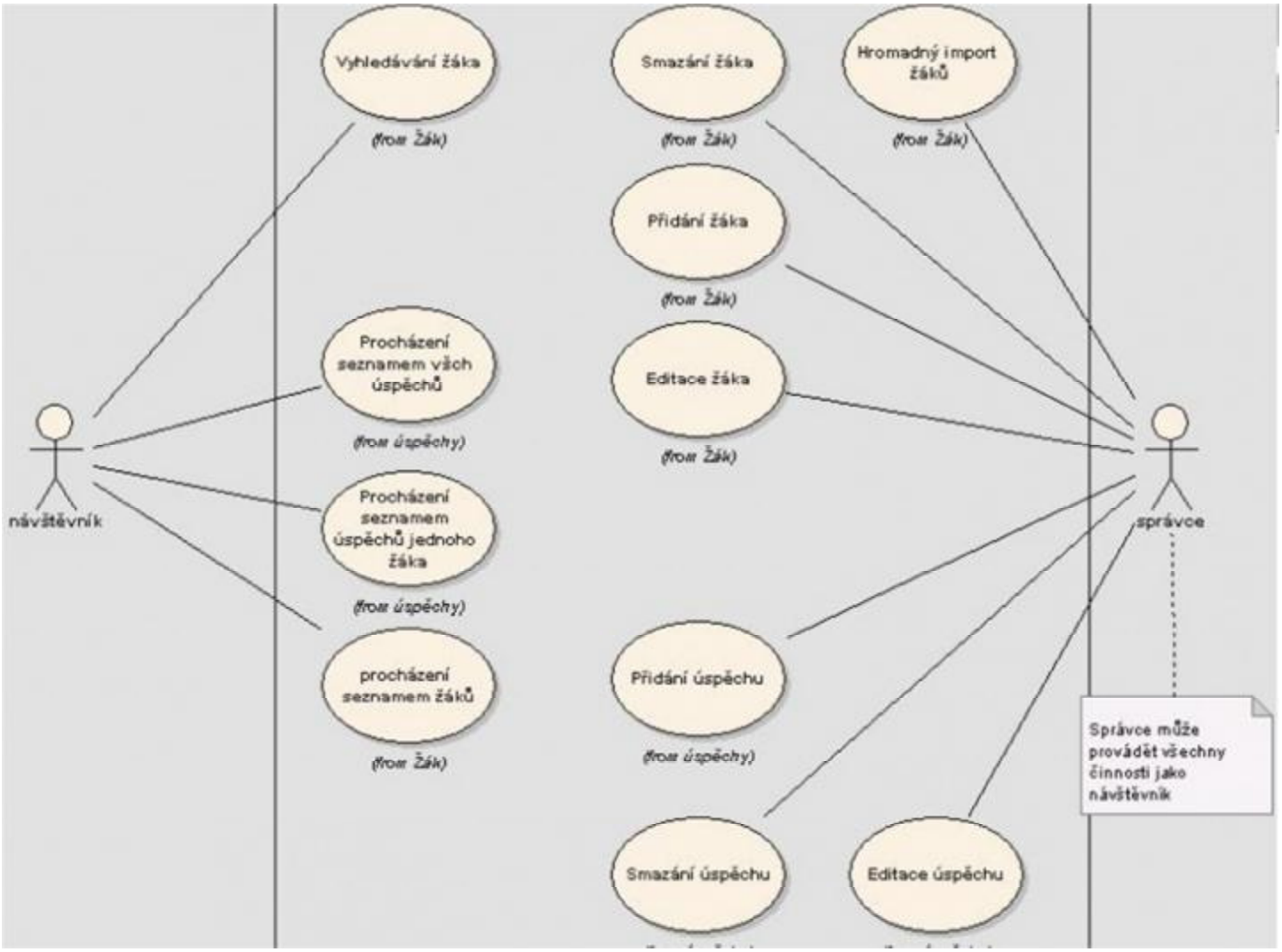
- Diagram případu užití - používá se pro zaznamenání obecných údajů o cílech uživatelů nebo podnikatelských úkolech, které musí uživatelé provádět. Získáme většinou tak, že se uživatelů ptáme na pracovní postupy nebo cíl, který mají, když si k systému sedají (potřebuji udělat to a to…)

Princip:

uvnitř systému - všechny úkoly a činnosti, které má systém provádět

vně systému - uživatelé, kteří se systémem jakkoliv pracují

šipky - od činností k uživatelům, kteří je provádí



18. Jakým způsobem získáte požadavky od uživatelů systému? Uveďte základní praktiky.

- Najít všechny třídy uživatelů systému (např. dle používaných funkcí, frekvence používání systému, úkolů ve firmě,... viz níže)
- Najít zdroje uživatelských požadavků (rozhovory s potenciálními uživateli, dokumentace popisující stávající nebo konkurenční systémy, specifikace systémových požadavků, chybová hlášení, sledování uživatelů při práci,...)
- Vybrat zástupce jednotlivých uživatelských tříd nebo účastníků a pracovat s nimi – tzv. produktoví šampióni (skuteční uživatelé, nikoliv jejich zástupci hrající si na uživatele)
- Dohodnout se, kdo bude rozhodovat o požadavcích (řešení protichůdných požadavků uživatelů, rozhodnout už na začátku projektu)
- Zapojení uživatelů je jediný způsob, kterým se dá vyhnout rozdílům mezi očekáváním a skutečným systémem
- Technicky:
- Používejte slovní zásobu aplikační domény
- Nespoléhejte na to, že všichni účastníci sdílejí společné definice
- Rozhovor o nějaké funkci neznamená, že by se měla v systému objevit
- Schopnost vést diskuse, znát motivy a názory účastníků, pochopit jejich skutečné potřeby
- Otázka proč ve vhodných situacích?
- Ptejte se na výjimky
- Které tři věci nejvíce vadí na starém systému
- Nabízení dalších nápadů a alternativ

19. Roztřiďte následující požadavky (uveďte typ požadavku)

a. Potřebuji nastavit parametry zařízení. - Požadavky na vnější rozhraní / tohle je podle mě spíš případ užití
b. Chceme snížit náklady na údržbu systému o 30%. - Podnikatelský požadavek
c. Slovník lze abecedně setřídit (A-Ž, Ž-A). - Funkční požadavky

d. Každý výrobek má jedinečný čárový kód. - Definice dat
e. Uživatel si vybere, jestli bude vyplňovat tuzemský nebo zahraniční cestovní příkaz. - Požadavky na vnější rozhraní / toto je spíš funkční požadavek
f. Systém musí být dostupný alespoň v 99,0% v pracovní dny mezi šestou hodinou ranní a čtvrtou hodinou odpolední. - kvalitativní požadavek
g. Systém musí být implementován v programovacím jazyce Java. - Omezení

20. Najděte problémy v následujícím zadání požadavku a opravte ho: *„Editor by měl okamžitě zobrazit/schovat všechny formátovací značky.“*

Jedná se o Kvalitativní parametr, které popisují kvalitu chování systému (poznají se podle slov rychle, dobře, jednoduše, spolehlivé, bezpečné, apod.). Výrazy nejsou jednoznačné, jsou subjektivní, je nutné dále pracovat na jejich přesném významu, aby mohla být definována jasná a měřitelná kritéria.
Zde nejspíše špatně slovo okamžitě. Nutno prodiskutovat např. "Kde se editor v systému nachází, aby mohl okamžitě skrýt/zobrazit formátovací značky?" a další doplňující otázky.
Správně: (kdyžtak někdo opravte)
Pokud editor edituje článek, má možnost skrýt/zobrazit všechny formátovací značky (panel s formátovacími značkami).

- *Pokud by toto bylo správně a editor by byla třída uživatelů nejedná se o kvalitativní požadavek!! Je to funkční požadavek!!!*

druhý názor
Jedná se o funkční požadavek, editor je nějaký kus softwaru (wysiwyg editor a podobě) .. a musí okamžitě po načtění skrýt formátovací značky. Nejedná se o třídu uživatelů. Definice požadavku není dostatečně specifická a jednoznačná.
Oprava: Editor musí zobrazit/schovat všechny formátovací značky.

třetí názor

To (nazor 2.) mi jeste neprijde dostatecne jednoznacne, chybi tam typ akce napriklad :"Po zmacknuti tlacitka schovat/zobrazit musi editor okamzite schovat/zobrazit vsechny formatovaci znacky"

21. Popište smysl a náplň čtyř základních aktivit při vývoji sw: specifikace, vývoj, validace, evoluce.

ve všech typech sw procesů (zahrnuje orto-, para-, i metaprocesy) můžeme rozeznat různým způsobem organizované 4 základní aktivity:

- specifikace sw – definice sw produktu
- vývoj sw – vytváření sw splňujícího specifikaci
- validace sw – ověření, že sw dělá, co potřebuje zákazník (systém zpětných vazeb)
- evoluce sw – přizpůsobení sw měnícím se požadavkům zákazníka + nabídka další funkčnosti (systém zpětných a dopředných vazeb)

22. Popište vodopádový model vývoje sw, uveďte jeho výhody a nevýhody.

- původní představa: další fáze začne, když předchozí (úspěšně) skončí (tzv. lineární řízení)
- později: zavedení zpětné vazby, v případě problémů se musíme vracet
- veškeré návraty jsou ale pracné, drahé (zahrnují i přepracování dokumentů) –> po několika iteracích dojde ke zmrazení příslušné části vývoje a přejde se na další fázi (Jak to odhadnout?... zkušenost...) a to znamená, že:

  - případné problémy se odkládají na později, čili jsou fakticky ignorovány

  - zmrazení požadavků vede k tomu, že systém nedělá, co uživatel chce

  - zmrazení designu obvykle vede ke špatně strukturovaným systémům (problémy designu se pak při implementaci obcházejí různými triky, které ztíží další vývoj – tzv. „informační skleróza“)

Nevýhody vodopádového modelu:

- Prodleva mezi zadáním projektu a vytvořením spustitelného systému je příliš dlouhá (zákazník může mít pocit, že se nic neděje)
- Rozdělení projektu do fází je málo flexibilní

- je obtížné reagovat na změny požadavků ze strany zákazníka - proto je model vhodný pouze v krátkých časových intervalech (uživatelé systému zřídka vědí, co přesně chtějí a co vědí, neumějí vyjádřit)
- i kdybychom mohli vyjádřit všechny požadavky, na některé detaily přijdeme až ve fázi implementace
- i kdybychom znali všechny detaily, jako lidé neumíme s tak složitými věcmi nakládat
- i kdybychom uměli nakládat se složitými věcmi, vnější prostředí vede ke změnám požadavků

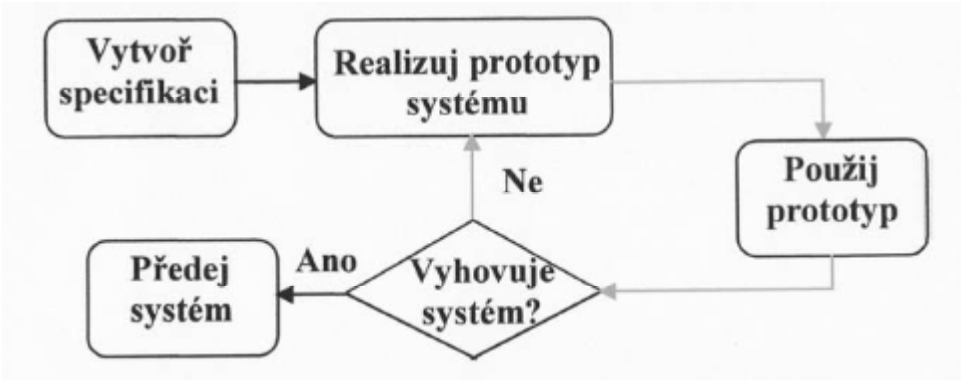
Výhody vodopádového modelu:

- model je sice nedokonalý, ale je lepší mít nějaký model než nechat zvítězit naprostý chaos při řešení projektu
- první návod na vytvoření kostry postupu vývoje
- přehledný – použitelný pro malé projekty, či jako první přiblížení (hrubé zrno)
- výhodný z hlediska organizace práce dodavatele
- občas vyžadován státem v roli zákazníka

23. Popište modely vývoje sw: výzkumník (evoluční prototypování) a prototyp (throw-away prototypování), uveďte vhodnou oblast jejich použití.

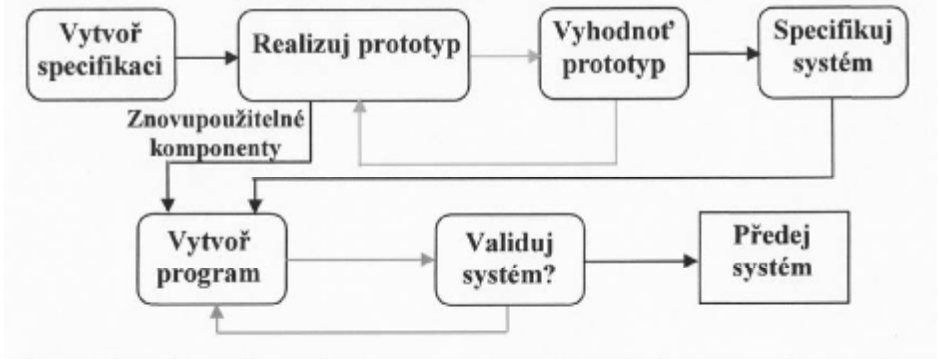
Model výzkumník (evoluční protypování)

- Specifikace, vývoj a validace jsou smíšeny
- Ze specifikace je velmi rychle vyvinut prvotní systém
- Ná základě požadavků zákazníka je systém dále upravován
- cílem je spolupracovat se zákazníkem na zjištění jeho požadavků
- abychom zákazníkovi dodali požadovaný systém
    - vývoj obvykle začíná dobře srozumitelnými částmi systému
    - systém se vyvíjí přidáváním nových vlastností navrhovaných zákazníkem
- během vývoje systému se často vracíme k předchozím etapám - základní charakteristika vývoje



Model prototyp (specifikační prototypování, throw-away prototyping)

- cílem je lepší pochopení zákazníkových požadavků a v důsledku vytvoření lepší definice požadavků (opět důraz na zákazníka)
- tvorba prototypu se zaměřuje na ty části požadavků zákazníka, kterým moc nerozumíme
- prototyp je částečně zavedený produkt se všemi vnějšími rozhraní



Evoluční vývoj SW vhodný pro

- malé systémy
- středně velké systémy s krátkou dobou života

24. Vysvětlete základní principy iteračního a inkrementálního modelu vývoje sw.

Iterativní modely vývoje SW

Základní myšlenky:

- počítají s vývojem požadavků v čase (opět více zaměřeno na zákazníka než vodopádový model) => nutnost několika (i mnoha) iterací procesu vývoje
- podporují myšlenku, že velké systémy je možné rozdělit na menší, je možné používat různé přístupy pro různé části, tzv. hybridní modely

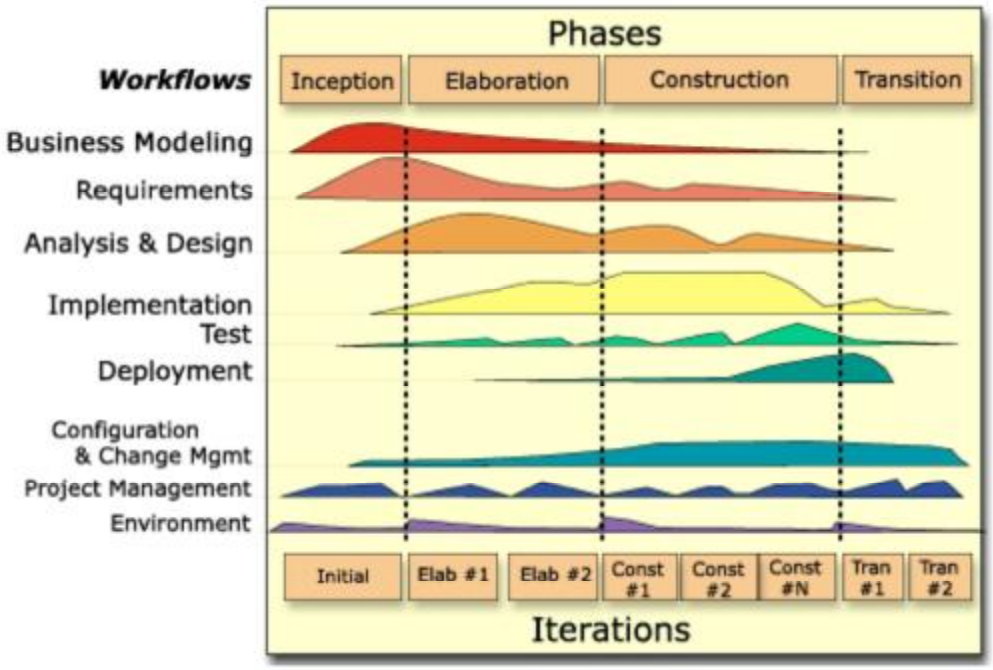
Model inkrementálního vývoje:

- česky často nazýván "přírůstkový model"
- navržen jako způsob, jak omezit přepracovávání částí v důsledku změn požadavků (inkrement = malý přídavek)
- někdy umožňuje zákazníkovi odložit rozhodnutí o detailních požadavcích, dokud nezíská zkušenost se systémem
- inkrement je vytvořen nejvhodnějším procesem vývoje; během vývoje nejsou akceptovány změny požadavků na vytvářený inkrement
- spolu s prvním inkrementem mohou být implementovány společné služby systému (občas lze společné služby vytvářet také inkrementálně)
- paralelně s vývojem může probíhat analýza požadavků pro další inkrementy
- po dokončení inkrementu může být tento předán zákazníkovi, který ho může používat - získá tím část funkčnosti systému
- zákazník může se systémem experimentovat, což mu pomůže upřesnit požadavky na další inkrementy nebo na novější verze předaného inkrementu

25. Popište základní principy metodiky Rational Unified Proces (RUP).

Dodržovány následující principy:

- iterační způsob vývoje software, inkrementální rozšiřování (iterace končí vytvořením spustitelného kódu)
- správa požadavků
- využití existujících softwarových komponent
- vizualizace modelování softwarového systému – standard UML
- průběžné ověřování kvality sw produktu – princip zpětné vazby – existence metrik a provádění měření
- změny systému jsou řízeny – řízení a sledování změn a jejich akceptace lidmi
- vývoj sw produktu v čase (Obrázek) – tzv. vývojový cyklus (vede k vytvoření verze sw produktu, kterou splňuje požadavky na software a lze ji předat zákazníkům)
- čtyři základní fáze (zahájení, rozpracování, tvorba, předání) – každá z fází několik iterací (detailnější rozpracování produktu)
  - zahájení – výsledek je vize koncového sw produktu rámec vývoje sw produktu
  - rozpracování – výsledek je podrobná specifikace požadavků a rozpracovaná architekturasw produktu
  - tvorba – výsledek je kompletní implementovaný a otestovaný sw produkt
  - předání – výsledek je předaný sw produkt (zahrnuje např. beta testování, zaškolení)
- činnosti probíhají souběžně, liší se objem prací v závislosti na fázi



25. Vysvětlete zásadní rozdíly mezi agilním přístupem k vývoji sw a vodopádovým modelem.

Agilní přístup:

- Individuality a interakce mají přednost před procesy a nástroji
- Fungující software má přednost před obsáhlou dokumentací
- Spolupráce se zákazníkem má přednost před sjednáváním smluv
- Reakce na změnu má přednost před plněním plánu

Společné rysy agilních metodik:

- Iterativní a inkrementální vývoj s krátkými iteracemi
- Komunikace mezi zákazníkem a vývojovým týmem
- Průběžné automatizované testování

Rozdíly oproti předchozím modelům:

- explicitně uvažuje rizika projektu - hlavní problém modelu - je závislý na zkušenosti vývojářů najít a realisticky posoudit zdroje rizik (pokud některé riziko podcení, nevyhnutelně nastanou potíže)
- v modelu nenajdeme pevné fáze jako je specifikace nebo návrh – model je obecnější, může různým způsobem zahrnovat ostatní modely SW procesu
- model je aplikovatelný i na jiné typy projektů, než je vývoj SW

26. Jakým způsobem zacházíme s požadavky na software při komponentově orientovaném vývoji sw?

Analýza komponent (prvotní požadavky):

- nejprve vyhledáváme komponenty, které odpovídají specifikaci požadavků
- nalezené komponenty obvykle specifikaci nesplňují přesně, resp. poskytují pouze část požadovaných funkcí
- výsledkem je informace o dostupných komponentách, jejich funkčnosti apod.

Modifikace požadavků:

- provedeme analýzu a modifikaci požadavků tak, aby požadavky odrážely dostupné komponenty
- pokud nemůžeme změnit požadavek, vrátíme se k analýze komponent se snahou najít alternativní řešení

28. Jak byste prakticky postupovali při zavádění modelu vývoje sw?

Důležité je určit pořadí kroků, pokud známe pořadí kroků můžeme vybrat model, který bude sedět na naše požadavky:

- vodopádový model - základní model, konzistentní se strukturovaným programováním shora dolů. Je vhodný, pokud jsou známé požadavky (např.: operační systémy, překladače apod.)
- evoluční vývoj - pokud části požadavků nejsou zřejmé, např. uživatelské rozhraní ("nevím, co chci, ale poznám to, až to uvidím")
- komponentově orientovaný vývoj - máme-li vhodné komponenty
- inkrementální vývoj - potřebujeme omezit přepracovávání, dodáváme systém po částech
- spirálový model - vhodné pro složité dlouhodobé projekty, zahrnuje předchozí modely jako speciální případy (projekt nemusí být SW, ale např. i HW)

29. Co je to projekt? Jaký je rozdíl mezi projektem a procesem?

Projekt

„časově omezená pracovní činnost, jejíž cílem je vytvoření jedinečného produktu, služby nebo dosažení jiného výsledku“. (PMBOK® Guide 2004, strana 5)

- projekt po splnění stanovených úkolů nebo svém ukončení skončí
- vs. běžný provoz nutný pro udržení chodu firmy, instituce
- Má jednoznačně určený cíl – zdokumentovaný ve specifikaci požadavků
- Řešení projektu se postupně vypracovává
- vyžaduje určité, často různorodé zdroje (hardware, software, lidé, ...)
- má hlavního zákazníka nebo zadavatele (investora)
- součástí projektu je neurčitost (obtížné definování cílů projektu, odhad doby potřebné k dokončení, náklady, nemoc, krach dodavatelské firmy)

Proces

- opakovaně probíhající transformace vstupu na výstup
- skládají se z jasně stanovených posloupností aktivit
- jedinečné – komparativní výhoda firmy na trhu
- dobře organizovaná cesta od vstupu k výstupu = popis a řízení firemních procesů (nejprve definice transformace, poté pořadí aktivit a celková podoba procesu)
- i malé zlepšení se může významně projevit

Rozdílů asi moc není. Jeden je v délce trvání a opakování ...

- Procesy jsou většinou dlouhodobé a opakující se, projekty jsou jednorázové.
- Proces je vnitřní organizace firmy, o projektu ví spousta lidí okolo.
- Proces je přesně určen a stanoven, projekt se v průběhu práce mění a přepracovává.

30. Co znamená trojí (čtvero) omezení projektu.

Každý projekt je omezen:

- rozsahem
- časem
- náklady

Nalezení vhodné rovnováhy mezi těmito omezeními (úkol projektového manažera) – nutnost přijímat kompromisy (např. abychom splnili rozsah a čas, musíme zvýšit náklady, nebo abychom splnili termín, musíme zredukovat rozsah a snížit náklady)

Málokdy se podaří dokončit projekty přesně v původně stanovených všech třech omezeních – nutnost rozhodnout se pro nejdůležitější veličinu trojího omezení (např. chceme splnit cíle projektu)

- Otázka kvality (uspokojení zadavatele) – někdy nazývaná čtvrté omezení, jindy považována za nedílnou součást rozsahu, času a nákladů – nutnost dobrého řízení projektu (nezahrnuje pouze splnění trojího omezení)

31. Vysvětlete pojmy Ganttův diagram a struktura rozpisu prací (WBS).

Ganttův diagram

Definuje jaké jednotlivé činnosti se budou během projektu provádět. K těmto činnostem definuje přesný časový interval trvání a udává návaznost jednotlivých činností na sebe. Snahou je zoptimalizovat návaznost jednotlivých prací na sebe.

Struktura rozpisu prací

Definuje jakým činnostem se jednotlivý členové budou během průběhu projektu věnovat a na kolik procent. Snahou je dosáhnout optimálního vytížení každého člena týmu.

32. Co je to konfigurační management a k čemu je dobrý?

Konfigurační management (Správa konfigurace)

- Podpora efektivity a stability
- Uchování smysluplných mezivýsledků jednotlivých aktivit (někdo je bude potřebovat) a jednotlivých podob výsledného sw produktu; podpora systému zpětných vazeb
- Podpůrná aktivita – týká se všech členů týmu během celého vývojového cyklu produktu
- Produkt vyvíjí více lidí, postupně existuje více verzí produktu -> nutně vznikají zmatky -> potřeba zvládnutí těchto zmatků ->nástroje, jak dostat změnu na jednom objektu pod kontrolu
- Proces identifikování a definování prvků systému, řízení změn těchto prvků během životního cyklu, a znamenávání a oznamování stavu prvků a změn, a ověřování úplnosti a správnosti prvků [IEEE-Std-729-1983]
- + sestavování výsledného produktu z jednotlivých částí
- = Software Configuration Management (SCM)

33. Popište princip práce s repository.

sdílený prostor se všemi složkami projektu a řízeným přístupem,

základní operace

- inicializace – vytvoření úložiště, naplnění inicializační verzí projektu
- check out (update) – zkopírování prvku konfigurace do lokálního pracovního prostoru
- check in (commit) – uložení prvku do konfigurace do repositoty
- zjišťování stavu – sledování změn úložiště vůči lokálnímu pracovnímu prostoru

34. Co je to UML a k čemu ho použiji.

UML Otevřený rozšiřitelný standard pro vizuální modelování

- Není to metodika (to znamená, že nestanovuje proces)
- Metodika, která doplňuje UML (kromě jiných) je Unified Process + jeho obměny
- Modelování jako svět vzájemně se ovlivňujících objektů obsahujících informaci + chování
- Modely v UML obsahují:
  - Statickou strukturu (jaké objekty jsou důležité a jak spolu souvisejí)
  - Dynamické chování (vzájemná spolupráce objektů)
- podporuje model-driven architecture - předpoklad, že model specifikujeme dostatečně přesně, abychom z něj mohli vygenerovat kód
- dá přizpůsobit konkrétním jazykům a prostředím pomocí tzv. profilů - ve specifikaci UML2 jsou uvedeny příkladové profily pro J2EE/EJB a .NET komponenty

35. Vysvětlete základní princip a použití diagramu případů užití.

Princip:

spočívá v tom, že se pro popis kontextu systému a popis uživatelských požadavků použije diagram případů užití. jsou funkce, které systém vykonává jménem jednotlivých účastníků nebo v jejich prospěch

Diagramy případů užití reprezentují vnější pohled na systém, modelují zamýšlené fce systému a jeho vztah k okolí

Použití

- Nalezení hranic systému

- Vyhledání účastníků (aktérů)

