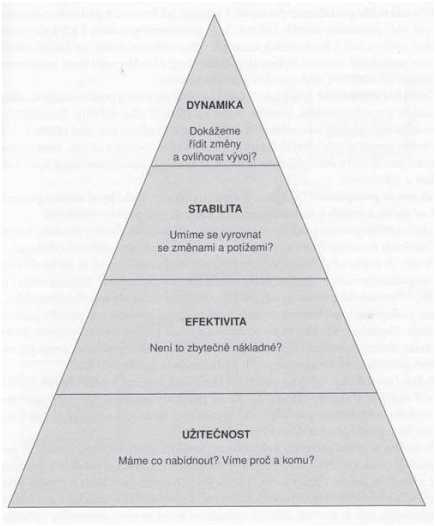
1. Firma a její fungování, teorie vitality

- následující texty a doprovodné obrázky jsou převážně převzaty z publikace Jiřího Plamínka: Vedení lidí, týmů a firem

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

Budování firmy - pyramida vitality



Obrázek 1 Pyramida vitality

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 -f*irma 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

Zajišťování užitečnosti

- probíhá dle schématu *Subjekty -> Potřeby -> Produkty*

1. *Subjekty:* Komu budeme užiteční (chceme či musíme)
   - majitelé
   - zákazníci
   - zaměstnanci
   - dodavatelé
   - stát

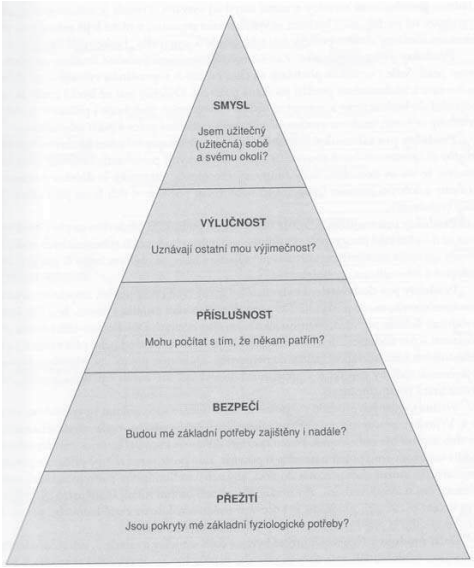
   - je důležité na klíčové subjekty nezapomenout (stává se např. u vznikajících firem s jasným a specifickým produktem – typicky IT) – pro konkrétní situace: nástroj vztahová analýza

2. *Potřeby*: Co od nás bude chtít?
- většina potřeb je univerzální a má hierarchickou strukturu – Abraham Maslow – hierarchie potřeb (kniha: Motivation and Personality)
   - *Přežití* – pokrytí základních biologických potřeb – metabolismus a reprodukce
   - *Bezpečnost* – zajištění přežití i v budoucnosti
   - *Příslušnost* – někam patřit – rodina, přátelé, škola, fotbalový tým,…
   - *Výlučnost* – nejen někam patřit, ale v daném prostředí i vynikat, získat uznání
   - *Smysl* – seberealizace, osobní rozvoj, užitečnost pro jiné

Význam Maslowovy pyramidy
   – univerzální lidské potřeby u všech skupin subjektů, lidé jim přisuzují různou váhu (podle toho, jak daleko se na pyramidě dostali)
   - pohyb po pyramidě směrem vzhůru i dolů
   - poskytuje vodítko pro definici potřeb a produktů

Vnímání potřeb
   – důležité je subjektivní vnímání vlastních potřeb (nikoli objektivní pohled) – vytváření umělých potřeb a problémů

3. *Produkty:*odvozeny od potřeb, je dobré se k jejich definici pravidelně vracet
   – Pro majitele – ekonomické výsledky, prestiž, image, seberealizace
   – Pro zákazníky – specifické potřeby (výrobky, služby), záruka, servis, image, kvalita jednání
   – Pro zaměstnance – mzdy, benefity, kariéra, seberealizace
   – Pro dodavatele – platby za dodávky, smlouvy, výhodná spolupráce, důvěra a dobré vztahy, prestiž, image
   – Pro stát – daně, pracovní místa, respektování zákonů, ekologická šetrnost



Obrázek 2 Pyramida potřeb

Zajišťování efektivity

- 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

Mít popsaný proces má řadu výhod
- posouzení efektivity – je-li proces dobře popsán, můžeme posoudit, je-li to optimální transformace, neopakují-li se některé aktivity zbytečně, nejsou úplně zbytečné (Pozn. úkolem v IT při výrobě zakázkového softwaru je pochopit skutečné transformace ve firmě nebo je pomoci definovat – tj. vyřešit problém efektivity, vzniklý software podporuje a definuje optimální transformace), srovnejte s efektivitou algoritmu na nižší úrovně abstrakce
- měření procesů – přechod mezi aktivitami – místo vhodné ke zjišťování parametrů důležitých z hlediska efektivity (čas, náklady, kvalita)
- definice zdrojů – stanovíme množství a povahu zdrojů pro čerpání vstupů
- rozdělení zodpovědnosti – za jednotlivé aktivity i celý proces, na základě sledování parametrů na rozhraní aktivit můžeme žádat případnou nápravu
- podpora procesů – lze připojit paralelní procesy a zdroje, které jej podporují (typicky informační systém), sledují (monitoring) a řídí
- řízení procesů – popsaný proces lze měnit v souvislosti se změnami prostředí, firemních cílů, ad., díky monitoringu a zavedenému pořádku lze předvídat a ověřovat důsledky změn

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

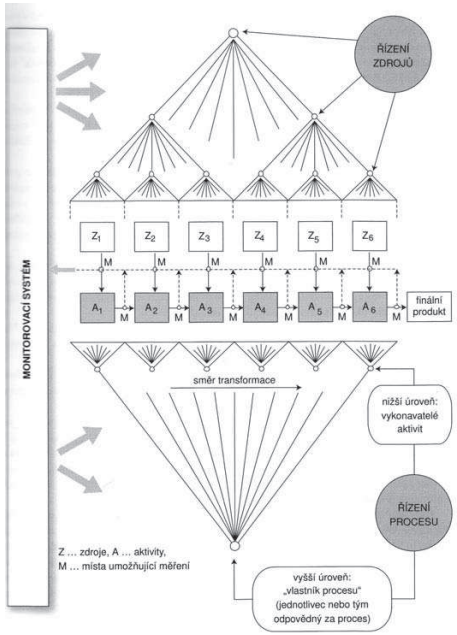
Řízení zdrojů - zajištění veškerého zázemí včetně péče o lidi
– individuálně se lze věnovat jen omezenému množství lidí
– specializovat se lze pouze na určité typy zdrojů
– počet úroveň řízení dle situace

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 vnějšího prostředí firmy, využito v procesech vedení a řízení, výsledek je změna

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



Obrázek 3 Definice procesů

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)

- nejprve procesy, potom zdroje – ochrana před zbytečnými investicemi do zdrojů, které nejsou třeba

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

Funkční firma se neobejde bez

- vedení – firemní myšlenky, jejich vytváření a prosazování, strategický rámec (definuje firmu, dává ji smysl, určuje směr vývoje) – role lídrů (často majitel nebo investor)
- řízení – aplikace firemních myšlenek do praxe – role manažera, zajišťování interních zdrojů, vstupů z externích zdrojů, definice a řízení procesů, garance vzniku a kvality výsledného produktu, vyhodnocování interních produktů, výsledků monitoring
- výkonu- bezprostřední vytváření produktů

Jeden člověk může hrát v různých souvislostech různé role. Funkční model firmy je na Obrázek 4 Funkční model firmy.

„Funkce dělá orgán" – nejdříve proces, potom struktura (musíme dát pozor na to, co je možné si dovolit vzhledem k nákladům) – uplatnit se má jen orgán odpovídající na potřebu

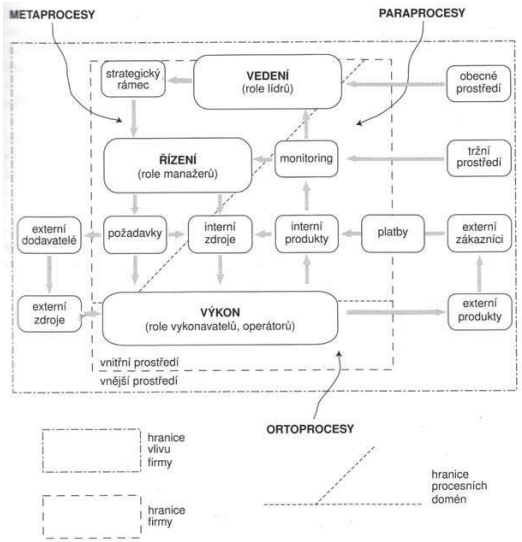
Rovnováha firemních dějů

- užitečnost+efektivita+zajištění zdrojů = rovnováha firmy
- lineární řízení – definování prvků zajišťujících užitečnost a efektivitu firmy probíhá opačně, než se tyto prvky zapojují do funkčního systému firmy: do firmy vstupují zdroje, ty jsou efektivně využívány, vznikají produkty, ty zajišťují zdroje pro další pokračování procesu (pojítko mezi produkty a zdroji jsou peníze) > stačí pro úspěch, pokud nedochází k vývoji a změnám (ale svět se samozřejmě neustále mění) -> nastavená rovnováha může být kdykoli porušena (mění se poptávka, trhy, konkurence, náklady na zdroje ad.) –> neustálé hledání rovnováhy
- schopnost hledat rovnováhu (adaptovat se na změnu) = **stabilita**

Peníze

- měřítko užitečnosti firmy pro všechny klíčové subjekty – priority
- nákup zdrojů podmiňujících poskytování užitku (energie, mzdy výrobních zaměstnanců, materiály)

- Platby daní, úvěrů
- Mzdy ostatních zaměstnanců
- Náklady považované za užitečné (marketingové náklady)
- Zisk majitelů a investorů
- Rozvoj způsobilosti lidí vykonávat svěřené aktivity (efektivita)
- Rozvoj výrobního zařízení, informačního systému (efektivita)
- Rozvoj zpětných vazeb, průzkumy, vzdělávání (stabilita)
- Zásadní inovace, vývoj, výzkum, prognózy, ovlivňování trhu (dynamika)



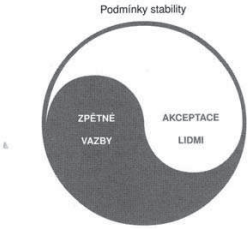
Obrázek 4 Funkční model firmy

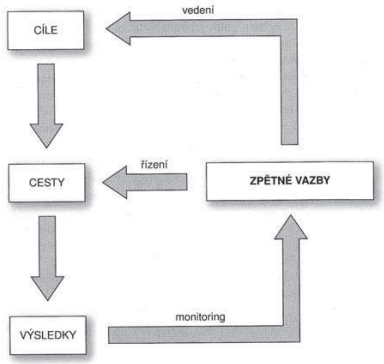
- pokud ne jsou první dvě patra postavena (užitečnost, efektivita), znamená to, že není nalezena rovnováha, objevují se znaky krize, uplatňuje se krizový management se všemi direktivními prvky (odvracení hrozeb)
- po dosažení rovnováhy nastupuje rozvojový management, mezi druhým třetím stupněm změna manažerského stylu

Zajištění stability

- pro zajištění stability platí dvě nezbytné podmínky: cyklické řízení a podpora 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)