- Nalezení případů užití

- Specifikace případu užití

- Tvorba scénářů

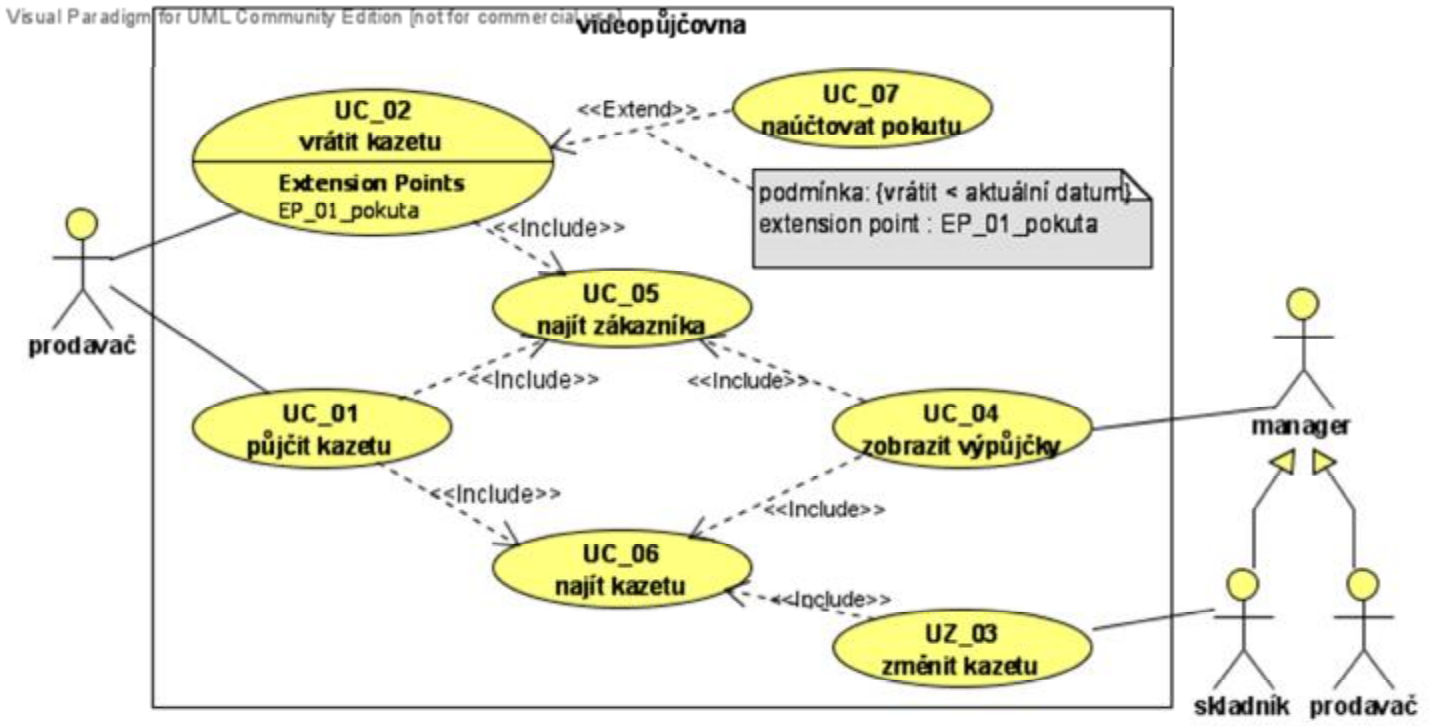
Model případů užití – 4 komponenty

- Aktéři – role přidělené osobám nebo předmětům používajících daný systém

- Případy užití – činnosti, které mohou aktéři se systémem vykonávat

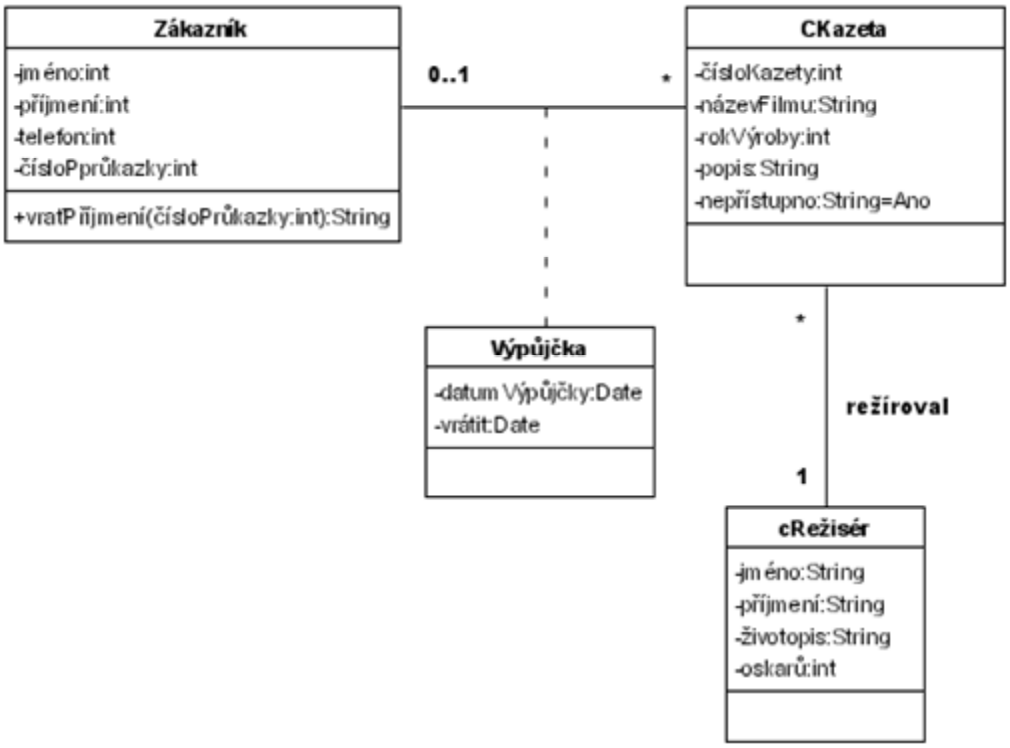
- Relace – smysluplné vztahy mezi aktéry a případy užití

- Hranice systému – ohraničení kolem případů užití, vymezení modelovaného systému



36. Vysvětlete základní princip a použití diagramu tříd.

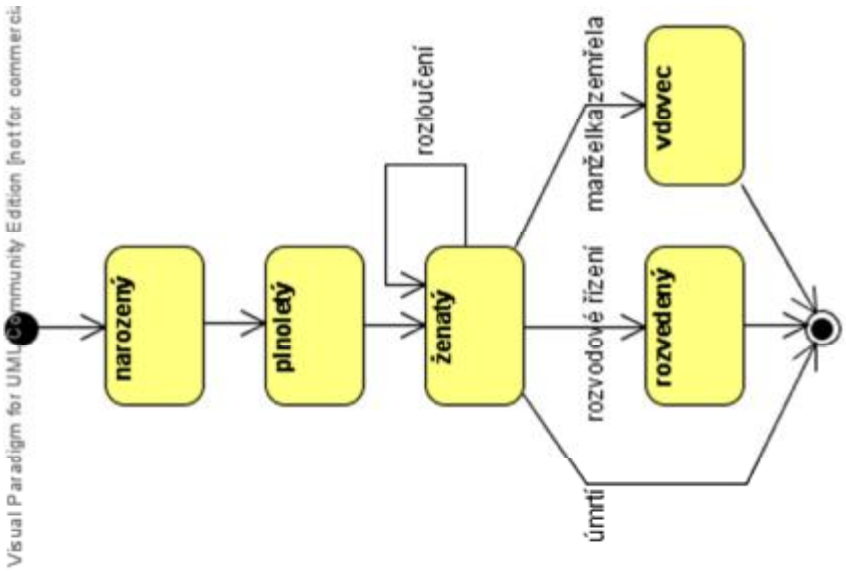
- ukazuje statickou strukturu tříd v systému, tj. třídy, jejich vztahy (dědičnost, asociace, závislost), atributy a operace
- diagram je považován za statický proto, že struktura popisovaná diagramem platí v jakémkoli okamžiku běhu systému



37. Vysvětlete základní princip a použití stavového diagramu.

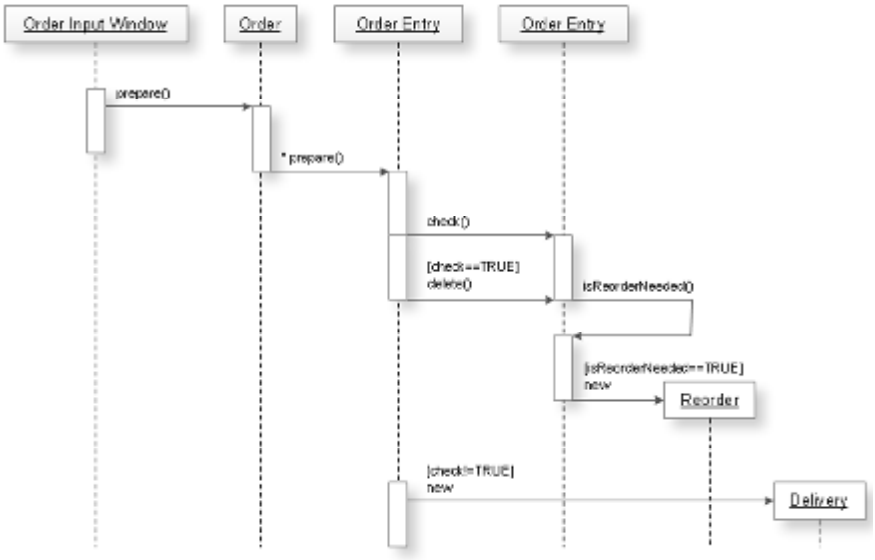
Stavový diagram

- typicky se používají pro popis chování instance třídy, někdy také pro případy užití nebo pro celý systém nebo podsystém
- ukazuje, jakými vztahy mohou instance třídy procházet během svého životního cyklu a jaké události mohou způsobit přechod mezi stavy
- událost může být způsobena zasláním zprávy objektu, např. pokud uplyne specifikovaná doba nebo může přechod nastat po splnění nějaké podmínky
- s přechodem může být spojena nějaká činnost objektu
- proti klasickým "plochým" stavovým diagramům umožňují stavové diagramy v UML strukturování



38. Vysvětlete základní princip a použití sekvenčního diagramu.

Sekvenční diagram
- ukazuje dynamickou spolupráci mezi množinou objektů
- má časovou osu - plyne shora dolů v diagramu
- diagram ukazuje posloupnost zpráv zasílaných mezi objekty
- v některých metodikách (např. OOSE) se k případům užití vytvářejí sekvenční diagramy místo diagramů spolupráce



39. Vysvětlete základní princip a použití alespoň jednoho z následujících diagramů: diagram aktivit, diagram spolupráce, objektový diagram.

Diagram spolupráce

- podobně jako sekvenční diagram ukazuje výměnu zpráv
- na rozdíl od sekvenčního diagramu nemá časovou osu, zato ukazuje vztahy mezi objekty (linky); interakci mezi objekty znázorňují oba diagramy, tj. pokud je důležitá časová posloupnost, používá se sekvenční diagram, a pokud je třeba znázornit kontext, použijeme komunikační diagram
- popisují komunikaci mezi objekty nebo jejich rolemi
- mohou být připojeny např. k případu užití jako jeho popis (
  - popisují, které objekty nabízejí chování popisované případem užití
  - jak objekty případ užití vykonávají)
- mohou mít i vyšší úroveň podrobnosti, lze je použít např. pro popis chování operací třídy apod.
- v diagramu spolupráce se vyskytují:
  - objekty účastnící se interakce, případně role objektů v interakci
  - spojení pro přenos zprávy - kreslí se jako plná čára - často se kreslí s šipkou ukazující průchodnost
  - zprávy (stimuly) - kreslí se jako šipka blízko čáry - šipka s plnou hlavou znamená volání procedury, volající pokračuje až po návratu z procedury
- každá zpráva má pořadové číslo, končící dvojtečkou
- zprávy na nejvyšší úrovni jsou číslovány postupně 1, 2, 3 atd.
- zprávy na další úrovni vnoření během stejného volání 1.1, 1.2, 1.3 atd.
- za pořadovým číslem může být uvedeno: zpráva (argumenty) nebo případně návratová_hodnota := zpráva(argumenty)
  - v diagramu spolupráce lze znázornit také iterace a podmínky
  - iterace - jako pořadí zprávy uvedeme hvězdičku (pokud nechceme uvádět podrobnosti) nebo např. *[i := 1..n] (pokud chceme iteraci popsat)
  - podmíněná zpráva - jako prefix uvedeme podmínku. např. [x>0]
  - zprávy je možné také "číslovat" identifikátorem

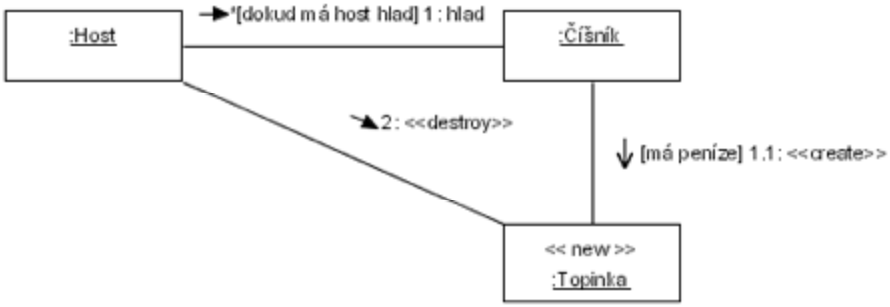
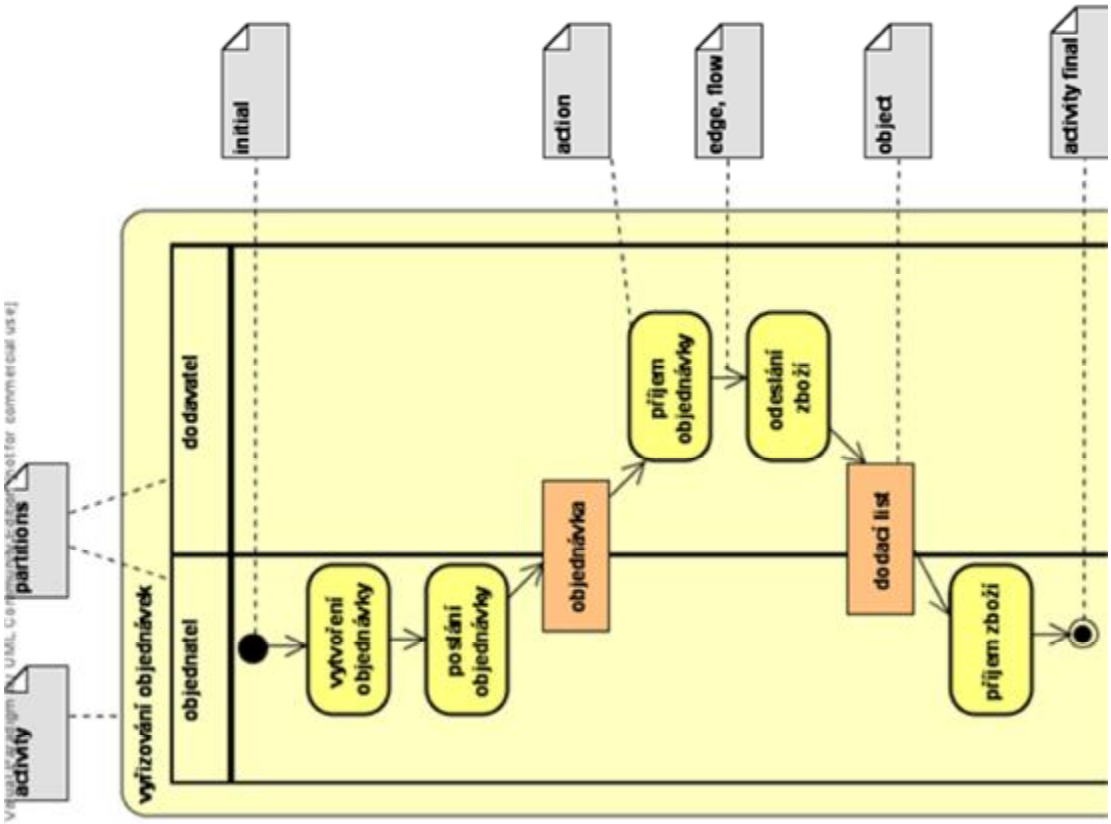


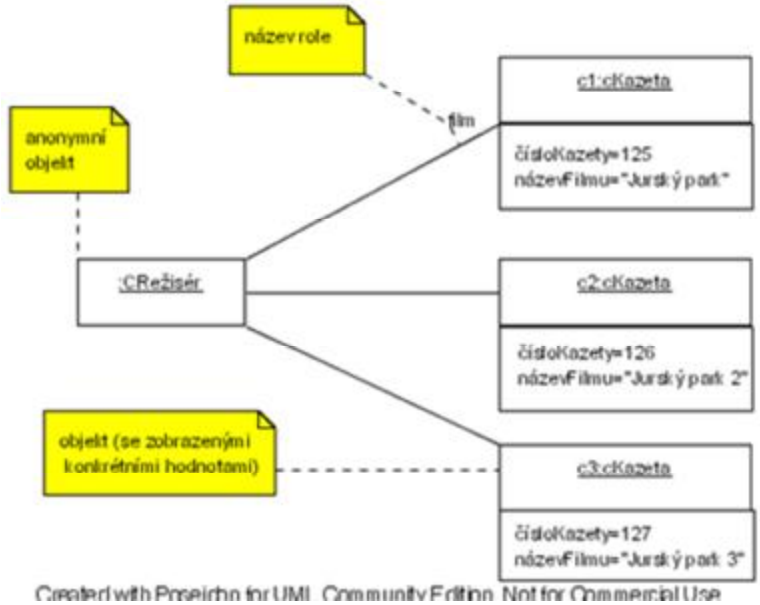
Diagram aktivit

- založen na notaci Petriho sítí
- znázorňuje sekvenční tok aktivit v rámci procesu
- umožňují modelovat paralelní chování, zpracování výjimek atd.
- při modelování se obvykle používají pro různé účely
    - pro popis akcí, které se budou vykonávat v průběhu operace, tj. popis interní činnosti objektu
    - pro popis příbuzných akcí a jakým způsobem ovlivní okolní objekty
    - pro znázornění instance případu užití
    - pro znázornění chodu firmy, tj. aktérů, organizace práce a objektů
- stavový diagram a diagram aktivit se doplňují (stavový diagram popisuje objekt a změnu jeho stavů v rámci probíhání procesů, diagram aktivit popisuje tok aktivit a jejich vzájemnou závislost v rámci procesu)



Objektový diagram (Objektový diagram zobrazuje objekty a jejich spoje (links) v jednom okamžiku, tj. je to snapshot běžícího systému (či jeho části). Objekty jsou instancemi tříd, linky jsou instancemi asociací)

- je variantou digramu tříd, diagram objektů ukazuje konkrétní instance a jejich vztahy (linky neboli propojení)
- ukazují možný vztah objektů v nějakém okamžiku běhu systému
- používá se poměrně zřídka, vhodné pro vysvětlení malých částí systému se složitými (zejména rekurzivními) vztahy



40. Vysvětlete základní princip a použití alespoň jednoho z následujících diagramů: diagram nasazení, diagram komponent.

Diagram nasazení (Diagram nasazení ukazuje rozmístění zdrojů (např. HW) a softwarové komponenty, procesy a objekty, které na nich žijí.)

- ukazuje fyzickou architekturu hardwaru a SW v systému, např. počítače, zařízení, jak jsou propojeny, jaké artefakty budou umístěny na kterých počítačích
- zobrazuje uzly, komunikační cesty a nasazené artefakty

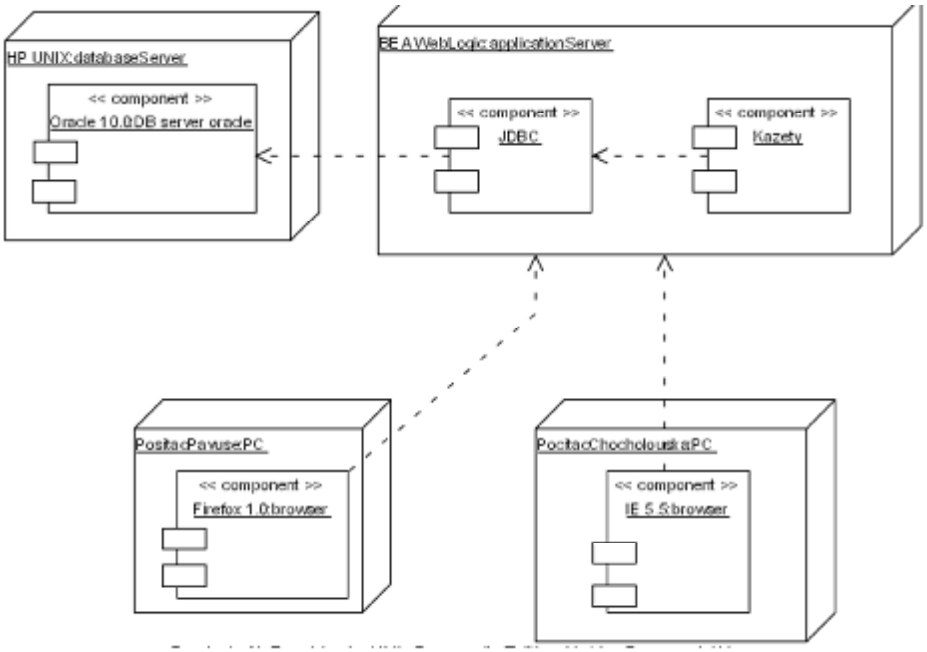
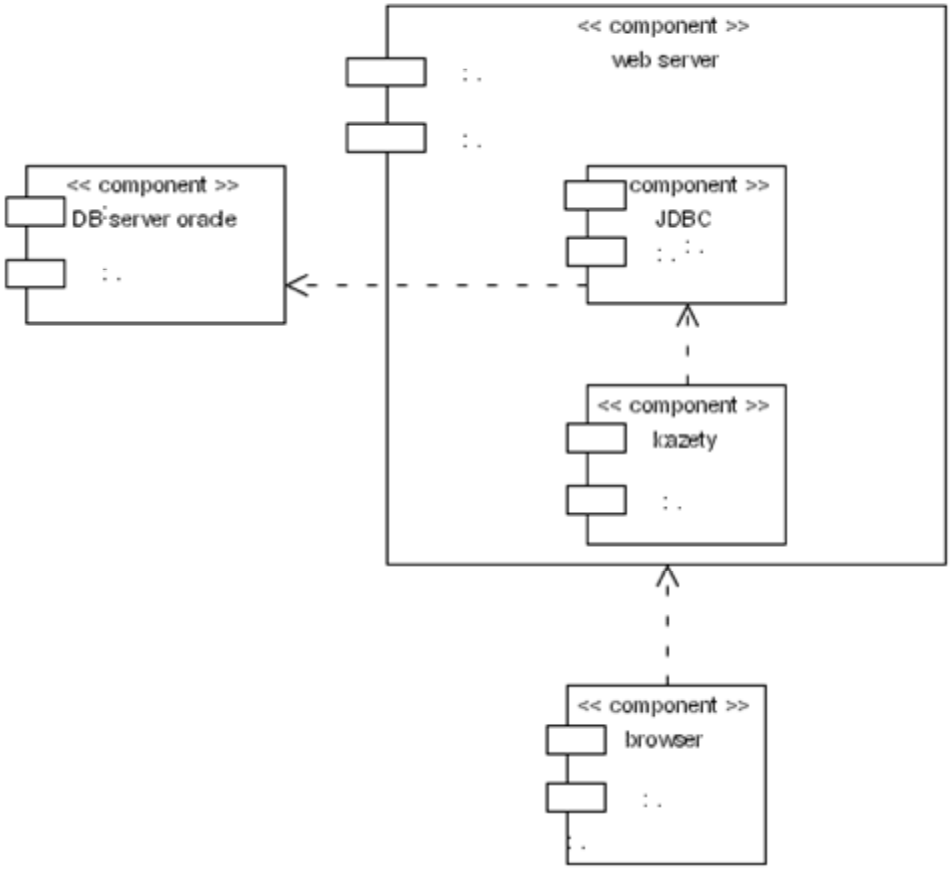


Diagram komponent

(Komponenta je fyzická nahraditelná část systému, která obaluje implementaci a poskytuje realizaci množiny specifikovaných rozhraní. Diagram komponent znázorňuje softwarové komponenty použité v systému. Mohou to být jak komponenty vytvořené vlastními silami, tak komponenty zakoupené od třetích stran.)

- ukazuje organizaci a závislosti mezi komponentami nebo mezi komponentami a rozhraními komponent
- komponenty reprezentují dobře zapouzdřené prvky logické architektury
- komponenta zapouzdřuje implementaci a zveřejňuje množinu rozhraní
- komponenta může být implementována např. jedním nebo více spustitelnými soubory, zdrojovými texty nebo objektovými moduly, knihovnami, příkazovými soubory apod. - tyto jsou modelovány jako artefakty



41. Vysvětlete pojem analytický model, popište, jak vypadá analytická třída

Analytický model = (Výstup objektové analýzy)

= co má systém dělat - zachycuje podstatné požadavky a charakteristické rysy požadovaného systému; můžeme jej modelovat v UML

- Vždy v jazyku domény
- Zachycuje problém z určité perspektivy (nebezpečí zabývání se přílišnými detaily)
- Obsahuje artefakty modelující problémovou doménu
  - Analytické třídy – klíčové pojmy v doméně, vyjadřují velmi přesně definované zobecnění entit problémové domény
  - Realizace případů užití – vzájemná komunikace instancí analytických tříd
- Vypráví příběh o požadovaném systému (používáme vhodné diagramy)
- Je užitečný pro maximální počet zainteresovaných osob

Analytická třída

- Obsahuje množinu hlavních kandidátů na atributy a množinu hlavních operací
- Z názvu je jasný její účel
- Modeluje jeden specifický element problémové domény
- Má definovanou malou a jasnou množinu (ideální počet je 3-5) odpovědností (odpovědnost je dohoda-závazek vůči klientům, který představuje sémanticky soudržná množina operací)
- Je vysoce soudržná (má jeden jedinečný účel)
- Má málo vazeb na okolní třídy
- Špatně navržená analytická třída
  - Má pouze jedinou operaci – tzv. funktoid
  - Má více než 5 odpovědností
  - Je všeobjímající (jmenuje se např. systém, software, apod.)
  - Obsahuje široce rozvětvený strom dědičnosti
  - Je málo soudržná
  - Má úzké vazby na příliš mnoho dalších tříd

42. Popište principy metod hledání analytických tříd (podstatná jména, slovesa, CRC karty, typické zdroje)

- Shromažďování a následná analýza podstatných jmen (kandidáti na třídy a atributy) a sloves (kandidáti na odpovědnosti a operace)
- CRC karty (Class/Responsibilities/Collaborators) – spontánní diskuse + hledání nápadů (forma brainstormingu) nebo procházení scénářů případů užití
  - pro třídy vytvoříme kartičky, nahoru napíšeme jméno třídy, vlevo zodpovědnost třídy, vpravo spolupracující třídy
  - proces:
    - § pojmenování doménových konceptů, uvedení odpovědností jednotlivých konceptů, označení spolupracujících tříd, vše – nejprve forma brainstormingu, poté analýza sebraných informací
    - § NEBO (bez fáze brainstormingu): procházíme scénáře případů užití, přidáváme třídy a nové odpovědnosti existujícím třídám; pokud neumíme, rozdělíme existující třídu na dvě nebo založíme novou
      - § nevýhoda - vazby mezi třídami nejsou znázorněny graficky (snažte se proto minimálně např. lepit lístečky na tabuli a kreslit mezi nimi spojovací čáry)
- přemýšlení o dalších typických zdrojích (fyzické objekty – auto, budova, výtah,…), kancelářské a obchodní záležitosti (faktura, objednávka, smlouva,…), známá rozhraní k vnějšímu světu (monitor, klávesnice, port,…) – důležité zejména u embedded systémů, koncepční entity, které nejsou uvedeny jako konkrétní fyzické předměty (věrnostní program)
- První analytický model- výsledek vzájemného porovnání výsledků jednotlivých metod

43. Vysvětlete pojmy asociace, spojení, závislost.

Asociace

Asociace mezi dvěma třídami je totéž, jako kdyby měla jedna třída pseudoatribut, který nese odkaz na objekt druhé třídy (asociace používejte vždy, když je na druhém konci asociace důležitá třída, jejíž přítomnost chcete zdůraznit)

= spojení tříd v modelu

Spojení

Vazba mezi objekty je spojení (jeden objekt obsahuje odkaz na jiný objekt), příslušná vazba mezi třídami je asociace (spojení je instance asociace)

= spojení objektů (instancí tříd) v modelu

Závislost

Závislosti jsou relace, kde se změna v dodavateli automaticky projeví rovněž v klientovi (tečkovaná šipka od klienta k dodavateli); univerzální stereotyp <<use>> - klient používá dodavatele jako argument, návratovou hodnotu nebo jako element vlastní implementace (např. vytvoření dočasného objektu)

44. Vysvětlete pojmy asociace, agregace, kompozice.

Asociace

Asociace mezi dvěma třídami je totéž, jako kdyby měla jedna třída pseudoatribut, který nese odkaz na objekt druhé třídy (asociace používejte vždy, když je na druhém konci asociace důležitá třída, jejíž přítomnost chcete zdůraznit)

Agregace

Agregace představuje volnou vazbu mezi celkem a součástí, kdy jeden objekt (celek) využívá služby dalších objektů (součástí). Například vztah mezi počítačem a tiskárnou je vztah typu agregace, kdy počítač s tiskárnou tvoří jeden celek, ale tiskárna může existovat i tehdy, pokud není k žádnému počítači připojena. Agregace je formou asociace a v grafické podobě se odlišuje prázdným kosočtvercem na straně celku.

Vztah : Pocitac <- Tiskarna

Kompozice

Silnější formou agregace je kompozice. Jde opět o vztah mezi celkem a součástí, ale tento vztah je velmi těsný a neumožňuje samostatnou existenci součásti, aniž by byla připojena k nějakému celku. Navíc na rozdíl od agregace tato součást musí patřit jen jedinému celku a není možné ji sdílet více celky.