Obrázek 5 Cyklické řízení

Strategický rámec firmy

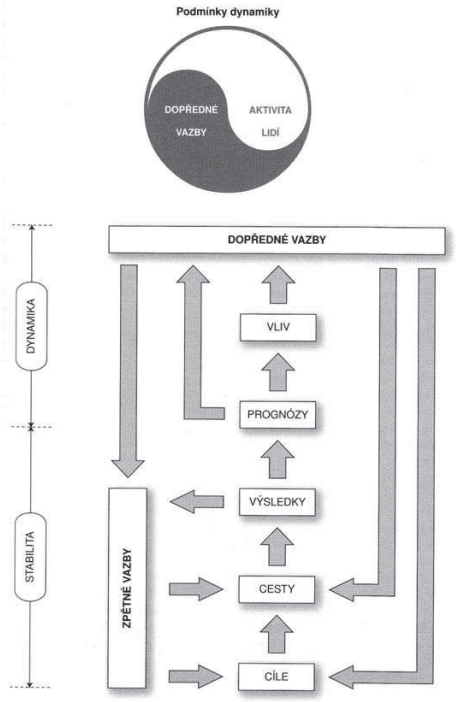
- podnikatelská hypotéza (podnikatelská příležitost + proč bude podnikání úspěšné – důležité např. i pro získání úvěru v bance)
- poslání – co firma poskytuje zákazníkům a dalším subjektům
- vize – představa, jak bude firma vypadat ve vzdálenější budoucnosti (musí odrážet vnitřní pocity lídrů, jinak se v krizovějších okamžicích bude rozhodovat jinak) – racionalizace vize – pro měření a vyhodnocování vize, měřitelná vize je strategický cíl firmy (strategický cíl firmy se týká celé firmy a vzdálené budoucnosti, je třeba z něho odvodit cíle pro kratší časové úseky až po operativní **úkoly**, tj. aktuální zadání – „naprogramuješ načítání dat ze souboru zasoby.txt do hash tabulky, která bude vypadat tak, že…“, a **úlohy**, tj. dlouhodobější platná zadání, jež mají jednotliví lidé plnit – „budeš udržovat část softwaru zodpovědnou za načítání dat z databáze do interních datových struktur, tj. třídy v balících…“)
- sdílené hodnoty - na základě skutečných interních hodnot lídrů – racionalizace – pravidla firemního života (každý si bude vést pracovní deník, do kterého bude vždy po čtyřech hodinách zapisovat, na čem pracoval; programátor může chodit do práce oblečený, jak uzná za vhodné;…)
- strategie – rámec pro definování systému metrik, pro různá období, tzv. operační strategické intervaly (všichni vědí, podle čeho jsou hodnoceni a odměňováni)

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

Zajišťování dynamiky

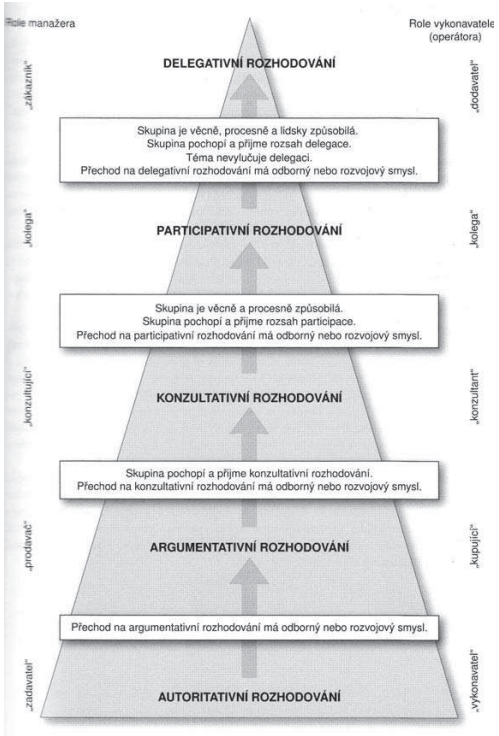
- firma je užitečná, efektivní a stabilní -> změny bere do své režie – ovlivňuje dynamiku vlastní i dynamiku svého okolí

- *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
  - 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



Obrázek 6 Proaktivní řízení

- *měkká podmínka*
  - 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
  - **!** 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



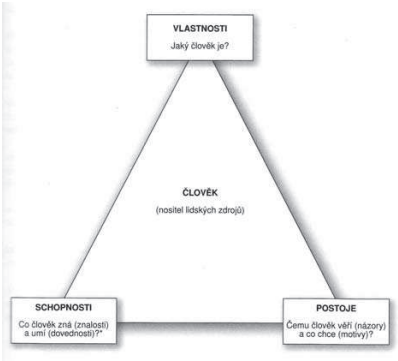
Obrázek 7 Pyramida rozhodování

- Dobrá taktika: opatrný pohyb po pyramidě vzhůru, každý krok vzhůru buď odborný smysl (zkvalitnění rozhodování) nebo rozvojový smysl (rozvoj podpory a aktivity lidí)

- každou expanzi limitují meze (trh, planeta Země) – expanze nad všechny limity je principiálně nemožná – přechod od kvantitativního růstu ke kvalitativnímu rozvoji (důraz na bezpečí, životní prostředí, kvalitní vztahy), redefinice představy užitečnosti, synergická seskupení s jinými subjekty

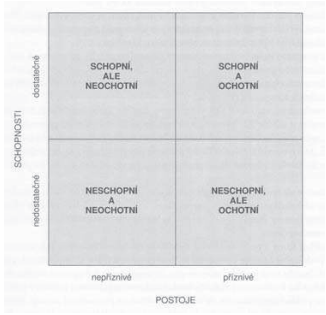
Vedení jednotlivců

- lidské zdroje = vlastnosti, schopnosti, postoje, jejich význam roste při zdolávání pyramidy vitality, ve fázi stability a dynamiky nutnou podmínkou úspěchu
- od nich přichází rozhodující vstup do firemních procesů - lidská práce
- 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"



Obrázek 8 Typy lidských zdrojů

- hrubá „diagnóza" jednotlivců na základě jejich schopností a postojů
    - umí, ale nechce – nenaléhavější problém, mohou být uznáváni pro své schopnosti ostatními, a tak se stát i vzorem chování pro své okolí, vzhledem k jejich nízké loajalitě je to nebezpečné
    - chce a umí – přirozená autorita a vzor chování v příznivých firemních podmínkách, chtěný zaměstnanec
    - chce, ale neumí – snaživí, ale ne úplně schopní lidé, jejich iniciativa není vždy prospěšná, nemají obdivovatele
    - neumí, ale nechce – těžko od nich něco čekat, nejsou napodobováni, nepředstavují akutní nebezpečí



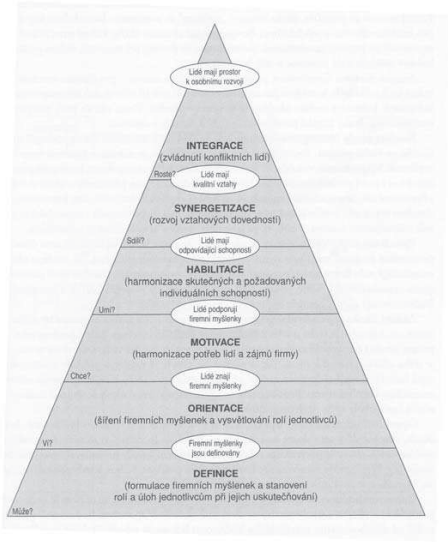
Obrázek 9 Postoje a schopnosti

- Z Obrázek 9 vyplývá strategie vedení a podpory jednotlivců
  - Udržet lidi v pravém horním kvadrantu, pokud tam jsou
  - Dostat lidi do horního pravého kvadrantu, pokud tam nejsou
    - Pohybem zleva doprava (zvýšením loajality)
    - Pohybem zdola nahoru (rozvojem schopností)
  - platí: nezvládnout neloajalitu je potenciálně nebezpečnější než přehlížet neschopnost – nejprve investice do pozitivní změny postojů, poté investice do rozvoje schopností

- změna postojů:
  - 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
  - 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“
  - 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**

- změna schopností
  - zaměstnanci jsou motivováni, schopnosti jsou však menší než nároky – můžeme snížit nároky, nebo rozvíjet schopnosti (přesně specifikovat cíl a formu vzdělávání) - **habilitace**
  - orientace, motivace i individuální schopnosti lidí jsou dostatečné, spolupráce vázne, chybí kvalitní vztahy (kurz týmové spolupráce, změna hodnocení, vliv chování managementu) - **synergetizace**
  - 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



Obrázek 10 Pyramida firemní kultury

Vedení skupin

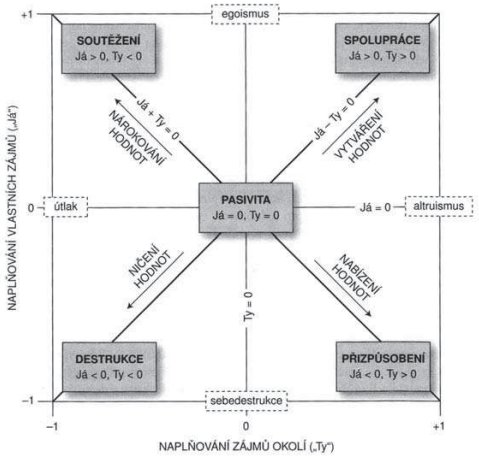
- pyramida firemní kultury – patro synergetizace

- proměna skupiny v tým a vznik specifické vlastnosti týmu – synergie – tým dosáhne jako celek více než kolik činí prostý součet příspěvků jednotlivých členů, úzce souvisí s mezilidskými vztahy

- synergii umožňuje dualita – koexistence sdílených společných myšlenek a zachování individuální diverzity (harmonie společného a rozdílného, sdílené nepotlačuje rozmanitost, individuální nedominuje nad společným)

Jak synergii vyvolat a udržet při životě?

- vztahové chování určitého subjektu – míra soustředění se na vlastní zájmy a na zájmy okolí



Obrázek 11 Vztahové chování

- *soutěžení* – prospěch na úkor jiných – kombinace nízké vstřícnosti a vysokého sebeprosazení, symbolická rovnice Já+Ty = 0 – distributivní linie (přerozdělování hodnot)

- *spolupráce* – o hodnoty se nebojuje, ale vytváří se (win-win strategie), v blízkosti symbolické rovnice Já-Ty = 0, proporciální linie

- *destrukce* – obě strany ztrácejí, hodnoty jsou ničeny, proporciální linie

- *přizpůsobení* – hodnoty jsou na úkor jedince distribuovány ve prospěch druhé strany, osou distributivní linie
    - ústupová taktika – pod tlakem okolností, protiváha úspěšného soutěžení
    - obětavost – přizpůsobení z vlastního, svobodného rozhodnutí

- při posuzování vztahů je třeba znát skutečné zájmy aktérů
- z pozice manažera je dobré mít vztahy ve skupině pod kontrolou
- vztahové chování není úplně náhodné, lze upravovat nastavováním určitých podmínek

1. **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 kompetitivní 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, prováženo 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.