Vztah : Faktura <- Polozka

45. Vysvětlete pojem realizace případu užití, co podstatného zahrnuje.

Realizace případů užití

- Explicitní znázornění spolupráce skupin objektů pro dosažení požadovaného chování systému -> diagramy interakce (diagram spolupráce a sekvenční diagram) ukazují, jak třídy a objekty plní požadavky specifikované v případech užití
- Dynamický pohled na systém
- Spolupráce instancí analytických tříd
- Každá realizace zachycuje právě jeden případ užití
- Skládá se z následujících součástí
  - Diagram analytických tříd, které „vypravují“ příběh o případu (případech) užití
  - Diagram interakcí – spolupráce skupin objektů pro dosažení chování případu užití
  - Speciální požadavky odhalené v průběhu realizace případu užití
  - Upřesňování (změna) případu užití
- Digramy obecné interakce ukazují role klasifikátorů a asociací, zprávy a tok zpráv
- Diagramy konkrétní interakce ukazují instance klasifikátorů, spojení, tvorbu a uvolnění instancí a spojení, zprávy a tok zpráv, iterace a větvení

- Digram spolupráce (zdůrazňují statické relace mezi instancemi a role jednotlivých instancí) a sekvenční diagram (popisuje chronologicky uspořádanou posloupnost zpráv předávaných mezi instancemi) jsou považovány za téměř zaměnitelné
- Modelování všech typů procesů – diagram aktivit – zachycuje jeden specifický aspekt chování systému – je to objektově orientovaný vývojový diagram

46. Vysvětlete rozdíl a návaznost mezi analýzou a návrhem softwaru.

Analýza = souhrn pracovních postupů a metod, které jsou nutné k vyřešení nějakého problému (obecnější pohled).
Hranice mezi analýzou a návrhem je tenká.
Návrhem softwaru rozumíme už konkretní řešení problému. Opět je to ale zase nějaký souhrn postupů.

47. Vysvětlete pojem rozhraní, jak rozhodnete ve fázi návrhu o vhodnosti existence rozhraní.

Rozhraní

- odděluje specifikaci od implementace, lze jej připojit k třídám, podsystémům, komponentám...(definuje služby poskytované těmito entitami)

Jestliže ve fázi návrhu dojdeme k bodu kdy třeba definujeme třídy které mají stejný nebo podobný účel či roli, je vhodné implementovat rozhraní.

Nebo Intuitivne: Dalo by se chapat jako jakysi prostrednik mezi tridami se stejnymi metodami. Pouzijeme ho tehdy kdyz vice trid pouziva stejnou metodu nebo ocekavame ze ji vice trid bude pouzivat v budoucnosti.

48. Vysvětlete rozdíl mezi objektově orientovanou analýzou (návrhem) a strukturovanou analýzou (designem), na základě čeho byste rozhodovali, kdy kterou použít.

Objektově orientovaný návrh

- přesné určení implementace funkcí specifikovaných v analytickém modelu
- primární modelovací aktivita v poslední etapě fáze Rozpracování a v počátcích fáze Konstrukce
- analýza a návrh probíhají do jisté míry současně

Strukturovaná analýza

- strukturované metodiky pro analýzu a návrh systému historicky předcházely objektovým metodikám
- na funkce a na data se zaměřují víceméně odděleně
- odpovídá strukturovanému programování

Jinak řečeno hlavní rozdíl spočívá ve využití, objektový návrh použijeme tam kde je projekt rozsáhlejší a kde se předpokládá určitá budoucí modifikace. Kdežto strukturovaná analýza se hodí pro malé projekty u kterých se předpokládá krátká doba života(malé programy, prototypy apod.)

49. Vysvětlete pojmy diagram datových toků (DFD) a model kontextu systému (nakreslete obrázek).

DFD

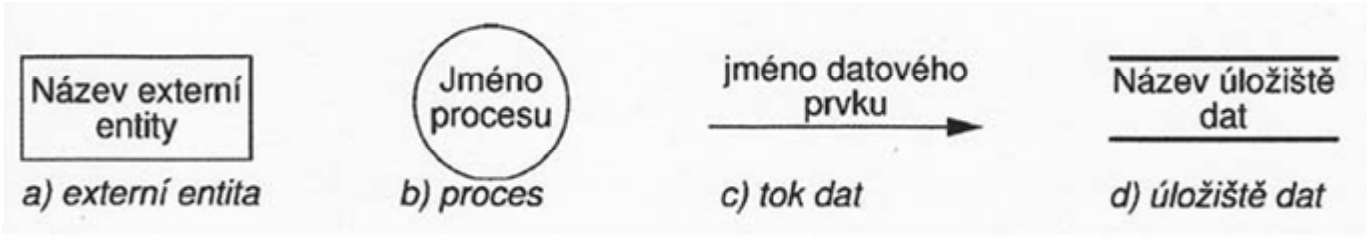
- diagram (data flow diagram)
- data jsou zpracovávána posloupností kroků (kroky provádějí lidé, funkce programu)
- použití pro reprezentaci systému libovolné abstrakce.

DFD úrovně 0 = fundamentální model systému, model kontextu systému

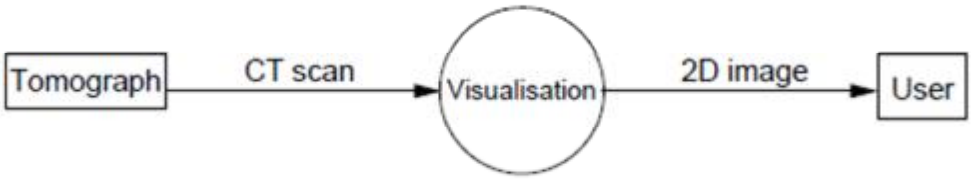
- celý SW systém je zakreslen jako jedna bublina, má jeden nebo více vstupů a výstupů

DFD dalších úrovní

- systém rozdělíme do menších částí a znázorníme na větší úrovni podrobnosti (model je postupně zjemňován)
- vytvořené DFD by neměly být příliš velké - měly by se bez problémů vejít na papír velikosti A4



Př. Vizualizace snímků z tomografu (vstup - 2D řezy tělěm pacienta, výstup -2D pohled snímek)



Obrázek 58 DFD úrovně 0

- **Model kontextu systemu**
- - hranice systému, tj. co bude tvořit systém a co bude okolí systému (již ve specifikaci požadavků)
- - hranice vytvářeného systému - kolečko s vepsaným názvem systému
- - lidé, organizace nebo jiné systémy, se kterými náš systém komunikuje (terminátory) - pojmenovaným obdélník
- - vstupující a vystupující data - šipky mezi systémem a terminátorem

50. Vysvětlete rozdíly mezi klasickou DeMarcovo metodologií a Yourdonovou "moderní strukturovaná analýzou".

DeMarcova i Yourdanova metodologie jsou metodiky strukturované analýzy.

Klasická DeMarcova metodologie

- vstupem uživatelské požadavky, výstupem tzv. strukturovaná specifikace
- systém je specifikován pomocí DFD, uvedeny podstatné procesy, paměti a údaje
- případné jednodušší procesy v DFD nižší úrovně
- elementární procesy zapsány v pseudokódu, rozhodovací tabulkou nebo stromem
- v datovém slovníku popis dat
- vytváření 4 modelů systému
  - fyzický model starého systému(analytik zmapuje stávající systém,strukturu, data..)
  - logický model starého systému(vytvoření logického z fyzického, zrušení implement. detailů, model systému za ideálních podmínek a technologií)
  - log. model nového systému(přidání nových funkcí)
  - fyz. model nového systému(návrh implementace)

Moderní strukturovaná analýza

- místo čtyř modelů systému se zaměřuje přímo na nalezení esenciálního modelu systému, tj. logického modelu nového systému

- kroky
  - vytvoříme model prostředí (definuje hranici mezi systémem a zbytkem světa)
  - vytvoříme seznam událostí
  - na základě událostí vytvoříme předběžný model chování systému
  - model chování systému přestrukturujeme do konečného modelu chování (= esenciální model systému)
  - vytvoříme uživatelský implementační model (doplňuje esenciální model o informace nutné pro implementaci modelu)
  - konec analýzy, následuje návrh architektury, podrobný návrh a kódování

Jinak řečeno rozdíl spočívá v samotné analýze. Moderní strukt. analýza se snaží o přímé nalezení nového systému pomocí modelu chování. Kdežto DeMarc analýzu rozdělí do 4 modelů systému a nový systém hledá přes mapování a analýzu starého systému.

51. Vysvětlete pojmy centrální transformace a strukturogram.

Centrální transformace

= slouží pro převod DFD do hierarchického stromu, kdy není tak jenoduché odhadnout transformaci. Hledáme centrální body, které lze zjednodušit a nahrazujeme je pomocí "šéfa".

Předpokládejme ideální svět a proces, při kterém odsekáváme nepotřebné vstupní(vstupy musí být bez chyby) a výstupní proudy(výstupy nemusí být formátované). To co zbyde je **centrální transformace.**

Strukturogramy

- jsou to grafy
- ukazují rozdělení systému do hierarchie bloků a jejich vzájemnou komunikaci
  - blok představuje podprogram programovacího jazyka; blok se znázorňuje jako obdélník obsahující název bloku
  - knihovní procedury a funkce (předdefinované bloky) se znázorňují pomocí zdvojení svislých čar obdélníka
  - volání podprogramu je znázorněno obyčejnou šipkou (od volajícího k volanému)
  - předávání zpracovávaných dat je znázorněno šipkou s prázdným kroužkem (šipka směřuje od odesilatele k příjemci dat)
  - předávání příznaků je znázorněno šipkou s plným kroužkem

52. Zhodnoťte význam návrhu architektury sw systému pro úspěšnost výsledného sw systému, popište, jak budete principiálně postupovat při návrhu architektury.

Význam spočívá v tom, že návrh architektury umožní včasné odladění produktu a umožní dobrou udržovatelnost. Pokud návrh následuje po analýze, pak umožní dělit navrhovaný systém do podsystému jejichž návrh pak může probíhat nezávisle.

Postup

- rozdělení systému do podsystémů
- rozdělení do vrstev a oddílů (partitions)
- návrh topologie systému
- identifikace paralelismu, alokace na uzly a volba komunikace
- volba způsobu řízení

53. Vysvětlete rozdíly mezi dvěma základními vztahy mezi podsystémy (klient-poskytovatel, peer-to-peer).

klient-poskytovatel (client-supplier)

- klient volá poskytovatele, který vykoná nějakou službu a pošle odpověď; poskytovatel nemusí znát rozhraní klienta

peer-to-peer

- oba podsystémy mohou volat druhý, tj. oba musejí znát rozhraní toho druhého, je komplikovanější, mohou v něm vznikat obtížně srozumitelné komunikační cykly => snažíme se o vztah klient-poskytovatel, kdekoli je to možné

54. Vysvětlete pojmy vrstva a oddíl, uveďte příklad vrstvené architektury

Vrstva

Základní stavební prvek určité architektury, každá vrstva poskytuje data či výstupy vrstvě, která je nad ní.

Oddíly

Rozdělují systém vertikálně na nezávislé nebo slabě svázané podsystémy, každý z nich poskytuje jiný typ služeb(operační systém obsahuje ovladače pro jednotlivé typy zařízení).

Příklady

- model OSI-ISO (7mi vrstvý model, znáte ze sítí)
- 3-vrstvý aplikační model (znáte z UUR nebo JXT - datová vrstva, prezentační, aplikační...)

55. Popište rozdíl mezi uzavřenou a otevřenou vrstvenou architekturou.

uzavřené (striktně vrstvené) - vrstva je implementována pouze prostředky nejbližší nižší vrstvy

- omezuje závislosti mezi vrstvami = princip skrývání informací
- dovoluje snadnější změny rozhraní - změna ovlivní jen nejbližší vyšší vrstvu
- například síťové modely (sedmivrstvý ISO/OSI model)

otevřené - může používat prostředky kterékoli nižší vrstvy

- omezuje potřebu definovat obdobné operace v každé vrstvě, kód může být kompaktnější a efektivnější
- změna podsystému může ovlivnit kteroukoli vyšší vrstvu - obtížná údržba (grafické systémy - změnu pixelu lze vyvolat z kterékoli vrstvy)

56. Na základě čeho rozhodnete, jestli zrealizujete podsystém jako hardware nebo software?

Podsystém jako HW nebo SW

- HW můžeme považovat za pevně zadrátovanou optimalizovanou formu SW
- k implementaci v HW vedou dva hlavní důvody:
  - existuje HW, který poskytuje přesně požadovanou funkčnost
  - HW implementace poskytne vyšší výkonnost, než SW implementace na obecném CPU (např. na signálovém procesoru poběží algoritmus rychleji)
- nevýhoda - HW řešení není flexibilní

57. Srovnejte souborová a databázová datová úložiště, jakým způsobem byste se rozhodovali pro použití souboru, nebo databáze

soubory

- levné, jednoduché a permanentní

- nízká úroveň abstrakce - v aplikaci je nutný další kód pro práci s nimi
- vhodné pro data, která jsou objemná, ale zároveň obtížně strukturovatelná v terminologii DB systémů
- vhodné pro data, která mají malou informační hustotu, nebo data, která jsou uchovávána krátkou dobu

databáze

- společné rozhraní pro množinu aplikací pomocí standardního přístupového jazyka (SQL)
- výhodné pro data, ke kterým bude přistupovat více uživatelů, případně více programů, a pro data, která mohou být efektivně spravována příkazy DB jazyka
- poskytují další vlastnosti jako podporu transakcí, zotavení po havárii apod.
- *nevýhody:*
  - vyšší režie
  - nedostatečná podpora pro složitější datové struktury (relační databáze předpokládají velké množství dat s relativně jednoduchou strukturou)
  - často není možná čistá integrace s programovacím jazykem (SQL je neprocedurální, zatímco aplikaci vytváříme většinou v imperativním programovacím jazyce)

58. Vysvětlete pojmy systém řízený procedurálně, systém řízený událostmi, paralelní systém.

Systém řízený procedurálně
- běh je řízen programovým kódem
- procedura žádá o vstup např. z klávesnice a pozastaví se
- po příchodu běh pokračuje v proceduře, která o vstup žádala
- používají např. všechny znakově orientované aplikace v MS-DOS a v Linuxu
- výhoda - jednoduché implementace
- nevýhoda - obtížné zpracování asynchronních událostí (program musí požádat o vstup)

Systém řízený událostmi
- běh systému řídí dispečer poskytovaný podsystémem, programovaný jazykem nebo OS
- s jednotlivými událostmi jsou svázány procedury aplikace
- systém někdy poskytuje frontu událostí, nově příchozí události se řadí do fronty, ze které dispečer vybere následující událost a zavolá odpovídající proceduru (callback)
- procedura po skončení obsluhy událostí vrací řízení dispečeru
- řízení událostmi ve skutečnosti simuluje spolupracující vlákna uvnitř úlohy, ale na rozdíl od skutečného paralelismu zablokuje dlouho trvající procedura celou aplikaci
- téměř všechny GUI (Windows), simulace apod.
- jednoduchá obsluha nových typů událostí
- vede přirozeně k objektově orientovaným systémům - události se posílají jako zprávy jednotlivým objektům