2. **Podpora spolupráce**

    - 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

3. **Regulování obětavosti**

    - obětavost není dlouhodobě zdravou a udržitelnou taktikou – nositel něco ztrácí, je vhodná pro případ výjimečných, krizových situací, ne běžný požadavek na zaměstnance, nepřispívá k dlouhodobé vitalitě firmy
    - obětavost nepřináší problémy jen lidem ve vyšších patrech maslowovské pyramidy potřeb, jejich obětavost je účinnější

4. Předcházení destrukci

- zdroje destrukce:
    - pocit nespravedlnosti, spojený s nejasným nebo nespravedlivým ohodnocením a odměňováním (není důležité, jestli jde o nespravedlnost objektivní nebo subjektivní) – proto musí být hodnocení a odměňování lidí objektivně zdůvodnitelné, založené na srozumitelných kritériích a dobře vysvětlené
    - nedostatek pozornosti, přehlížení, pocit odcizení, vykořenění - proto součástí hodnocení je i vztahová informace

- rysy destrukce - malé sabotáže při plnění úkolů, krádež kancelářských potřeb, pocit nejistoty a dezorientace
- destrukce je nezdravou a dlouhodobě neudržitelnou taktikou – vede ke ztrátám v systému, k ničení hodnot, systém samotný je významně ohrožován
- destrukci se nejméně daří v přehledných a srozumitelných systémech, které se zajímají o své lidi

5. Zvládání pasivity

- uprostřed digramu vztahového chování
- vyvolána lhostejností, bezvýchodností (hodnoty, o které se hraje, připadají lidem bezcenné nebo nedosažitelné – přesvědčení člověka o ceně hodnot, či jejich dosažitelnosti
    - cena hodnot (na které potřeby může být člověk citlivý) – maslowská pyramida potřeb, postup odshora dolů
    - nedosažitelnost hodnot – v kořenech pasivity neschopnost nebo nejistota – řešení habilitace a orientace

- pasivita může přerůst v epidemii, pokud systém nenabídne dostatek podnětů, zvláště ze strany vedení a řízení, pasivita jako taková je tolerována nebo systém poskytuje příklady k napodobení
- pasivita není úspěšnou taktikou, objem hodnot v systému je nerostoucí funkce, v případě pasivity nutnost prevence nebo intervence

Potenciál pro generování úspěšného výsledku klesá v řadě
spolupráce-soutěžení-obětavost-pasivita-destrukce

Použitá literatura:

Plamínek J., Vedení lidí, týmů a firem, praktický atlas managementu, Grada, Praha, 2008

Další literatura:

- Berne Eric: Jak si lidé hrají (z angl. originálu Games people play), Dialog, Litvínov, 1992.
- Covey S.R., 7 návyků skutečně efektivních lidí (z anglického originálu The Seven Habits of Highly Effective People), Management Press, Praha, 2008.
- Goldratt E.M., Kritický řetěz (z angl. originálu Critical chain), InterQuality, Praha, 1999.
- Goldratt E.M., Jak vzniká zisk, Grada, Praha, 2004.
- Maslow A., Motivation and Personality, Harper, New York, 1970 [1954].

2. Zajišťování užitečnosti – požadavky na software

- následující texty a doprovodné obrázky jsou převážně převzaty z publikace Karla E. Wiegerse Požadavky na software

- z pohledu firmy vytvářející softwarový produkt
- nutnost definice výsledného produktu – nejtěžší fáze při vývoji softwaru
- komunikace důležitější než programování – snaha pochopit potřeby zákazníka
- spoléhání se na zdravý rozum

Co je to požadavek? – definic je mnoho, některé z nich

- cokoliv, co ovlivňuje rozhodování při návrhu
- IEEE Standard Glossary of Software Engineering:
    - Podmínka nebo funkce, kterou uživatel potřebuje pro řešení problému nebo dosažení nějakého cíle
    - Podmínka nebo funkce, kterou musí systém nebo jeho část splňovat, aby vyhověl smlouvě, standardu, specifikaci nebo jinému dokumentu, jenž se na něj formálně vztahuje
    - Dokumentovaná podoba některého z předchozích dvou bodů
- 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

- nikdy nepředpokládejte, že všichni účastníci projektu sdílejí stejnou představu, **co je to požadavek**
- požadavek není to, co uživatel nepotřebuje
- vývoj a řízení požadavků je složité (často se nedaří) – hledání postupů hodících se na větší počet situací (návrh nového sytému, udržování starého sw, vývoj krabicového sw)

Co bývá problém:

- Cíl a rozsah projektu nedefinované
- Zákazníci nemají čas na analytiky (vývojáře)
- Zástupci uživatelů (nejčastěji z managementu) si myslí, že mluví i za uživatele (podřízené), jejich požadavky však nedokážou popsat přesně
- Zákazníci tvrdí, že všechny požadavky jsou pro ně velmi důležité (nerozlišují priority)
- Požadavky drží „v hlavě“ odborníci, nejsou zdokumentovány
- Vývojáři si domýšlejí informace, které nemají
- Komunikace vývojář-zákazník se soustředí na obrázky uživatelského rozhraní, nikoli na průběh práce se softwarem
- Zákazník neustále mění již schválený seznam požadavků
- Změna požadavků zvětší rozsah softwaru bez dodání dalších zdrojů a upravení termínů
- Změny v požadavcích způsobí chaos – nikdo nemá přehled o aktuálním platném stavu
- Naprogramovanou funkcionalitu nikdy nikdo nepoužije
- Hotový systém plní specifikaci, ale zákazník není spokojený

Z množiny **subjektů, kterým budeme užiteční**, nás budou zajímat zejména zákazníci a zaměstnanci

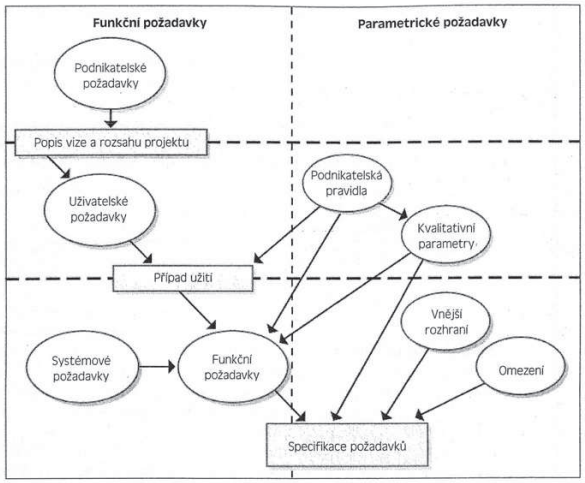
- Zákazníci
    - Majitelé (investoři) zákaznické firmy – projekt financují, chtějí dostat systém, který pokryje jejich podnikatelské potřeby
    - Management - potřebují podpořit a monitorovat procesy (také často zároveň re/definovat procesy nebo dlouhodobě podporovat změnu procesů), podpořit definici zdrojů a struktur,…
    - Uživatelé (výkonní pracovníci), kteří se systémem přímo či nepřímo pracují – potřebují jasné, přímočaré a uživatelsky komfortní ovládání
- Zaměstnanci
    - Analytik požadavků – sepisuje požadavky na systém a tlumočí je vývojářům (definuje produkt a orientuje vedoucího projektu, vývojáře, testery, dokumentátory,…)
    - Vývojáři – navrhují, implementují, udržují systém
    - Testeři – zjišťují, jestli se systém chová tak, jak má (zajišťují systém zpětných vazeb)
    - Dokumentátoři – píšou uživatelské příručku, nápovědy, …
    - Vedoucí projektu – odpovídá za řízení projektu (zajištění efektivity, definice úloh a kompetencí,…)
    - Technici, právníci, marketingové oddělení,…- zajišťují paraprocesy

Potřeby subjektů

- zákazníků:
    - Majitelé (investoři): podpora vitality firmy – podpora ekonomických výsledků, bezpečnost, podpora a šíření strategického rámce (forma předání poslání, vize a sdílených hodnot – např. informační portál), prestiž
    - Management - podpora definice a řízení procesů, podpora definice zdrojů a struktur (informační systémy, systémy pro řízení výroby, systémy pro definici a změnové řízení procesů,…), záruka, servis
    - Výkonní pracovníci – zachování pracovního místa (nesmí jim hrozit, že je vyvíjený systém „zbaví“ místa), podpora a orientace při pracovním výkonu, snadné ovládání (podpora motivace)
- zaměstnanců sw firmy:
    - definice výsledného produktu je východisko pro volbu efektivních postupů (stanovení procesu – modelu vývoje sw, definici zdrojů a struktur – týmu vývojářů), konkrétních úloh a kompetencí, orientace vývojářů, zpětných vazeb (testů), případně dopředných vazeb (nabídky dalších funkčností, které by mohly zákazníkům prospět)
    - Bod, od kterého začíná firma software reálně vyvíjet

Definice produktu – požadavky na software

- 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)
- funkční požadavky (také behaviorální požadavky, požadavky na chování)
    - 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
- 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,…
- omezení
    - některé cesty při návrhu a vývoji nemohou být použity (např. existuje požadavek na konkrétní OS, nutnost integrace se stávajícím systémem, apod.)



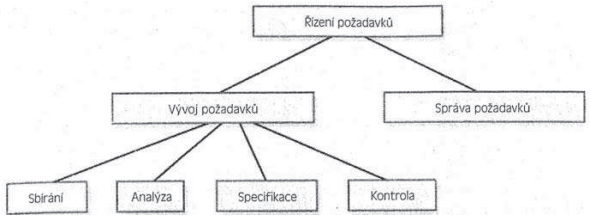
Obrázek 12 Požadavky a vztahy mezi nimi

- funkce (features) – skupina logicky souvisejících funkčních požadavků (vhodné např. pro marketing)
- seznam funkcí (např. kontrola překlepů, online aktualizace virové databáze) je něco jiného než úplný popis požadavků na funkčnost

- několik iterací celého procesu, postupné upřesňování podnikatelských, uživatelských i funkčních požadavků – vznikne Dokument specifikace požadavků (může být i tabulka, databáze, …) – popisuje očekávané chování celého systému

- Otázka: Patří nový požadavek, nový případ užití nebo nová funkce do vymezeného rozsahu systému?
    - Ano - zařadíme
    - Ne – vyřadíme
    - Ne, ale měl by – otázka pro investora, zda upraví rozsah projektu

- Co nepatří do specifikace požadavků – podrobnosti o návrhu a implementaci, informace o plánování projektu a testování, požadavky na projekt (rozpočet, školení,…)



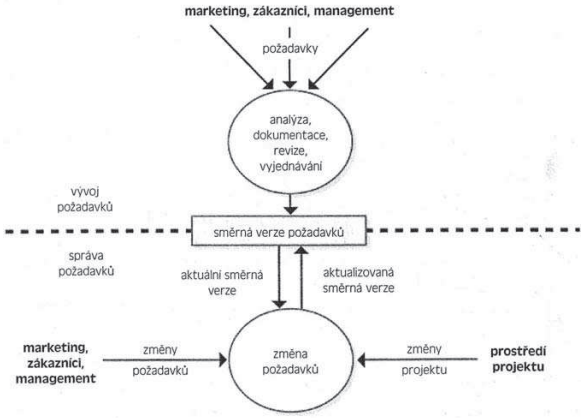
Obrázek 13 Jednotlivé disciplíny práce s požadavky

Vývoj požadavků

- sběr
- analýza – upřesňování požadavků tak, aby jim všichni účastníci projektu rozuměli
- specifikace - dokumentování požadavků tak, aby byly jednotně popsané, přístupné a daly se snadno změnit
- kontrola (validace)

Správa požadavků

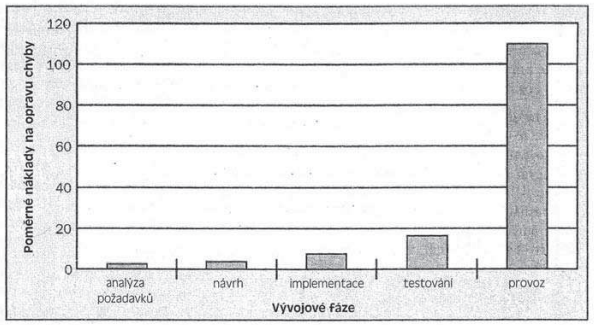
- shoda zákazníka a zhotovitele na požadavcích na softwarový produkt a udržování této shody
- shoda zakotvena v psané specifikaci požadavků a modelech
- Metoda postupného upřesňování představ (některé požadavky se dají plně specifikovat až po zahájení vývoje) – jedna návštěva zákazníka rozhodně nestačí
- Složité je hledání požadavků, nikoli jejich sepisování



Obrázek 14 Hranice mezi vývojem a správou požadavků

Proč se analýza požadavků často nepodaří?