Paralelní systémy
- řízení - několik nezávisle běžících objektů
- události přicházejí objektům jako zprávy
- objekt může čekat na vstup, zatímco ostatní pokračují v činnosti
(pozn. je dobré /někdy nutné/, aby pro každý projekt existovala osoba, která je zodpovědná za architekturu systému /šéf-architekt/)

59. Vysvětlete pojem architektonický styl, jaký architektonický styl jste použili ve vašem projektu?

Architektonický styl

- znovupoužitelná abstrakce systému
- vyskytuje se opakovaně v různých aplikacích (typické řešení)

- můžeme z něj vycházet při tvorbě vlastních aplikací
- vlastnosti jednotlivých stylů jsou známé již z existujících aplikací daného stylu (škálovatelnost, výkonnost, bezpečnost apod.)
- styl slouží jako komunikační nástroj a základ pro manažerské rozhodování (např. pro rozdělení do týmů, organizaci dokumentace projektu, apod.)
- je popsán:

```
1) množinou podsystémů a jejich typů (např. datové úložiště, proces, UI)
2) topologickým uspořádáním podsystémů
3) množinou sémantických omezení (např. datové úložiště nesmí měnit uložené hodnoty)
4) množinou interakčních mechanismů (volání podprogramu, událost, roura)
```

60. Popište architektonické styly dataflow, aktivní a pasivní datové úložiště.

- **Styl dataflow**
  - - používá se, pokud se můžeme na systém dívat jako na posloupnost transformací zpracovávajících vstup a produkujících výstup
  - - výhodou integrovatelnost - relativně jednoduché rozhraní mezi komponentami
  - - dva hlavní podstyly - "roury a filtry" a "dávkově sekvenční" architektura
  - - roury a filtry (pipe-and-filter architecture)
    - § • podsystémy nazýváme filtry, jsou propojené rourami, které přenášejí data
    - § • každý filtr pracuje nezávisle, pouze očekává data v určitém formátu a produkuje výstup v určeném formátu
  - příklad - překladače programovacích jazyků, např. překladač jazyka C: zdrojový text nejprve zpracuje preprocesor, poté se provede lexikální a syntaktická analýza, optimalizace, nakonec generování kódu

- **styl repositář (repository)(Pasivní datové úložiště}**
  - - centrem architektury je datové úložiště (databáze nebo soubor)
  - - ostatní podsystémy (klienti) k úložišti přistupují a čtou, přidávají, ruší nebo modifikují data
  - -- používá se, pokud je požadováno uchování, výběr a správa velkého množství dat a pokud jsou data vhodně strukturovaná
  - - hlavní cíle
    - § o integrovatelnost - klientské podsystémy pracují nezávisle
    - § o škálovatelnost - možnost snadno přidávat nové klienty a data
  - Příklad Origo, Rapidshare

- **Aktivní datové úložiště: styl tabule (blackboard)**
  - - množina agentů spolupracuje pomocí datového úložiště
    - § o úložiště posílá oznámení agentům o změně dat, která je zajímají
    - § o agent vyhodnotí obsah úložiště, případně vloží výsledek nebo částečné řešení
  - - například systémy pro umělou inteligenci - neznáme vhodné uzavřené algoritmické řešení problému, ale umíme řešit pomocí např. znalostních agentů, kteří k řešení přispívají

61. Popište a vysvětlete princip třívrstvé architektury.

- funkčnost se rozděluje do tří částí (každá vrstva svá specifika, vyžaduje jiný typ programování, jiné testování apod.)
- prezentační vrstva
  - klienti - obvykle jeden klient slouží jednomu uživateli
  - vytváří ji ve velké míře designéři
  - výhodné co největší oddělení od aplikační vrstvy, zvláště pokud aplikace poběží na víc zařízeních
- datová vrstva
  - "technický" kód, databázoví specialisté
  - načtení-uložení dat z/do databáze
  - zajišťuje správné fungování transakcí, odolnost proti chybám, rychlost při vysoké zátěži
  - databáze + uložené procedury + obslužný kód
- aplikační logika
  - logika aplikace, reprezentace tříd (např. účet), obchodní pravidla (např. jak se počítá úrok)
  - obchodní pravidla se můžou často měnit – modifikace komponent
- v současné době nejoblíbenější
- častá varianta: tenký klient - klientem je např. WWW prohlížeč

2 možné varianty:

- Tlustý klient - klient obsahuje jak aplikační tak prezentacni vrstvu, datova vrstva je zvlast.
- Tenký klient - klien obsahuje pouze prezentační vrstvu, data získává z externí aplikační a datové vrstvy.

62. Popište a vysvětlete princip architektury MVC.

- architektura používaná pro klasické i webové interaktivní aplikace
- blízká pojetí třívrstvé architektury
- odděluje funkční vrstvu aplikace ("Model") od dvou aspektů uživatelského rozhraní (nazývaných "View" a "Controller")
  - Model - objekty, které budeme v aplikaci prohlížet a měnit (např. obchodní data)
  - View (náhled)
    - zobrazení modelu (prezentace modelu na obrazovce)
    - při změně obsahu (stavu) modelu oznámí toto model všem závislým view, je vyvolána změna zobrazení
    - více možností prezentace modelu – více view
  - Controller - obsluhuje interakci uživatele s modelem
    - na základě změn modelu a vstupů uživatele (stisku kláves, výběru z menu) vybírá View, které se má zobrazit
    - reakci na uživatelský vstup je možné měnit, aniž by to ovlivnilo model nebo view (klávesová zkratka, výběr z menu)
    - typicky jeden Controller pro množinu příbuzných fcí

63. Vysvětlete pojmy spojitost, stavový prostor.

Stavový prostor a chování

= opět jednoduše, je to množina všech stavů, ve kterém se objekt může nacházet, to znamená varianty jeho stavových proměnných (atributů).

= vlastnosti třídy

- stavový prostor – celek všech povolených stavů libovolného objektu třídy T

- dimenze stavového prostoru přibližně odpovídá atributům definovaným v dané třídě (nelze počítat atributy, které lze odvodit z jiných)

Spojitost

= jednoduše je to to když jsou třeba 2 objekty provázány proměnnou a pokud změníme v jednom objektu typ proměnné tak musíme změnit v druhém objektu práci s proměnnou aby byly oba objekty správné a funkční.

- spojité sw prvky jsou zrozeny ze související potřeby (během analýzy, návrhu, programování)
- definice spojitosti pro sw: Spojitost mezi dvěma sw elementy A a B znamená:
  - lze postulovat nějakou změnu A, která bude vyžadovat i změnu B (nebo alespoň důkladné prověření), aby byla zachována celková správnost
  - lze postulovat změnu, která bude vyžadovat společnou změnu A i B, aby byla zachována celková správnost

64. Vysvětlete pojmy neměnná třídy, předběžné a následné podmínky.

Neměnná třídy – omezení stavového prostoru

- neměnná třídy je podmínka, kterou musí v každém okamžiku naplnit každý objekt dané třídy (když je objekt v rovnováze)

= lidsky = pochopil jsem to tak, že pokud máme třídu a její instance (objekty), pak třeba podmínka že atribut váha musí být větší než nula je omezení celého stavového prostoru objektů (nemůžeme se pohybovat v záporných hodnotách).

Předběžné a následné podmínky

= lidsky = máme metodu, metoda vyžaduje jako parametr jméno (= předběžná podmínky), pokud je jméno prázdné vyhodíme třeba NullPointerException. Pak máme podmínku na konci, například vypočítávám v metodě váhu, a váha mi vyjde záporná (já požaduji kladnou váhu = následná podmínka).. nemohu ji přeci vrátit to by byla chyba.. tak vyvolám další vyjímku.

- vztahují se k operacím
  - předběžná podmínka musí být pravdivá na začátku operace, pokud není, může operace odmítnout vykonání a vyvolat stav výjimky
  - následná podmínka musí být pravdivá na konci vykonávání operace, v opačném případě je implementace operace nesprávná je třeba ji opravit
  - podmínky společně formují kontrakt s klientem (viz scénáře užití)

65. Vysvětlete princip kovariance a kontravariance.

1. Neměnná podřízené třídy je přinejmenším tak silná jako neměnná nadřízené třídy
2. Každá operace nadřízené třídy má odpovídající operaci v podřízené třídě se stejným názvem a podpisem
3. Předběžná podmínka každé operace není silnější než odpovídající předběžná podmínka operace v nadřízené třídě – princip kontravariance
4. Následná podmínka každé operace je přinejmenším tak silná jako odpovídající podmínka operace v nadřízené třídě – princip kovariance

= toto celé je jednoduše princip dědění tříd.

1. následník třídy nebude měnit již omezené hodnoty stavového prostoru.
2. následník obsahuje metody od rodiče.
3. následník nebude v metodách spřísňovat ve vstupních podmínkách.
4. následník nebude v metodách povolovat ve výstupních podmínkách

66. Vysvětlete pojmy návrhový vzor, katalog návrhových vzorů.

při řešení problémů se často dá vycházet z ověřeného způsobu řešení
Návrhový vzor:

- Popisuje zobecnění konkrétních situací (přirovnání k matematickému vzorci)

Katalog vzorů

- Vyhledávání podle smyslu vzoru
- Vyhledávání podle kritérií – klasifikace vzoru:
  - Podle účelu (purpose)
  - Rozsahu platnosti (scope)
  - Důležitější rozdělení dle účelu

67. Popište základní principy, na kterých jsou založeny návrhové vzory.

- Dva pohledy: klient-rozhraní a rozhraní-implementace, pro návrhový vzor důležitý první pohled -> změnu implementace klient nepozná -> nebudeme se zaměřovat na implementaci
- Polymorfismus
- Posílání zpráv a vyvolávání metod
- Abstraktní třída
- Abstraktní operace
- Čistá abstraktní třída
- Rozhraní

68. Popište základní princip návrhového vzoru Iterátor.

- smysl: jednoduchý a sekvenční přístup ke složité struktuře objektů

- motiv: agregovaný objekt (jako je např. seznam) by měl umožňovat procházení, aniž by k tomu uživatel musel znát vnitřní strukturu agregátu (která se může změnit) – uživatel = zjednodušený pohled na složitou strukturu a její snadné ovládání

- iterátor udržuje informaci o aktuálním objektu v agregátu, umí se posunout na následující prvek agregátu a prvek poskytnout uživateli (anebo také vrátit první prvek seznamu, poslední prvek seznamu apod.)

69. Popište základní princip návrhového vzoru Abstraktní továrna.

- klasifikace: Tvořivý, Objekt
- smysl: Zavedení rozhraní pro tvorbu rodin (řad) příbuzných nebo závislých objektů, aniž by se musely specifikovat konkrétní třídy, klient je izolován od volání těchto tříd
- motiv:
  - dva (více) standardů vzhledu grafického uživatelského rozhraní, označme St1 a St2,... lze zavést buď St1, nebo St2, nebo...
  - jiný vzhled definuje odlišná zobrazení a chování prvků GUI - oken, posuvníků, atd.
  - aplikace může přepínat mezi standardy
  - definována abstraktní třída FactorySt – rozhraní pro tvorbu prvků GUI, konkrétní třídy FactorySt1 a FactorySt2
  - definování abstraktní třídy pro prvky GUI - pro okna, posuvníky,...tj. např. pro okna WindowSt1 a WindowSt2 abstraktní třída WindowSt, pro posuvníky ScrollBarSt1, ScrollBarSt2, abstraktní třída ScrollBarSt
  - FactorySt – abstraktní operace pro vytvoření prvku GUI např. CreateWindow() a CreateScrollBar()

- FactorySt1 a FactorySt2 - konkrétní implementace rozhraní FactorySt, vracejí odpovídající typy prvku GUI, tj. např. ScrollbarSt1 nebo ScrollBarSt2
- Volba prvků GUI je převedena na volbu objektu továrny FactorySt1, nebo FactorySt2,...
- klient vytváří prvky GUI pomocí rozhraní FactorySt, zavazuje se pouze k rozhraní definovanému abstraktní třídou, nevolá konstruktory objektů, pouze metody rozhraní
- objekty FactorySt, tj. např. objekt FactorySt1 a objekt FactorySt2 umístíme do kolekce, nazveme např. ManagerFactory, tyto objekty se přidávají spolu s klíčem

70. Popište základní princip návrhového vzoru Tovární metoda.

- klasifikace: Tvořivý, Třída
- smysl: Definuje metodu rozhraní pro zrod objektu na úrovni předka, ale ponechává potomkovi k rozhodnutí, jakého konkrétního typu tento objekt bude
- alias: Virtuální konstruktor
- motiv: použití ve vzoru Abstraktní továrna, rozhraní obsahuje metody CreateProduct(), návratové typy metod jsou vrcholy stromu daných produktů, každá konkrétní třída abstraktní továrny dosazuje do návratové hodnoty konkrétní typ dědice produktu
- jiný motiv:
- vývoj knihovny pro tvorbu editorů, několik možných editorů podle typů dokumentů
- daný typ aplikace pracuje s daným typem dokumentu, abstraktní třídy Application a Document, podtřída třídy Document je specifická dané oblasti
- třída Application nemůže předvídat konkrétní třídu dokumentu, pouze ví, kdy by měl být dokument vytvořen, ale neví, jaký druh dokumentu vytvořit
- řešení: abstraktní metoda CreateDocument() předefinovaná konkrétními aplikačními podtřídami, které vytvářejí konkrétní dokumenty

příklad

- metoda openWindow() ve třídách Calculator a PhoneBook má téměř stejnou podobu:

```
x = new nějakéOkno(); // typ okna závisí na třídě
x.openWidget();
abstract class DesktopObject {
    abstract Window getWindowInstance();
    void openWindow() {
        Window w = this.getWindowInstance();
        w.openWidget();
    }
}
class Calculator extends DesktopObject {
    Window getWindowInstance() {
        .... // vrací instanci správné třídy
    }
}
```

- getWindowInstance() je tovární metoda, protože (většinou) vyrábí instanci související
- = v aplikaci se zavolá x.openWidget, x je libovolne cokoliv co se kdekoliv vzalo a je to oddedeno od DesktopObject takze ma implmentovanu metodu getWindowInstance, na základě konkrétní implementace se zavolá konkrétní metoda pro vytvoření instance okna.

71. Popište základní princip návrhového vzoru Singleton.

- klasifikace: tvořivý, objekt
- smysl: třída má jen jednu jedinou instanci globálně viditelnou
- motiv:
  - často chceme, aby nějaký objekt byl globálně viditelný a jedinečný (objekt faktory u vzoru Abstract factory)
  - instance se umístí jako privátní statický člen, zpřístupňuje se přes statickou veřejnou metodu

= máme jednu jedinečnou statickou instanci uloženou ve statické třídě a pokaždé při volání tovární metody jí dostaneme = všechny instance jsou stejné.