- nedostatečné zapojení uživatelů
- změny a přidávání požadavků bez souvisejících nároků na termín, zdroje,…
- nejednoznačné požadavky – je možné interpretovat požadavky různým způsobem – formální posouzení požadavků, psaní testovacích scénářů, prototypů
- šperkování – přidávání funkcí, o kterých si vývojář myslí, že je bude uživatel potřebovat
- příliš malá specifikace – počítá se s tím, že si vývojáři specifikaci dotvoří
- přehlédnuté skupiny uživatelů
- nepřesné odhady – nedělejte žádné odhady, dokud nemáte dostatek informací, odhady udávejte rozsahem (v nejlepším případě - nejspíš – v nejhorším případě)



Obrázek 15 Porovnání nákladů na opravu chyby podle toho, kdy byla nalezena

Vlastnosti dobře napsaných požadavků

- 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

Vlastnosti celé specifikace

- Úplnost – žádný požadavek nechybí
- Jednotnost – žádný požadavek není v rozporu s jiným požadavkem stejného typu anebo požadavkem vyšším – je třeba řešit co nejdříve s autorem požadavku
- Přizpůsobitelnost – specifikace musí být přepsatelná a musí být k dispozici nástroje pro vedení historie změn – každý požadavek jednoznačné označení
- Dohledatelnost – u požadavku se dá vysledovat zdroj, návrh, příslušný zdrojový kód i testovací scénáře
- Specifikace, která splňuje vše výše uvedené stoprocentně, **se napsat nedá**

Vztah mezi zákazníkem a sw firmou

- Důvěra
- Podpora spolupráce (společný cíl,…)
- Desatera, shoda na nich – nebojte se strávit čas diskusí o tom, jak co nejlépe spolupracovat

Základní práva softwarového zákazníka:

1. očekávat, že s vámi analytik bude mluvit vaší řečí
2. očekávat, že se analytik poučí o fungování vaší firmy a vašich cílech
3. očekávat, že analytik získané požadavky zpracuje do podoby psané specifikace
4. nechat si od analytika vysvětlit všechny výstupy získané během zpracování požadavků
5. očekávat, že se k vám analytici budou chovat s respektem a budou udržovat profesionální a vstřícný přístup
6. očekávat, že vám analytici a vývojáři budou předkládat nápady a varianty požadavků a jejich implementace
7. zadat požadavky na systém tak, aby se dobře používal
8. dostat příležitost na takovou změnu požadavků, aby mohl být při řešení použit již existující software
9. získat při každé změně upřímný odhad nákladů a dalších souvisejících záležitostí, které změna způsobí
10. dostat systém, který splňuje všechny vaše požadavky na funkce a kvalitu

Základní povinnosti softwarového zákazníka:

1. vysvětlit analytikům a vývojářům procesy ve své firmě a vysvětlit jim firemní žargon
2. obětovat tolik svého času, kolik bude třeba pro sdělení vyjasnění a postupné upřesnění všech požadavků
3. být při popisování požadavků co nejkonkrétnější a nejpřesnější
4. rozhodovat o požadavcích včas
5. respektovat vývojářův odhad ceny a složitosti implementace vašich požadavků
6. ve spolupráci s vývojáři rozdělit prioritu jednotlivých funkčních požadavků, funkcí systému a případů užití
7. důkladně přečíst specifikaci požadavků a vyzkoušet prototypy
8. hlásit změny požadavků včas
9. provádět změny požadavků podle procesu, který mají vývojáři pro tyto případy nachystaný
10. respektovat procesy, které analytik používá pro práci s požadavky

Často se specifikace požadavků podepisuje

- 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
- všechny změny probíhají dle předem stanoveného procesu
- zákazníkův management má jistotu, že se rozsah projektu nevymkne z jeho rukou, všechny změny musí schválit
- uživatelé (zástupci uživatelů) mají jistotu, že s nimi vývojáři budou pracovat na vývoji správného systému, i když je některé věci napadnou až po zahájení prací
- management vývojářů má kvalitního partnera, který s vývojáři spolupracuje na udržení rovnováhy termínů, rozsahu, zdrojů a kvality
- analytici požadavků mají jistotu, že změny budou probíhat tak, aby nezpůsobily chaos
- vývojáři ví, co mají implementovat

Dobré zvyky (best practices)

- založeny na zdravém rozumu – lepší než rozsáhlé a detailní metodologie
- v tabulkách sedm kategorii zvyků

Tabulka 1 Dobré zvyky pro řízení požadavků 1

Znalosti	Správa požadavků	Řízení projektu
■ Poskytněte analytikovi požadavků dostatečné školení.	■ Stanovte si proces pro řízení změn.	■ Zvolte vhodný životní cyklus projektu.
■ Vysvětlete zástupcům uživatelů i jejich vedení práci s požadavky.	■ Založte komisi pro řízení změn.	■ Plánujte na základě požadavků.
■ Poskytněte vývojářům školení v aplikační doméně.	■ Pro každou změnu proveďte analýzu důsledků.	■ Po změnách požadavků se dohodněte na nových podmínkách.
■ Udělejte si projektový slovník.	■ Verzujte specifikaci požadavků.	■ Analyzujte a popište rizika spojená s jednotlivými požadavky.
	■ Veďte si historii změn.	■ Sledujte úsilí věnované řízení požadavků.
	■ Veďte si poznámky o stavu jednotlivých požadavků.	■ Poučte se ze starších projektů.
	■ Sledujte stabilitu požadavků.	
	■ Používejte nějaký nástroj pro správu požadavků.	
	■ Vytvořte si spojovací matici.	

Vývoj požadavků

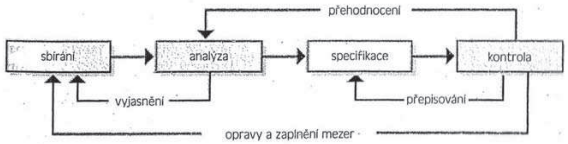
Sbírání	Analýza	Specifikace	Kontrola
■ Stanovte si proces pro vývoj požadavků.	■ Nakreslete si kontextový diagram.	■ Pište specifikaci podle šablony.	■ Proveďte revizi požadavkové dokumentace.
■ Popište vizi a rozsah projektu.	■ Prototypujte.	■ Najděte zdroje jednotlivých požadavků.	■ Testujte požadavky.
■ Najděte třídy uživatelů.	■ Analyzujte proveditelnost požadavků.	■ Každý požadavek označte jedinečným identifikátorem.	■ Definujte kritéria pro přijetí systému uživateli.
■ Vyberte produktové šampióny.	■ Rozdělte požadavky podle priority.	■ Zapište si podnikatelská pravidla.	
■ Vyberte skupinu typických uživatelů.	■ Udělejte si model požadavků.	■ Zapište si kvalitativní parametry.	
■ Najděte případy užití.	■ Udělejte si datový slovník.		
■ Najděte systémové události a odpovědi na ně.	■ Rozdělte požadavky mezi podsystémy.		

Tabulka 2 Dobré zvyky pro řízení požadavků 2

Vývoj požadavků

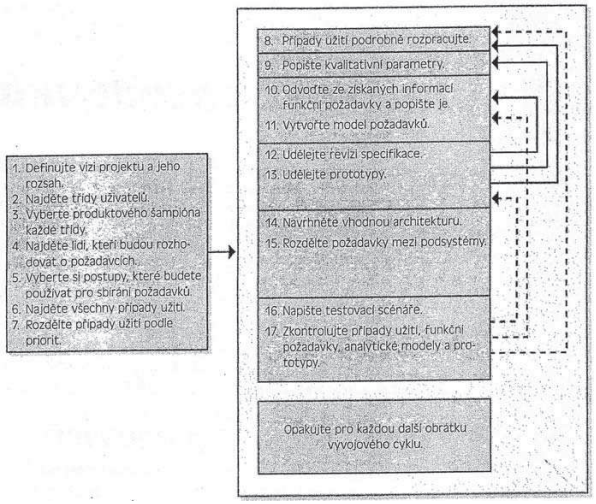
Sbírání	Analýza	Specifikace	Kontrola
■ Uspořádejte požadavkový workshop.	■ Použijte QFD.		
■ Sledujte uživatele při práci.			
■ Projděte si požadavky na zlepšení stávajícího systému.			
■ Recyklujte požadavky ze starších projektů.			

- doporučený postup zavádění nových zvyků do praxe je k dispozici viz Tabulka 3 a Tabulka 4
- vývoj požadavků je iterativní proces (má zpětné vazby – zajištění stability)



Obrázek 16 Iterativní proces vývoje požadavků

- neexistuje univerzální přístup k vývoji požadavků
- možná šablona jak na to (po drobných úpravách vhodná pro většinu projektů)
- prvních sedm kroků se většinou provádí jen jednou za začátku projektu, jen priority je třeba občas přehodnotit, ostatní kroky se provádí u každé iterace nebo verze
- způsob sbírání požadavků vyberte podle toho, jaký máte přístup k uživateli
- rozdělení případů užití podle priorit zejména v případě, že systém stavíte přírůstkově, můžete se rozhodnout, v které verzi se který případ objeví
- u nových systémů nebo větších zásahů můžete navrhnout novou architekturu nebo vylepšit stávající



Obrázek 17 Proces pro vývoj požadavků

Tabulka 3 Zavádění dobrých zvyků do praxe 1

Vliv na projekt	Obtížnost		
	Malá	Střední	Velká
Velký	■ Poskytněte vývojářům školení v aplikační doméně.	■ Najděte případy užití.	■ Stanovte si proces pro vývoj požadavků.
	■ Definujte vizi a rozsah projektu.	■ Zapište si kvalitativní parametry.	■ Plánujte podle požadavků.
	■ Najděte třídy uživatelů.	■ Rozdělte požadavky podle priority.	■ Po změnách požadavků se dohodněte na nových podmínkách.
	■ Nakreslete si kontextový diagram.	■ Pište specifikaci podle nějaké šablony.	
	■ Najděte zdroje jednotlivých požadavků.	■ Stanovte si proces pro řízení změn.	
	■ Verzujte specifikaci požadavků.	■ Založte komisi pro řízení změn.	
		■ Proveďte revizi požadavkové dokumentace.	
		■ Rozdělte požadavky mezi podsystémy.	
		■ Zapište si podnikatelská pravidla.	
Střední	■ Analyzujte proveditelnost požadavků.	■ Poskytněte analytikovi požadavků dostatečné školení.	■ Vysvětlete zástupcům uživatelů i jejich vedení práci s požadavky.
	■ Udělejte si projektový slovník.	■ Vyberte si produktové šampióny.	■ Vytvořte si model požadavků.
	■ Udělejte si datový slovník.	■ Vyberte skupinu typických uživatelů.	■ Analyzujte a popište rizika spojená s jednotlivými požadavky.
	■ Sledujte uživatele při práci.	■ Prototypujte.	■ Používejte nějaký nástroj pro správu požadavků.
	■ Najděte systémové události a odpovědi na ně.	■ Definujte kritéria pro přijetí systému uživateli.	■ Vytvořte si spojovací matici.
	■ Každý požadavek označte jedinečným identifikátorem.	■ Pro každou změnu proveďte analýzu důsledků.	■ Uspořádejte požadavkový workshop.
	■ Požadavky testujte.	■ Zvolte si vhodný životní cyklus projektu.	

Tabulka 4 Zavádění dobrých zvyků do praxe 2

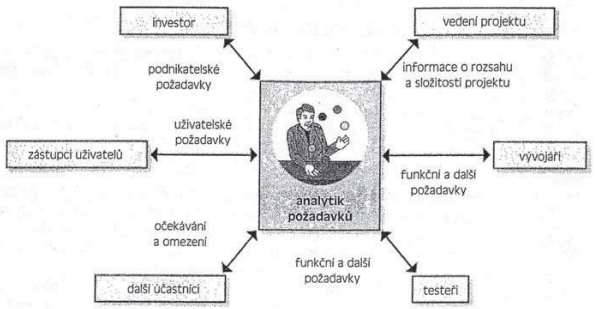
Vliv na projekt	Obtížnost		
	Malá	Střední	Velká
	■ Sledujte stav každého požadavku.		
	■ Poučte se ze starších projektů.		
Malý	■ Hledejte nápady v požadavcích na zlepšení starého systému.	■ Sledujte stabilitu požadavků.	■ Recyklujte požadavky z předchozích projektů.
		■ Sledujte úsilí věnované řízení požadavků.	■ Použijte QFD.
			■ Veďte si historii změn.

Analytik požadavků

- zodpovědnost za sbírání, analýzu, dokumentaci a kontrolu požadavků
- může zastávat jeden, nebo i více lidí

Úkoly analytika:

- definice uživatelských požadavků
- hledání účastníků a tříd uživatelů
- sběr požadavků
- analýza požadavků
- psaní specifikace
- modelování požadavků
- kontrola hlavních požadavků
- rozdělení požadavků podle priority
- správa požadavků



Obrázek 18 Analytik jako komunikační most mezi zákazníkem a vývojářským týmem

Vlastnosti, schopnosti a postoje analytika

- Umění naslouchat
- Umění ptát se a vést rozhovor
- Analytické dovednosti
- Moderátorské schopnosti
- Pozorovací schopnosti
- Vyjadřovací schopnosti
- Organizační schopnosti
- Modelovací schopnosti
- Sociální inteligence
- Nápaditost
- Ovládnutí technik pro řízení požadavků a jejich správné nasazení
- Schopnost řídit projekt a rizika
- Znalost aplikační domény

Analytik

- bývalý uživatel
- bývalý vývojář (ne příliš časté)
- oborový expert

Vývoj požadavků

Vize produktu a rozsah

- nejvyšší úroveň abstrakce – podnikatelské požadavky – definují vizi sw systému a jeho rozsah
- uživatelské a funkční požadavky se musí přizpůsobit kontextu a cílům plynoucím z podnikatelských požadavků
- požadavky, které nepomáhají v dosažení podnikatelských cílů, se neberou v úvahu
- protichůdné podnikatelské požadavky musí rozhodnout investor (v žádném případě analytik požadavků nebo dokonce vývojář)
- podnikatelské požadavky říkají, které úkoly, tj. případy užití bude systém podporovat (šířku aplikace), a do jaké hloubky budou implementovány (musí být jasné, které podnikatelské požadavky žádají obsáhlou implementaci a které stačí implementovat povrchně)
- podnikatelské požadavky ovlivňují prioritu implementace jednotlivých případů užití
- projekt bez jasného směru si koleduje o obrovský průšvih

- 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)
- směrná verze dokumentace požadavků – podrobný rozsah projektu

- jasná vize a rozsah ještě důležitější u projektů, kde týmovou spolupráci komplikuje fyzická vzdálenost
- pokud chybí sdílené, roste význam individuálních priorit, nefunguje spolupráce, účastníci si mohou i nevědomky škodit
- diagnostický prvek, že jsou nedostatečně definované uživatelské požadavky: nějaká funkce se z projektu odstraní a pak se zase přidá
- musí být jasné předtím, než se pustíte do podrobné specifikace funkčních požadavků

- vize a rozsah každého projektu – mohou být samostatné dokumenty v případě velkých projektů, v našem případě vše v jednom dokumentu
- dokumentace vize a rozsahu projektu (project charter, business case document, market requirement document,…) - pomáhá při diskusi o navrhovaných funkcích a verzích

Příklad podnikatelských cílů jsou v Tabulka 5 Příklady podnikatelských cílů.

Tabulka 5 Příklady podnikatelských cílů

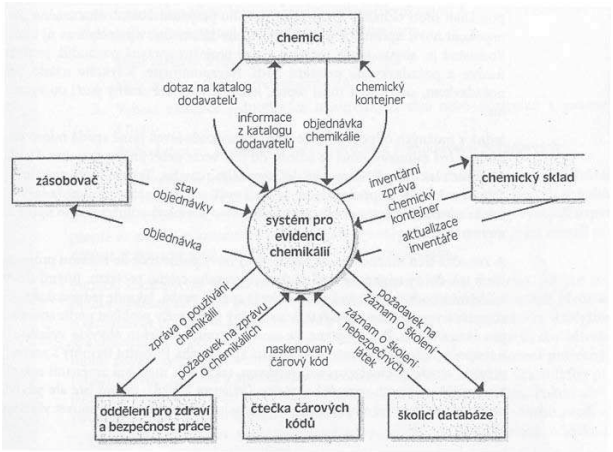
Finanční	Nefinanční
■ Dosáhnout do X měsíců Y% podíl na trhu.	■ Dosáhnout během X měsíců práce uživatelské spokojenosti alespoň Y.
■ Dosáhnout v zemi X do Y měsíců Z% podíl na trhu.	■ Zvýšit produktivitu při zpracování transakcí na X % a snížit chybovost na maximálně Y %.
■ Dosáhnout do X měsíců prodeje v objemu Y kusů nebo Z Kč.	■ Dosáhnout určeného termínu pro vydání produktu, abychom obsadili trh.
■ Dosáhnout X% zisku nebo návratnosti investice.	■ Vyvinout robustní platformu pro rodinu příbuzných produktů.
■ Dostat se u výrobku během X měsíců do černých čísel.	■ Zlepšit v organizaci schopnosti práce s konkrétní technologií.
■ Ušetřit X Kč, které momentálně každý rok spolkne udržování starého systému.	■ Získat v odborném tisku do určeného dne alespoň X pochvalných recenzí.
■ Ušetřit během X měsíců Y % nákladů na podporu.	■ Dostat se ve srovnávacích testech do stanoveného data na pozici nejbezpečnějšího produktu.
■ Vyřídit během X měsíců po prodeji každého kusu nanejvýš Y hovorů na technickou podporu a nanejvýš Z reklamací.	■ Vyhovět konkrétním státním předpisům.
■ Zvýšit hrubý zisk stávajícího podnikání z X % na Y %.	■ Zkrátit u X % telefonických hovorů na technickou podporu dobu vyřízení na Y hodin.

Vize

- můžeme vytvořit např. podle šablony
  Co [název produktu]
  Je [kategorie, do které produkt patří]
  Pro [cílový zákazník]
  Který [potřeba nebo příležitost]
  Poskytne [klíčové výhody, hlavní důvod pro koupi nebo nasazení]
  Náš produkt [hlavní odlišnosti a výhody nového produktu]
  Na rozdíl od [hlavní konkurent, současný systém, současný proces]

Rozsah projektu

- popisuje hranici a spojení mezi vyvíjeným systémem a okolím
- grafické vyjádření – kontextový diagram – diagram datových toků (viz později) na nejvyšší úrovni abstrakce
  - 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
- srozumitelnost pro všechny zúčastněné strany je důležitější než jedině "správné" kreslení kontextových diagram



Obrázek 19 Příklad kontextového diagramu

Jak udržet rozsah projektu

- Projekt se může rozšířit, pokud přijde nový, nebo upřesněný požadavek zákazníka
- Je nutné správně posoudit, jestli se dle vize a rozsahu projektu navrhované požadavky do projektu hodí (jestli navrhované změny patří do rozsahu projektu)
  - Požadavek patří mimo rozsah – musí se vložit do jiné verze/jiného projektu
  - Požadavek je uvnitř rozsahu – zařadíme, pokud mají vyšší prioritu než ostatní požadavky (je možní, že budeme muset odložit nebo úplně zrušit implementaci jiných požadavků)
  - Požadavek mimo rozsah projektu, ale je to výjimečně dobrý nápad (pokud máme rezervu, zapracujeme, nemáme rezervu, použijeme stanovený proces pro řízení změn)
- Dobře dokumentované požadavky umožňují snadněji odmítnout, pokud se „vlivní" snaží do napjatého projektu dostat další funkce

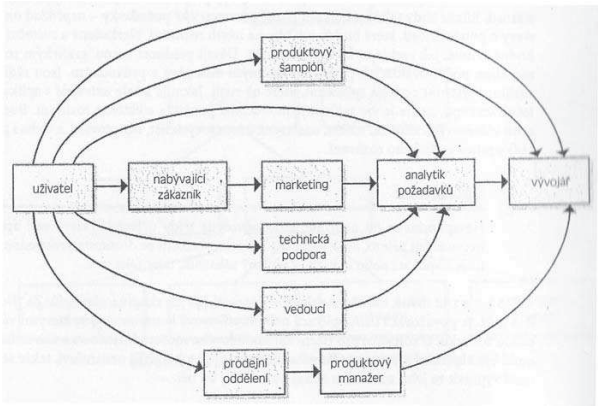
Sběr požadavků aneb Jak u zákazníka

- 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
- Zákazníci na začátku nevědí, co chtějí (nestačí se na začátku jednoduše zeptat)
- Iterativní proces – zabere hodně času (avšak vyplatí se to)

Třídy uživatelů

- Jeden z uživatelů může patřit do více tříd
- Kandidáty jsou každý z koncových prvků v kontextovém diagramu
- Liší se
    - Frekvencí používání systému
    - Zkušenostmi s aplikační doménou a výpočetní technikou
    - Používanými funkcemi
    - Úkoly ve firmě
    - Oprávněním (uživatel/správce/host,…)
- Nedělit uživatele podle jejich fyzického umístění, typu firmy nebo postavení ve firmě
- Některé uživatelské třídy jsou důležitější než jiné (dle podnikatelských cílů projektu), dostanou vyšší prioritu
- Uživatelská třída může být i další aplikace nebo hardwarová komponenta
- Pozor na uživatele, kteří chtějí mluvit za jiné třídy uživatelů

- Úspěšnější projekty – větší počet komunikačních kanálů
- Každý prostředník mezi uživatelem a vývojářem zvyšuje pravděpodobnost nedorozumění



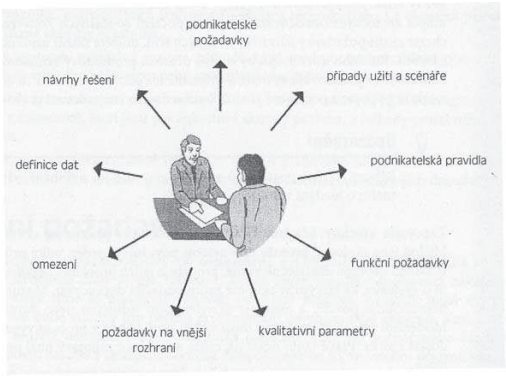
Obrázek 20 Příklady komunikačních kanálů mezi uživatelem a vývojářem