72. Popište základní princip návrhového vzoru Skladba (Composite)

- klasifikace: strukturální, objekt
- smysl: skládá prvky do stromové struktury, k prvkům stromu přistupuje jednotným způsobem
- motiv:
  - obrazce se společnou abstraktní třídou Obrazec (metoda spoctiObsah()) viz PT
  - složený obrazec začleněn do stromu (člen rodiny obrazců, implementuje rozhraní Obrazec, rekurzivní volání) – prvky strom nejsou unifikované (listy a vnitřní uzly)
  - transparentní řešení – všechny možné operace do abstraktní třídy CObrazec (i operace pro správu dětí – pridej(), odeber(), zjistiDeti()) – unifikace prvků

73. Popište základní princip jednoho z následujících návrhových vzorů: Dekorátor (Decorator), Most (Bridge), Muší váha (Flyweight), Stavitel (Builder), Stategie (Strategy), Stav (State).

Dekorátor:

- smysl: Dynamické přidání další funkcionality k objektu, zavedení flexibilní alternativy k dědění
- motiv:
  - chceme přidat funkcionalitu k nějaké třídě – možné řešení je dědičnost – nevýhodou statická vazba a omezená možnost kombinací
  - jiné řešení – uzavření komponenty do jiného objektu, tzv. dekorátoru, využití skladby, výsledná struktura připomíná zřetězený seznam, nutné stejné rozhraní pro propojení mezi prvky reprezentované jedinou metodou operation()

Strategie (Stratgy)

- Klasifikace: Chování, objekt
- Smysl: Definuje množinu algoritmů, které lze za běhu vyměnit
- Motiv:
  - máme k dispozici více algoritmů vyhledávání dat, požadavek na použitý algoritmus dán obsluhou za běhu programu, neflexibilní řešení: přepínač
  - jiné řešení: rozhraní Strategy s operací např. SearchAlgorithm(), dědicové ji přepisují, výběr algoritmu na základě výběru instance třídy Strategy

74. Popište základní princip jednoho z následujících návrhových vzorů: Adaptér, Proxy, Fasáda, Mediátor, Observer.

Fasáda

- Smysl: zjednodušuje přístup klienta k subsystému přes jeden nebo více rozhraní
- Motiv:
  - Užívaný zejména u komponentové technologie
  - Je lepší nabídnout klientovi jedno (popřípadě více) rozhraní místo celou množinu tříd – je to pro něj jednodušší
  - Hlavně pro vnitřně složité subsystémy – rozhraní = pohled na subsystém z pohledu případu užití (ostatní funkcionality se odstíní)

Observer

- smysl: možnost sledování změny objektu; když objekt změní stav, ostatní objekty na to zareagují, avšak není přímá vazba od sledovaného objektu k těmto objektům
- Motiv
  - obdoba událostního systému (Java - listener)

- první krok: objekt se spolu se svojí operací zaregistruje ke službě události jiného objektu, druhý krok: dojde k události, všechny registrované operace se zavolají
- nutnost kompatibility volání zaregistrovaných operací -> vzor Observer

75. Popište základní princip jednoho z následujících návrhových vzorů: Transfer Object, Data Access Object, Front Controller.

Transfer Object (J2EE pattern)

- kontext: klient si potřebuje vyměňovat data s komponentou
- motiv:
  - komponenty na straně serveru poskytují data, tj. dávají k dispozici přístupové metody, každé volání metody klientem je potenciálně vzdálené, pokud je volání mnoho a klienti žádají jednoduché atributy, klesá výkonnost celé aplikace
  - řešení: aplikační data jsou zapouzdřena do Transfer objectu, manipulace s tímto objektem pak probíhá přes jedinou metodu, klient žádající o data obdrží celý Transfer object poté, co je tento objekt zkonstruován
  - využití Transfer objectu je spojeno s mnoha strategiemi

Front Controller (J2EE pattern)

- kontext – mechanismus řízení požadavků na úrovni prezentační vrstvy, řízení a koordinace zpracování uživatelských požadavků, tento mechanismus může být centralizovaný/decentralizovaný
- motiv
  - pokud uživatel může přistupovat k prezentační vrstvě bez „průchodu“ centralizovaný mechanismem řízení požadavků, je často nutné duplikovat kód jednotlivých view, navíc dochází ke smísení obsahu a řízení daného view
- řešení:
  - použití controleru jako prvotního bodu kontaktu s požadavkem, controller zpravuje další zpracování požadavku (např. přihlášení, výběr konkrétního požadavku apod.)
  - pro imlementaci Front Controlleru existuje několik strategií

Data Acces Object

- kontext – přístup k úložišti dat závisí na typu daného úložiště (relační databáze, soubor,.....) a jeho implementaci
- motiv:
  - potřebujeme získávat/ukládat data z/do persistentních datových úložišť, tyto se mohou měnit, mají různá API, klient chce však používat jednotné rozhraní
  - řešení: Data Access object zapouzdřující přístup k datovému zdroji, DAO zařizuje veškerou komunikaci s datovým zdrojem ohledně ukládání/získávání dat (je to takový Adaptér)

76. Vysvětlete pojmy motivace a stimulace.

motivace

- činnost, kterou vyžadujeme, dáváme do souvislosti s existujícími vnitřními potřebami člověka
- působí i bez našeho vlivu tak dlouho, dokud je v souladu s aktuálními motivy člověka
- vyžaduje, abychom uměli odhadnout aktuální motivy lidí
- podstatně složitější nástroj než stimulace

(motivem chování je naplňování potřeb (nejzákladnější potřeba = příjemný pocit) viz Maslowova pyramida potřeb)

stimulace

- účinná tak dlouho, dokud působí jako podnět
- jakmile přestaneme investovat prostředky (čas, úsilí, prostředky), žádoucí lidská činnost se zastaví

77. Interpretujte zlaté pravidlo motivace.

Nepřitesávejte lidi k obrazu jejich úkolů, ale snažte se spíše přizpůsobit úkoly lidem a jejich aktuálním motivům.

78. Vysvětlete pojmy motivační založení, motivační poloha, motivační naladění

motivační založení

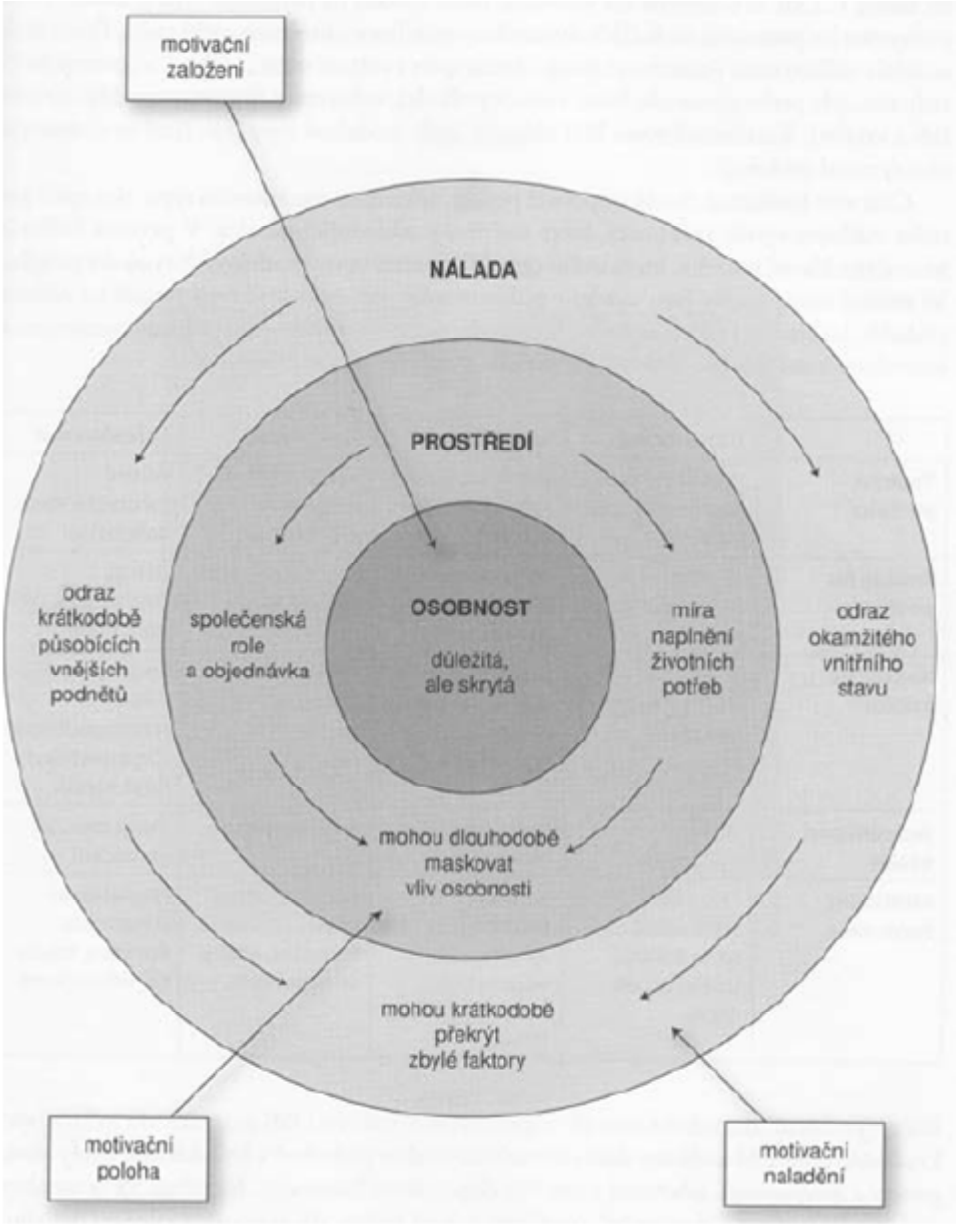
- prakticky neměnná charakteristika lidské osobnosti (během života se mění pomalu a neochotně)
- můžeme poznat, pochopit, použít
- může být maskováno a překryto vnějšími podmínkami, „společenskou objednávkou“, rolemi v životě (spolupracovník, rodinný příslušník, soused)
- pokud je v rozporu se „společenskou objednávkou“, projevuje se zejména v krizových situacích nebo při spontánních reakcích

motivační poloha

- souvisí s životními a pracovními podmínkami; prostřednictvím podmínek ji můžeme ovlivňovat
- za určitých okolností může dominovat a potlačit vliv podnětů souvisejících s motivačním založením
- dominuje, když okolí naléhavě vyžaduje po člověku specifické chování (dané např. sociální rolí)

motivační naladění

- okamžitý stav vnitřních pohnutek
- rychle přichází a odeznívá, nemá smysl jej významněji řešit



79. Vysvětlete pojem MBTI, k čemu je dobré?

MBTI (Myers-Briggs Type Indicator)

- osobnostní test pro zjištění způsobu zpracování informace

- zjištění nijak nehodnotí, neexistují dobré a špatné výsledky
- nástroj pro zjišťování předpokladů, neříká, že to tak určitě bude
- nehodnotí schopnosti a dovednosti, ale preference a typy
- žádná z preferencí není nadřazena jiné
- není černobílým pohledem
- čtyři dvojice parametrů - poměr
  - odkud čerpáme podněty (I - introverze vs. E -extroverze)
  - jak podněty přijímáme (S – smysly vs. N – Intuice)
  - jak podněty zpracováváme (F- citem vs. T- rozumem)
  - jak se rozhodujeme (P-vnímající vs. J- usuzující)

80. Jak byste se starali o rozvoj týmu, který je vám svěřen?

- Péče o diverzitu (v lidech a rolích) – zvládnutí rozdílů mezi členy týmu, efektivní využívání těchto rozdílů, učení se z vlastních výsledků, vzájemná pomoc – její forma není rozhodující, smyslem není řešit problém, ale poskytnout jiné názory, podněty
- Péče o sdílení (cílů a cest) – analogie k vedení z řízení firmy, sdílení vize, vlastní cíle a cesty, funkční zpětné vazby… nejlepším prostředkem pro hledání jsou společné diskuse (neformální diskuse - zkušenosti, formální diskuse o konkrétních problémech – postupný posun: schopnost prezentovat, schopnost naslouchat a chápat, schopnost pomáhat – nejlepší nástroje sebeřízení týmu)
- Proces přeměny skupiny v tým

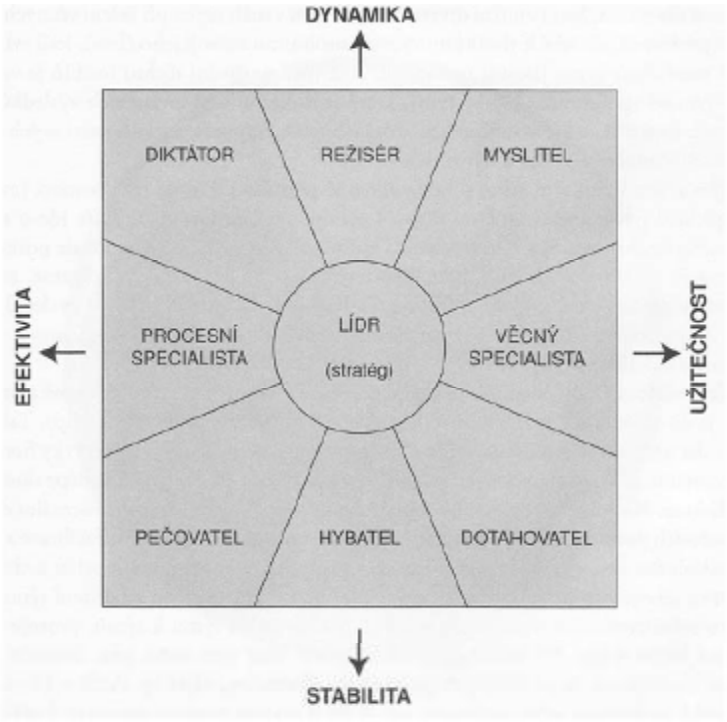
81. Vysvětlete pojem týmová role, „strach z prázdna“, popište význam rozdělení rolí mezi členy týmu.