Sběr požadavků - techniky

- Nejdůležitější, nejsložitější, komunikačně nejnáročnější část vývoje sw
- 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
- Požadavkové workshopy (DrPV)

Třídění získaných informací

– analytik musí dodané informace roztřídit (zákazník roztříděné informace nedodá☺)

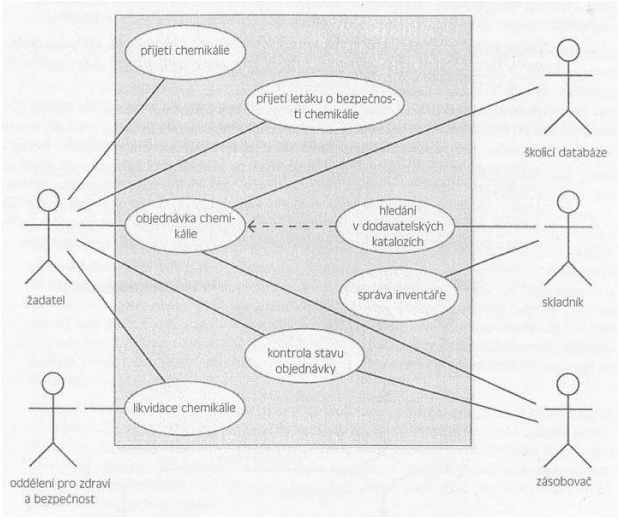


Obrázek 21 Třídění získaných informací

- Podnikatelské požadavky
    - cokoli, co popisuje ekonomické, tržní nebo jiné podnikatelské výhody, které zákazníci od sw očekávají, poznají se podle zmínek o hodnotě, kterou zákazníci dostanou
    - *Zvýšíme podíl na trhu o X procent.*
    - *Potřebujeme ročně ušetřit X korun, které polyká údržba staršího systému.*
    - *Potřebujeme proniknout do <tohoto> segmentu trhu.*
    - *Potřebujeme zvýšit obrat v internetovém obchodu.*

- Případy užití a scénáře
    - obecné údaje 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…)
    - Jedna konkrétní cesta skrze případ užití je scénář užití
    - cesta od jednotlivých scénářů k obecnějším případům užití

    - *Potřebuji vytisknout nálepku s poštovní adresou.*
    - *Potřebuji převést data do jiného formátu.*
    - *Potřebuji nastavit parametry zařízení, než ho spustím.*



Obrázek 22 Příklad diagramu případu užití

- Podnikatelská pravidla
    - Některou z aktivit smí provádět pouze vybrané třídy uživatelů ve vybraných podmínkách
    - odvozují se od nich také funkční požadavky

    - *Systém musí odpovídat zákonu XXX předpisu YYY.*
    - *Systém musí splňovat <nějaký standard>.*
    - *Pokud <je splněna podmínka> (uživatel prošel školením), musí (může) se< něco stát> (může nastavit parametry zařízení).*
    - *Vzorec pro výpočet tlaku v nádobě zní takto:…*
- Funkční požadavky
    - Popisují viditelné chování systému za daných podmínek a úkoly, které by jeho prostřednictvím měli dělat uživatelé
    - Tvoří převážnou část specifikace požadavků

    - *Pokud přesáhne teplota 80 stupňů, musí být informován zvukovým znamením informován vedoucí provozu.*
    - *Seznam načtených slov musí jít abecedně setřídit.*
    - *Je možné vypsat všechna slova začínající zvoleným prefixem.*

- Kvalitativní parametry
    - Popisují kvalitu chování systému (poznají se podle slov r*ychle, 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

- Požadavky na vnější rozhraní
    - Popisují spojení mezi systémem a okolním prostředím
    - Rozhraní vůči uživatelům, hardwaru a jiným softwarovým systémům

    - *Je třeba načítat signály z <nějakého zařízení>.*
    - *Je třeba posílat zprávy nějakému <jinému systému>.*
    - *Je třeba zapisovat soubory <v takovém a takovém formátu>.*
    - *Prvky uživatelského rozhraní musí odpovídat <nějakému standardu>.*

- Omezení
    - Berou vývojářům některá možná řešení
    - software musí respektovat fyzická omezení
    - Důvod omezení je třeba si zapsat, aby ostatní respektovali jeho nutnost
    - Technologická omezení, nutnost držet se standardních prostředků zvoleného programovacího jazyka

    - *Elektronicky odevzdané dokumenty nesmí být větší než 10MB.*
    - *Všechny transakce musí být zabezpečené minimálně na úrovni…*
    - *Musí být použita pouze standardní knihovna Javy.*
    - *Systém nesmí vyžadovat více než X MB paměti.*
    - *Systém musí běžet na <tomto>hardwaru.*

- Definice dat
    - Popis formátu, datových typů, přípustných hodnot, výchozích hodnot atd.
    - *Telefonní číslo se skládá právě z devíti číslic.*

Návrhy na řešení

    - Konkrétní způsob práce se systémem, není to požadavek
    - Uživatel: *„A pak si v rozbalovacím menu vyberu stát“* –> funkční požadavek: „umožnit uživateli vybrat stát“
    - Návrh na konkrétní řešení je v podstatě omezení požadavku

- Požadavky je dobré organizovat
- Je možné, že zjistíte, že je rozsah systému příliš velký nebo naopak malý – předělání produktové vize a rozsahu systému
- Pokud bychom kvůli rozhodnutí potřebovali provést výzkum, chápejme to jako samostatnou část projektu se samostatnými požadavky a cíli (-> prototypování)
- Pokud projekt vyžaduje větší výzkum, použijeme inkrementální vývojový model (požadavky jsou analyzovány po malých částech s malým rizikem)

Hledání chybějících požadavků

- rozveďte požadavky do podrobností
- zkontrolujte, jestli máte požadavky od všech uživatelských tříd
- systémové požadavky sledujte, seznamy událostí, případy užití,… sledujte až k podrobným funkčním požadavkům
- hlídejte hraniční případy
- zapisujte požadavky v různých podobách, na různých úrovních abstrakce
- ve skupinách požadavků provázaných booleovskou logikou sledujte pokrytí všech možných situací
- CRUD (CRUDL) matice, C- create, R – read, U – update, D – delete, L – list

případ užití \ entita	Objednávka	Chemikálie	Zásobovač	Dodavatelský katalog
Vytvoření objednávky	C	R	R	R, L
Změna objednávky	U, D		R	R, L
Správa chemického inventáře		C, U, D		
Výpis seznamu objednávek	R	R, L	R, L	
Změna Zásobovače				

Obrázek 23 Příklad CRUDL matice

Jak poznáme, že je sběr požadavků u konce

- Neexistuje jednoduchý ukazatel
- Bod, za kterým už další snaha nepřináší výrazné výsledky
- Uživatelé nemohou přijít na žádný další případ užití
- Nové případy užití nepřinesou nové funkční požadavky
- Témata se opakují
- Nově navrhované funkce, uživatelské i funkční požadavky jsou mimo rozsah projektu
- Nové požadavky mají nízkou prioritu
- Uživatelé navrhují funkce „pro budoucí verze“

Dokument specifikace požadavků

1. Dobře strukturované a pečlivě psané dokumenty v přirozeném jazyce
2. Grafické modely (transformace, stavy systému a přechody, vztahy mezi daty, třídy objektů)
3. Formální specifikace – matematicky přesný formální jazyk

- Ani nejlepší specifikace nenahradí osobní komunikaci v průběhu projektu

- Pro několik cílových skupin
  - Zákazníci, marketingové oddělení, prodejní oddělení – definice produktu
  - Projektový manažer – odhady termínů, náročnosti, potřebných zdrojů
  - Vývojáři – implementace sw
  - Testovací tým – vývoj testovacích plánů, scénářů, postupů
  - Technická podpora – jak se daná část systému bude chovat
  - Dokumentátoři – tvorba manuálů a nápovědy
  - školitelé – výukové materiály
  - Právní oddělení – specifikace musí vyhovovat zákonům
  - Subdodavatelé – podle specifikace pracují
- Nemusí být hotová před začátkem vývoje, ale požadavky na každou další část systému musíme mít hotové před tím, než se do vývoje této části pustíme
- Směrná verze – požadavky, které prošly kontrolou a byly schváleny účastníky projektu
- Specifikace musí být srozumitelná všem účastníkům
- Požadavky musí být jednoznačně identifikovány
- Ve specifikaci i návrhy uživatelského rozhraní (popisují řešení, nikoli požadavky) – návrhy obrazovek - nemohou nahradit uživatelské a funkční požadavky

Šablony pro specifikaci požadavků

- standardních šablon je hodně

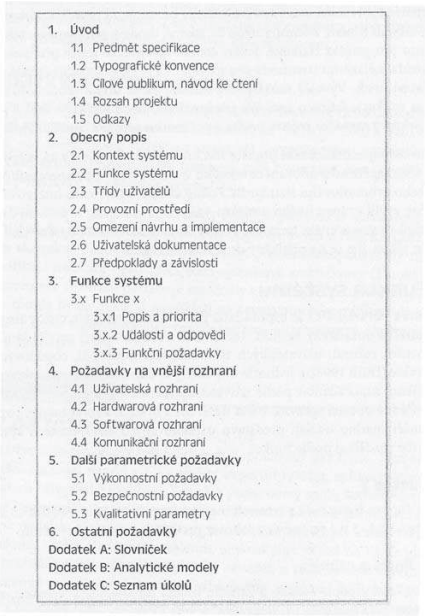
1. **dokument specifikující požadavky na systém a dokument specifikující požadavky na software**
   - **dokument specifikující požadavky na systém** (angl. Concept of Operations, ConOps document; používají se ještě další názvy)
     - vysokoúrovňový popis požadavků z hlediska zadavatele (musí se vyjadřovat v termínech zadavatele, protože ho budou číst také zástupci zadavatele)
     - kromě seznamu požadavků obsahuje informace o celkových záměrech systému, cílovém prostředí, omezeních, předpokladech a mimofunkčních požadavcích
     - může obsahovat modely kontextu systému, případy užití, toky informací a práce apod.
     - slouží pro validaci systémových požadavků

   - **dokument specifikující požadavky na software** (angl. Software Requirements Specification, SRS)
     - v češtině termín Dokument specifikace požadavků (DSP)
     - podrobná specifikace požadavků na software, odvozená z požadavků na systém
     - předpokládá se, že čtenáři už mají ponětí o SW inženýrství => jazyk může být přesnější, notace podrobnější
     - měl by obsahovat specifikaci uživatelských i systémových požadavků
       - ty mohou být sdruženy v jednom popisu
       - často lepší uvést uživatelské požadavky v úvodu k systémovým požadavkům
       - pokud by byl rozsah dokumentu neúměrný, je možné vytvořit systémové požadavky jako samostatné dokumenty
       - obsahuje oficiální vyjádření o tom, co se od vyvíjeného systému očekává => pro produkty vyvíjené na zakázku může sloužit jako základ kontraktu

Různé velké organizace definovaly vlastní standardy a doporučení pro strukturu obou typů dokumentů, např. IEEE std 1362-1998 (struktura ConOps) a IEEE std 830-1998 (struktura DSP).

2. Dokument specifikace požadavků podle **IEEE/ANSI 830** (IEEE Guide to Software Requirements Specifications) - zjednodušená verze viz Obrázek 24.

- Poznámka – dokument DSP k dispozici na portálu
- osnova je odvozená ze standardu IEEE 830 (MS Word)
- soubor můžete využít - před odevzdáním z něj zrušte nápovědu
- rozumným způsobem si ho přizpůsobte
- body, které se netýkají vaší úlohy, můžete odbýt poznámkou **Není**



Obrázek 24 Šablona pro specifikaci požadavků

Způsoby specifikace požadavků

Přirozený jazyk

- obvyklý a nevyhnutelný popis požadavků - výhodou srozumitelnost pro uživatele i pro vývojáře
- nevýhody
  - nejednoznačnost popisu
  - složité koncepce (např. algoritmy) je obtížné popsat přesně
  - příliš flexibilní - stejná věc se dá říci mnoha různými způsoby; jak čtenář zjistí, že se požadavky liší a čím?
  - neexistuje jednoduchý způsob modularizace - jak zjistíte důsledek změny požadavků, tj. kterých všech dalších požadavků se změna dotkne?
- nutnost používat jazyk jednoduše, snažit se vyhnout běžným příčinám chybné interpretace

- Příklad problematické definice: *Pokud uživatel zadá jméno delší, než je šířka formuláře, jeho pokračování se zobrazí na následující obrazovce. (Nejednoznačné: zobrazí se na následující obrazovce pokračování jména, nebo formuláře?)*

- Vyhněte se:
  - dlouhým souvětím s mnoha vedlejšími větami
  - používání termínů s několika přijatelnými významy
  - prezentaci několika požadavků jako jediného požadavku
  - nekonzistenci v používání termínů, např. používání synonym

- slabá místa specifikace lze najít hledáním výskytu neurčitých slov nebo frází (několik, především) apod., proto je snaha omezit nejednoznačnost (např. formuláře nebo pseudokódy)

Formuláře

- přirozený jazyk příliš flexibilní, někdy se přidává struktura pomocí formulářů
- jeden nebo více typů
- obsah:
  - popis specifikované funkce nebo entity
  - popis vstupů a odkud se berou
  - popis výstupů a kam putují
  - jaké další entity specifikovaná funkce nebo entita používá
  - případné vstupní a výstupní podmínky (pre-conditions a post-conditions), tj. co platí při vstupu do funkce a co při výstupu z ní
  - pokud vznikají vedlejší efekty, pak jejich popis

Příklad systémové specifikace funkce pro vkládání prvku do diagramu, zapsaná ve formě formuláře:

Funkce: Vlož prvek do diagramu

Popis: *Vloží prvek do existujícího diagramu. Uživatel určí typ prvku a jeho pozici.*

Vstupy: *Typ prvku, Pozice prvku, Identifikátor diagramu.*

Zdroje: *Typ prvku a Pozici prvku zadá uživatel, Identifikátor diagramu získáme z databáze diagramů.*

Výstupy: *Identifikátor diagramu.*

Úložiště: *Databáze diagramů. Při dokončení operace je proveden COMMIT.*

Vyžaduje: *Diagram odpovídající vstupnímu Identifikátoru diagramu.*

Vstupní podmínka: *Diagram je otevřen a zobrazen na obrazovce uživatele.*

Výstupní podmínka: *Diagram je nezměněn kromě přidání prvku určeného typu na určenou pozici.*

Vedlejší efekty: *Nejsou.*

Případy užití

- viz výše, podrobně ASWI

Pseudokódy

- někdy je funkce specifikována jako posloupnost jednodušších akcí, pořadí je **podstatné**
- v přirozeném jazyce bychom obtížně vyjadřovali např. vnořené podmínky nebo smyčky
- vhodné doplnit specifikaci popisem v pseudokódu

- vlastnosti:
  - jazyk s abstraktními konstrukcemi, které právě potřebujeme (sekvence jednoduchých příkazů, iterace, větvení):
  - vnoření konstrukcí je vyjádřeno odsazením
  - vyhýbáme se syntaktickým konstrukcím cílového programovacího jazyka (které by nás zbytečně omezovaly a sváděly k programování), popisujeme požadovaný záměr, nikoli jak bude v cílovém jazyce implementován
  - musí umožňovat téměř automatickou konverzi do kódu (pokud příliš vysoká úroveň, nemusíme postřehnout problémy ve specifikaci)

Příklad (část popisu činnosti bankomatu):

Přečti kartu

Vypiš výzvu: Prosím zadejte PIN

Přečti zadané_PIN

Opakuj nejvýše 3x

Přečti zadané_PIN

Jestliže zadané_PIN je PIN_karty, pak opusť smyčku

Jestliže zadané_PIN není PIN_karty pak ...

Formuláře vs. pseudokód

- formuláře - celková specifikace systému
- pseudokód - řídící sekvence, rozhraní

Specifikace rozhraní

- pokud systém komunikuje s dalšími systémy, musí být přesně specifikováno softwarové nebo komunikační rozhraní
- specifikace rozhraní měla by být součástí DSP, pokud je rozsáhlá, tak v příloze

2 typy rozhraní, které musí být definovány:

- procedurální rozhraní - existující podsystémy poskytují množinu služeb, které jsou přístupné prostřednictvím volání procedur rozhraní (např. *popis knihovních procedur nebo tříd programovacího jazyka, v klasických jazycích např. prototyp procedury nebo fce, vstupně/výstupní parametry, popis činnosti, návratová hodnota, v objektových jazycích např. obecný popis třídy, popis konstruktorů, popis metod)*
- popis předávaných dat - popis struktury dat - pro popis se používají datové modely (ERA diagramy) + popis reprezentace dat, např. *datum je reprezentován jako řetězec*

Validace požadavků

- vstupem úplný DSP
- zjišťujeme, zda jsou požadavky úplné, konzistentní a zda odpovídají tomu, co zadavatel od systému chce

- u požadavků se kontroluje:
  - *platnost*
    - uživatel si myslí, že systém má poskytovat určité funkce, ale podrobnější analýza může ukázat něco jiného
    - různí uživatelé mají různé požadavky, budou kompromisy platné?
  - *konzistence* - požadavky v dokumentu nesmějí být v konfliktu, tj. nesmějí být různé popisy nebo různá omezení stejné fce
  - *úplnost* - definovány by měly být všechny fce a omezení systému
  - *realizovatelnost* - systém je implementován s danými prostředky a v daném čase
  - *ověřitelnost* - systémové požadavky jsou napsány tak, že jsou ověřitelné (ušetříme si dohadování se zákazníkem při předávání produktu)
  - *sledovatelnost* původu požadavku - abychom dokázali odhadnout dopad změny

Použitelné metody:

- přezkoumání (reviews)
- prototypování
- tvorba testů
- automatická analýza konzistence

Přezkoumání (reviews)

- požadavky jsou systematicky zkontrolovány týmem (manuální proces, více čtenářů od zákazníka i od dodavatele)
- vývojový tým provází klienta systémovými požadavky, vysvětluje důsledky každého požadavku, kontroluje konzistenci, úplnost atd.

Prototypování

- zákazníkovi předvedeme spustitelný model systému, může zjistit, zda odpovídá požadavkům (typicky chování uživatelského rozhraní)
- nevýhoda prototypů - pozornost uživatele odvádějí kosmetické záležitosti nebo omezení prototypu (proto někdy papírové modely obrazovek)

Generování testovacích případů

- vytvořený test požadavků často odhalí problémy
- pokud je obtížné vytvořit test, bude požadavek obtížně implementovatelný

Automatická analýza konzistence

- pro modely ve formální nebo strukturované notaci – spíše teoretická nežli praktická záležitost

Správa požadavků

- požadavky na velký systém se neustále mění -> nutný proces řízení změn systémových požadavků
- trvalé a nestálé požadavky
  - trvalé požadavky - relativně stabilní, odvozeny z podnikatelských cílů organizace nebo z aplikační domény (např. v nemocnici vždy pacienti, lékaři a sestry)
  - nestálé požadavky - pravděpodobně se změní během vývoje nebo po uvedení systému do provozu (např. nemocnice - pravděpodobně se změní systém plateb od zdravotních pojišťoven)

- začíná plánováním, rozhodne se:
  - způsob identifikace požadavků - každý požadavek bude mít jedinečný identifikátor, abychom ho mohli odkazovat (křížové reference apod.)
  - proces změny požadavků - definujeme proces, abychom se ke změnám požadavků chovali konzistentním způsobem
  - sledovatelnost - které vztahy mezi požadavky navzájem atd. budeme uchovávat a jak (čím více, tím dražší)
    - zdroj požadavku - kdo požadavek navrhl
    - vztahy mezi požadavky v DSP; pro určení kolika požadavků se změna dotkne
    - vztahy mezi požadavky a designem systému; pro určení dopadu změny na systémový design a implementaci
  - jaké nástroje se použijí pro uchovávání informací o požadavcích (malé projekty - postačují prostředky jako textové a tabulkové procesory, databáze; velké projekty - CASE nástroje)

Proces změny požadavků

- všechny navrhované změny požadavků by měly podléhat procesu o třech základních krocích:
  - analýza problému a specifikace změny
  - analýza změny a určení její ceny (ve výsledku přijdeme za zákazníkem: stálo by to X, budete to chtít teď nebo později?)
  - implementace změny (ve výsledku modifikace DSP, případně designu a implementace)

Literatura:

Wiegers K. E., Požadavky na software, Computer Press, a.s., Brno, 2008.

3. Zajišťování efektivity – Softwarový proces, definice zdrojů a struktur

Softwarový proces

- odpovídá obecné definici procesu (opakovaně se opakující transformace vstupu na výstup), všechny aktivity a mezivýsledky musí mít smysl!
- množina aktivit a (mezi)výsledků, jejichž výsledkem je softwarový produkt
- dle takto stanoveného předpisu se následně definují projekty (tzv. instanciace procesu), které jsou vztažené k jednotlivým zakázkám

- existují různé sw procesy, protože
    - vytvářený sw je různorodý – pro různé typy systémů vhodné rozdílné procesy
    - lidé mají různé schopnosti, postoje a vlastnosti (není dobře, když zdroje a struktury definují proces - Kapitola 1, ale reálně to tak v IT často bývá)

- 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)

- úroveň definice a hodnocení softwarového procesu je hodnocena dle stupnice SEI (Software Engineering Institute) a vyjadřuje vyspělost organizace z daného hlediska - model vyspělosti a schopností dodavatele se nazývá CMMI (Capability Maturity Model Integration, http://www.sei.cmu.edu/cmmi/), dříve CCM (Capability Maturity Model)

- proces je jedinečný pro každou firmu

Modely sw procesu (-> modely životního cyklu sw produktu, metodiky)

- zobecněný popis procesu nebo části procesu, resp. popis procesu či jeho části z určitého pohledu
- existuje řada modelů, žádný však nebyl přijat (a ani být z principu přijat nemůže) jako referenční (základem a „prarodičem“ všech je tzv. vodopádový model)
- vhodně zvolený model může významně zvýšit efektivitu softwarového vývoje
- v jednodušších případech (neexistuje ucelený systém) nemluvíme o modelu, ale o pravidlech (často pokrývají jenom např. zadání projektu a jeho nasazení, vývojáři dodržují všeobecně uznávaná pravidla, tzv. best practices, mohou z nich vybírat)
- pro různé části projektu se mohou hodit různé modely