Rozdělení rolí v týmu

- Lidé si v týmu přirozeně hledají svoje místa
- Nejde o formální role (manažer, sekretářka), ale o spontánní vykonávaní potřebných funkcí týmu (např. rozdělení úkolů, dotahování věcí do konce)
  - Lídr (stratég) – komplexní myšlení, dokáže nadchnout ostatní, univerzální, všestranný
  - Myslitel – racionální inteligence, kreativní, hravý, nezávislý
  - Režisér – uvedení myšlenek, strategií
  - a další jako Diktátor, Procesní specialista, Pečovatel, Hybatel, Dotahovatel, Věcný specialista
- Strach z prázdna – u skutečného týmu se vždy najde člověk, který se spontánně přesune do neobsazeného prostoru a vezme na sebe funkce, které s tímto prostorem souvisejí

Význam rozdělení rolí

- specifické vlastnosti každého člena skupiny plně využity v kontextu specifických vlastností ostatních členů skupiny



Obrázek 118 Týmové role

82. Jaký je smysl testování? Vysvětlete následující tvrzení: testování snižuje náklady.

Smysl: ten je snad jasný, při testování se odlaďují chyby a testuje se celá stabilita aplikace.

Proč testování snižuje náklady?Protože po vydaní softwaru z chybami má zadavatel/klient/zákazník právo na náhradu škody, která mu vznikla vadným softwarem a navíc může požadovat opravení chyb což má opět za následek další výdaje.

- Testování je poslední obranou linií před špatnou kvalitou produktu.
- Každý software obsahuje alespoň jednu chybu.
- Bezpečnost

83. Z jakých dokumentů vychází testování?

Základní dokumenty:

1. Dokumentace specifikace
2. Vize

1. Specifikace požadavků(vytvářím správný produkt?).

2. Objektový návrh/Konceptualní model(vytvářím produkt správně?).

84. Vysvětlete pojmy black box a white-box testování

Dva základní přístupy k testování.

Black-Box

- tester se dívá na program jako na černou skříňku, to co je uvnitř ho nezajímá
- hledají se případy, kdy se program nechová podle specifikace
- pro nalezení všech defektů se program testuje se všemi možnými vstupy - platnými i neplatnými (v některých případech je to nereálné)

White-Box

- testovací data se odvíjejí od vnitřní logiky
- pro úplné otestování programu bychom potřebovali pomocí testovacích případů otestovat všechny možné logické cesty v programu (analogie otestování programu se všemi možnými vstupy, viz výše)

85. Vysvětlete v rámci testování princip rozdělení vstupů do ekvivalentních tříd.

Rozdělení vstupů do ekvivalentních tříd

- dobrý testovací případ bude mít dvě vlastnosti:
  - bude vyvolávat co nejvíc vstupních podmínek, tím omezí celkový počet potřebných testovacích případů
  - bude pokrývat určitou množinu vstupních hodnot

Množinu vstupů pak rozdělíme do tříd ekvivalence tak, abychom mohli rozumně předpokládat, že test nějaké reprezentativní hodnoty v dané třídě je ekvivalentní testu kterékoli další hodnoty - z těchto úvah je odvozena metodologie pro black-box testování známá jako equivalence partitioning = rozdělení do tříd ekvivalence - nejprve identifikujeme třídy ekvivalence a pak definujeme testovací případy

= Dokonalý test by měl být takový, že co nejmenším počtem testů pokryje nejvíce vstupních hodnot a vnitřních podmínek a větvení.

86. Jak byste otestovali cyklus s podmínkou na začátku?

Musíme vytvořit tolik testovacích případů, aby se v každém příkazu "if" vykonala alespoň jednou větev při podmínce "false" a alespoň jednou větev při podmínce "true" atd.

Jinak řečeno rozumným kritériem je pokrytí všech kombinací podmínek v rozhodovacím příkazu (multiple condition coverage.

87. Jak byste otestovali validitu e-mailové adresy?

Regulárním výrazem.

88. Vysvětlete pojmy testování jednotek, JUnit.

Způsob testování tříd (třída = jednotka). Pro každou testovanou třídu se vytvoří nová třída, která se bude jmenovat stejně,ale bude ke jménu třídy přidáno "Test". Testují se např. proměnné, zda jsou inicializované, na základě vstupu se testuje očekávaný výstup apod. Testují se i metody.

JUnit je nástroj pro testovaní v JAVE.

89. Vysvětlete pojmy integrační testování, validační testování.

Integrační testování

- testování samotných komponent poté celého systému
- nejprve se sestaví samotné jádro a otestuje, poté se postupně přidávají jednotlivé komponenty a po každém přidání se znovu testuje
- nevhodné pro velké projekty, komponenty se můžou navzájem ovlivňovat a hledání defektu ve velkém systému je složité

Validační testování

- začíná tam, kde končí integrační testování
- testujeme, zda SW splňuje požadavky zadavatele
- testuje se pouze externí chování systému, vnitřní strukturu ignoruje
- obsah testu by měl specifikovat zadavatel
- snaha o zautomatizování, abychom mohli spouštět po změnách aplikace

90. Vysvětlete pojmy alfa a beta testování, zátěžové testování.

Alfa a beta testování

- pro generické produkty není většinou možné vykonat přejímací testování u každého zákazníka, proto alfa a beta testování
- alfa testy: na pracovišti, kde se SW vyvíjí (známé prostředí) - testuje uživatel, vývojoví pracovníci ho sledují a zaznamenávají problémy
- beta testy: testují vybraní uživatelé ve svém prostředí (vývojářům neznámém) - defekty ohlášené uživateli jsou opraveny => finální produkt

Zátěžové testování

- obvykle se používají testy, kde se zátěž postupně zvyšuje, dokud není výkonnost systému neakceptovatelná nebo dokud systém nehavaruje
- zátěž = množství dat, frekvence požadavků, data, která jsou extrémně náročná na zpracování
- je vhodné určit části kódu, které mohou být problematické při velké zátěži, zátěžové testy navrhnout tak, aby pokrývaly především tyto části kódu
- ověření, zda havárie systému nepoškodí data apod.

- může odhalit některé defekty, které se normálně neprojeví
- důležité zejména u internetových aplikací, v distribuovaných systémech apod., kde se vysoce zatížené systémy mohou zahltit, protože si vyměňují mnoho koordinačních dat, čímž se opět zvyšuje zátěž systému atd.

91. Co jsou to autotesty a jaký je jejich význam?

- autotesty jsou automatické systémy pro testování softwaru podle stanovených kritérií (konfigurace)
- mnohem efektivnější práce testerů.. umožňuje spouštět rutinní testy (například přes noc) a ráno testeři pouze testy vyhodnotí.

- odhalí se tím zásadní chyby, které by nikdo nečekal - autotest se neunaví.. testeři potom mohou testovat pouze nerutinní záležitosti, které nejdou otestovat autotestem.

- výhody:
  - rychlost
  - efektivity
  - redukuje čas potřebný pro testování
  - pracuje přesně a nedělá chyby

92. Jak v závislosti na stavu výsledků testů poznáte, že můžete vydat produkt?

V případě dobrých výsledků testů se může produkt vydat. Výsledky testů nemusí být 100%. Záleží na druhu produktu a cílové skupině, pro kterou je produkt určen.

Test by měl být 100% v námi známé a otestované oblasti.. neměli bychom vydat produkt, kde známe chybu, ale neopravili jsme ji. Samozřejmě žádný produkt nemůže být 100% funkční, ale vše známé by mělo být odstraněné.

93. Proč podnik nakupuje informační systém?

- ICT v podniku jsou v každém případě investicí.
- Management očekává, že se něco zlepší!

- Aktivní a pasivní motivy
- Příklad aktivních motivů:
  - Vyšší komfort obsluhy zákazníků:
  - rychlejší obsluha
  - více zákazníků
  - více obslužných kanálů (např. internet), atd.
  - Získání náskoku před konkurencí
  - Získání image
- Příklad pasivních motivů:
  - Legislativní požadavek (zákon)
  - Opatření regulátora (např. ČTÚ)
  - Oborová norma (Basel II, apod.)
  - Příkaz (zahraničního) vlastníka

94. Vysvětlete pojmy aktivní a pasivní motivy pro nákup ICT, uveďte příklady

Příklad aktivních motivů:

- Vyšší komfort obsluhy zákazníků:
- rychlejší obsluha
- více zákazníků
- více obslužných kanálů (např. internet), atd.
- Získání náskoku před konkurencí
- Získání image

Příklad pasivních motivů:

- Legislativní požadavek (zákon)
- Opatření regulátora (např. ČTÚ)
- Oborová norma (Basel II, apod.)
- Příkaz (zahraničního) vlastníka

95. Jakým způsobem byste řešili platební kalendář během vývoje IS, jakým způsobem se vypořádáte se změnami?

- **Externí**
  - Jaký rozpočet má klient na provedení investice?
  - Jaké jsou možnosti úprav rozpočtu v průběhu dodávky?
  - Co nabídla konkurence a za jaké ceny pracuje?
  - Použít model Fixed Price nebo Time & Material?
  - Vyvážený platební kalendář
- **Interní**
  - Náklady na zkontrahování OP („pre-sale“)
  - Realizační rozpočet (vč. rizik)
  - Marže
- **OP musí být přínosný!**
  - Přínosy nemusí být vždy finanční.

Rozpočet fixovat pouze na fáze, které již nevyžadují další analýzu. Jinde použít kvalifikovaný odhad.

Důležité je skloubit Externí a interní stránky OP. Řádně rozvrhonout platební kalendář vybrat si mezi paušální platbou a nebo Člověkohodinovou. Případný průsery (obvzlášť u paušálních plateb) zahrnout jako dodatek ke smlouvě...

96. sVyberte tři z dobrých praktik při prodeji IS a vysvětlete je.

1. Oddělovat část projekce IS, implementace IS a servisu IS (jak v nabídce, tak smluvně)
2. Rozpočet fixovat pouze na fáze, které nevyžadují další analýzu (jinde použít kvalifikovaný odhad)
3. Harmonogram uvádět relativní (T+n...)
4. Zodpovědně a co nejpřesněji specifikovat součinnost klienta
5. Předkládat anonymizovaná CV
6. Nabídku zcela nahradit Smlouvou o dílo
7. Vytvořit a prezentovat prototyp

8. Zůstávat v maximálním možném kontaktu s klientem

97. Vyberte tři z dobrých praktik při nákupu IS a vysvětlete je.

1. V případě požadavku na fixaci ceny předsadit projektu Úvodní studii či Studii proveditelnosti, jejímž jedním cílem bude fixace rozpočtu na konkrétní požadované úrovni.
2. Ptát se, jak dodavatel minimalizuje rizika
    1. Iterativní přístup
    2. Průběžné prezentace mezivýstupů
    3. Znalost prostředí klienta
3. V případě IS na zakázku:
    1. Požadovat zdrojové kódy a kompletní dokumentaci
    2. Minimalizovat licenční omezení a zajistit exkluzivitu
4. „Rozumné“ smluvní sankce a záruky
5. Dostatečný čas na zpracování nabídek

98. Jaké právní předpisy platí na Internetu?

- Platí obecný právní řád (obecně závazné i speciální právní předpisy - ObčZ, ObchZ, TrZ, PřestZ apod.
- Problémem je obtížná uchopitelnost Internetu z hlediska standardního práva a to mj. vzhledem k dobrovolnosti. ... typickým příkladem domény

- Internet je jen fikce - Internet jako takový právně neexistuje, neboť nemůže nabývat práv ani se zavazovat (viz ust. § 18, odst. 1 ObčZ)
- Absence majitele = absence odpovědnosti
- Absence zvláštní právní úpravy Internetu... podléhá obecně platným právním normám

99. Vysvětlete pojem autorské dílo, uveďte příklady autorských děl, co není autorským dílem?

Autorské právo (anglicky označováno jako copyright) je odvětví práva, které popisuje nároky tvůrců tzv. „autorských děl“, tzn. spisovatele, hudebníky, filmaře, programátory apod. na ochranu před nespravedlivým využíváním jejich tvorby. Prostřednictvím autorského práva poskytuje stát po jistou omezenou dobu autorům výlučnou možnost rozhodnout o některých aspektech využívání jejich děl. Autorské právo je součástí tzv. duševního vlastnictví.
Autorské právo nechrání samotné *myšlenky* či *ideje*, pouze konkrétní díla, konkrétní *vyjádření* takových myšlenek, dílo v objektivně vnímatelné podobě. Autorským dílem je pouze jedinečný výsledek tvůrčí činnosti autora, dílem není námět, zpráva, informace, metoda, teorie, vzorec, graf, tabulka fyzikálních konstant, výstup počítačového programu apod. samy o sobě.

Předmětem práva autorského je dílo literární a jiné dílo umělecké a dílo vědecké, které je jedinečným výsledkem tvůrčí činnosti autora a je vyjádřeno v jakékoli objektivně vnímatelné podobě včetně podoby elektronické, trvale nebo dočasně, bez ohledu na jeho rozsah, účel nebo význam (dále jen "dílo").

1. Dílem je zejména dílo slovesné vyjádřené řečí nebo písmem, dílo hudební, dílo dramatické a dílo hudebně dramatické, dílo choreografické a dílo pantomimické, dílo fotografické a dílo vyjádřené postupem podobným fotografii, dílo audiovizuální, jako je dílo kinematografické, dílo výtvarné, jako je dílo malířské, grafické a sochařské, dílo architektonické včetně díla urbanistického, dílo užitého umění a dílo kartografické.
2. Za dílo se považuje též počítačový program, je-li původní v tom smyslu, že je autorovým vlastním duševním výtvorem.

3. Za dílo souborné se považuje databáze, která je způsobem výběru nebo uspořádáním obsahu autorovým vlastním duševním výtvorem.
4. Fotografie, která je původní ve smyslu věty prvé, je chráněna jako dílo fotografické.
5. Právo autorské se vztahuje na dílo dokončené, jeho jednotlivé vývojové fáze a části, včetně názvu a jmen postav.
6. Sborník, jako je časopis, encyklopedie, antologie, pásmo, výstava nebo jiná databáze (§ 88), je-li souborem nezávislých děl nebo jiných prvků, který je způsobem výběru nebo uspořádáním obsahu jedinečným výsledkem tvůrčí činnosti autora, je dílem souborným.

- Dílem podle tohoto zákona není zejména námět díla sám o sobě, denní zpráva nebo jiný údaj sám o sobě, myšlenka, postup, princip, metoda, objev, vědecká teorie, matematický a obdobný vzorec, statistický graf a podobný předmět sám o sobě.

100. Popište princip dualismu autorského práva – výlučné právo majetkové a výlučné právo osobnostní

Dualismus práv – moderní kompromisní řešení, které odděluje práva osobnostní a majetková, lišící se i režimem ochrany.

Právo autorské zahrnuje výlučná práva osobnostní (§ 11) a výlučná práva majetková (§ 12 a násl.).

… dualita práv, novinka.

Jednoduše řečeno: osobnostních práv vás nikdo nikdy nezbaví, jste autorem na věky věků. Ale právo prodávat, šířit atd. nějaké dílo můžete prodat/ může přejít na někoho jiného. Po 70 letech od smrti autora majetková práva zanikají a dílo se stane public domain, tedy veřejným