Zaměření a rozsah modelů

- velmi podrobné (definují všechny procesy v rámci vývojového cyklu) – definovány zpravidla většími firmami, ve kterých je vývoj softwaru důležitým prvkem činnosti (konzultační firmy, organizace vyvíjející interní software – banky, pojišťovny, apod.)
- komplexní modely (např. RUP – viz později)
- méně podrobné (a také obecnější) – popisují celkovou koncepci vývoje, role pracovníků, postupy testování, apod.
- zaměřené na určitou fázi vývojového cyklu (sběr požadavků, testování,…)
- existence vlastního firemního modelu – vnitrofiremní pravidla aplikovaná na obecnější model -> výhody:
    - nový pracovník si přečte dokumenty – na některé věci přijde rychleji – není třeba se mu tolik věnovat
    - prevence před problémy sw vývoje – méně koncepčních chyb, případně jejich rychlejší a snadnější náprava
    - snadnější a úspornější komunikace v týmu – některé postupy jsou předem dány

– **modely zaměřené na technickou část projektu**
    - důraz především na návrh aplikace, fázi kódování a testování
    - chybí úvodní fáze projektu, sběr požadavků (nezávislá na technologii) – chybí natolik, že se ukazuje ->

vývoj sw musí být postaven tak, aby umožňoval co největší interakci se zákazníkem (důležitější než např. použitá technologie)

Vytváření a dostupnost modelů

- vývoj modelu stojí velké náklady (je to v podstatě intelektuálně velmi náročný sw produkt)

- často doménou konzultačních firem (nedostatečná komunikace se zákazníkem měla u nich největší dopad) – model nebývá veřejně dostupný
- modely zveřejněné na internetu, knižně apod. (vytvořené obvykle na univerzitách či v open-source komunitách) – volně k použití
- CASE nástroje - vytváření softwarového produktu + konkrétní model pokrývající určitou fázi vývojového procesu (drahý nástroj)

Rizika a slabiny použití modelů

- Obecný model musí umožňovat více variant pro různé typy projektů (malé projekty vs. velké projekty – jiné nároky na řízení - vedoucí projektu se musí rozhodnout, kterou variantu a případně pro kterou část projektu použít – toto rozhodnutí je velmi důležité)
- Použití nesprávných, příliš náročných a složitých modelů – nižší efektivita vývojového týmu („utopení se“ v modelu, některé aktivity nemají smysl, vznikají nepotřebné interní produkty)
- Všichni členové týmu musí modelu porozumět a akceptovat jej – je třeba čas k porozumění i k akceptaci (viz tzv. „měkká“ podmínka stability) - postupné zavádění a ukázky výhody vzniklých postupů
- Pozor na přehnaný důraz na dokument (dokument vytvořený jen kvůli naplnění modelu a nepotřebný pro projekt = ztráta času) – výsledný produkt je funkční zdokumentovaný program, nikoli sada dokumentů
- Model je průvodce projektem (vede nás k cíli, tj. fungujícímu programu, méně snadno zabloudíme, někdy ho však můžeme slepě následovat a dostat se na scestí)
- Model sám o sobě nenahradí zkušené vývojáře, ani technologii

- Mnoho specialistů na modely vůbec nerozumí technologii – najímají si externí programátory – pokud není dobře zvládnuta komunikace, vzniklé produkty nebývají nejkvalitnější (i když splňují zadání)
- Nutné přizpůsobení modelu vlastním potřebám (je však nutné vědět či dobře odhadnout, co ne/můžeme vynechat)

Důležitost modelu aneb klíčové faktory pro úspěch projektu

1. Podpora firemního vedení
2. Angažovanost uživatelů
3. Zkušený projektový manažer
4. Jasně stanovené podnikatelské cíle
5. Minimalizovaný rozsah
6. Standardní softwarová infrastruktura
7. Pevné základní požadavky
8. Formální metodika
9. Spolehlivé odhady
10. Jiná kritéria

- 70% úspěchu = body 1-5
- 97% úspěšných projektů = zkušený projektový manažer v čele
- 70% aplikačního kódu je infrastruktura

Podle výzkumu Standish group

Modely a certifikáty

- normy ISO
- model – odborné rozhodnutí vs. certifikace – obchodní rozhodnutí
- obecně certifikace v oblasti vývoje softwaru dnes zřejmě přeceňována
- certifikace může znamenat jen formalizaci dosud používaného modelu
- různé typy certifikací
    - ne certifikace používané pro výrobní procesy (mají zaručit jejich opakovatelnost) – proces vývoje sw produktu zpravidla opakovatelný není
    - ano certifikace zaměřené na projektově orientované organizace a činnosti – počítají s tím, že každý projekt představuje nový a neopakovatelný úkol, požaduje se, aby všechny projekty stejného typu měly stejnou strukturu činností
    - ano certifikace na dodržení kvality (v oblasti vývoje sw produktu především testování, údržba, tvorba dokumentace) – nelze však kvalitu zajistit pouhým dodržováním pravidel

Přehled modelů sw procesu

„Pravěk“ softwarového inženýrství - 60. léta

- vývoj sw je náročný, nejlepší specialisté, neuspokojivé výsledky
- první formalizované postupy, jak úspěšně tvořit software
- veškerá práce kolem počítačů (i využívání aplikací) – akademicky vzdělaní specialisté, vědci, ekonomové apod. – předpoklad podrobných teoretických znalostí
- předpoklad teoretických znalostí i u zadavatelů úkolů – přesné formální definice požadavků na začátku vývojového cyklu
- postupy složité, nepružné, odstraňování chyb v zadání nákladné
- cíl si určuje programátor na základě vágního zadání, časový termín vágní, požadavky na dotažení programu až na výjimky žádné
- všeobecně uznávaný model – **vodopád**

Vodopádový model (waterfall model)

- výše uvedené čtyři aktivity chápe jako samostatné fáze procesu
- princip vodopádu – vývojový cyklus rozdělen na několik přesně definovaných fází - po splnění se fáze ukončí a přechází se na další (tj. neexistují zpětné vazby)

Evoluční modely vývoje

- specifikace, vývoj a validace jsou smíšeny
- ze specifikace je velmi rychle vyvinut prvotní systém
- na základě požadavků zákazníka je systém dále upravován

Formální modely vývoje (formální transformace)

- na začátku formální (matematická) specifikace systému, která je posléze transformována do funkčního programu
- transformace zachovávají správnost – na konci víme, že program odpovídá specifikaci
- v běžné praxi není použitelné

Sestavení systému ze znovupoužitelných (reusable) komponent

- předpokládá se, že části systému již existují
- vývoj systému se zaměřuje na integraci těchto částí
- podíl takto vytvářeného softwaru roste

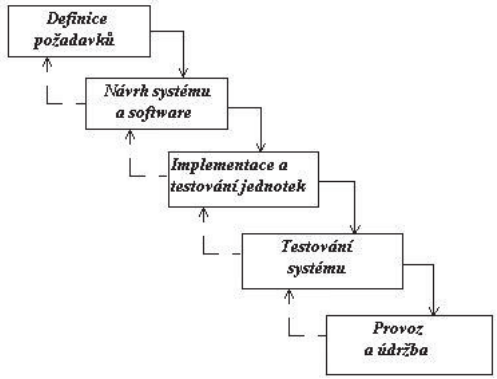
Iterativní modely – inkrementální vývoj a spirálové modely

- hybridní modely – různé přístupy pro různé části systému (odpovídají potřebě tvorby velmi rozsáhlých systémů)

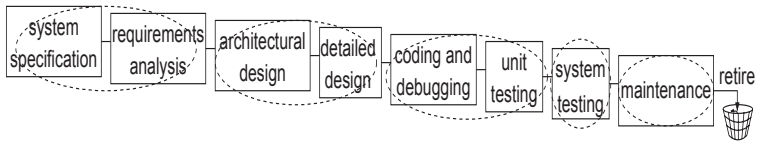
- **Neexistuje nejsprávnější model**, různé modely jsou vhodné pro různé situace
- v praxi různé kombinace vodopádového modelu (nejlépe rozumově postihnutelné, organizovatelné) v kratších časových úsecích (řádově týdny), evolučního a iterativního vývoje (důraz na potřeby zákazníka).

Vodopádový model vývoje SW

- první model vývoje SW
- schémata se mohou značně lišit; viz následující dva obrázky (rozdělení do jednotlivých fází můžeme nazvat různě, principy ovšem zůstávají tytéž):



Obrázek 25 Vodopádový model vývoje sw 1



Obrázek 26 Vodopádový model vývoje sw 2

Přehled základních aktivit:

Analýza domény, získávání a definice požadavků

- viz kapitola 2, definice sw produktu (proces zahrnuje definování užitečnosti)

Návrh systému a návrh SW (system&software design)

- určení celkové architektury systému
- podrobný návrh SW – identifikace a popis základních abstrakcí a jejich vztahů (často grafická notace, např. UML)

Implementace a testování jednotek (modulů)

- realizace návrhu – množina modulů, balíků, tříd, programů…
- testování jednotek – ověření, že každá jednotka odpovídá specifikaci

Integrace a testování systému

- jednotlivé moduly jsou sestaveny do výsledného systému
- úplný systém je otestován na shodu se specifikací
- po otestování je systém předán zákazníkovi

Provoz a údržba (maintenance)

- nejdelší fáze životního cyklu – systém se prakticky používá
- údržba = změny software vynucené provozem (nikoli administrace systému), tj. oprava chyb implementace (a také návrhu, či specifikace), které nebyly odhaleny v předchozích fázích + rozšiřování systému podle nových požadavků

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

Výhody a 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
  - problémy s vodopádovým modelem vedly k formulaci alternativních modelů

- 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

Evoluční modely vývoje SW

základní myšlenky:

- vytvořit počáteční implementaci a vystavit jí komentářům uživatele (zákazník se dostává do centra pozornosti)
- počáteční implementaci postupně vylepšovat přes další meziverze tak dlouho, dokud nedostaneme adekvátní výsledek (vše na základě další interakce se zákazníkem a jeho dalších, nových, pozměněných atd. požadavků)
- tj. fáze specifikace, vývoje a validace neprobíhají vždy sekvenčně, ale mohou probíhat i paralelně, mezi fázemi je zpětná vazba
- modely vedené riziky

2 základní typy modelů evolučního vývoje SW:

1. model výzkumník (také evoluční protypování nebo průzkumný vývoj)
2. model prototyp (throw-away prototyping, tj. doslova tvorba prototypu, který bude zahozen)

Klady a výhody evolučního vývoje SW:

- vede efektivněji než vodopádový model k systému splňujícímu bezprostřední požadavky zákazníka
- specifikace může být vytvářena postupně
- zákazník vidí, že se něco děje

Problémy evolučního vývoje SW:

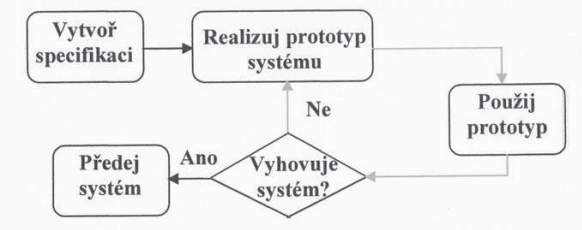
- proces není viditelný - manažeři nemohou měřit postup vývoje (na rozdíl např. od vodopádového modelu), problematická organizace z pohledu dodavatele
- systémy jsou často špatně strukturované (neustálé změny vedou k porušení struktury systému) -> vývoj dalších verzí se postupně stává obtížnější a dražší

Model výzkumník (evoluční protypování)

- 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

Výhody a nevýhody modelu výzkumník:

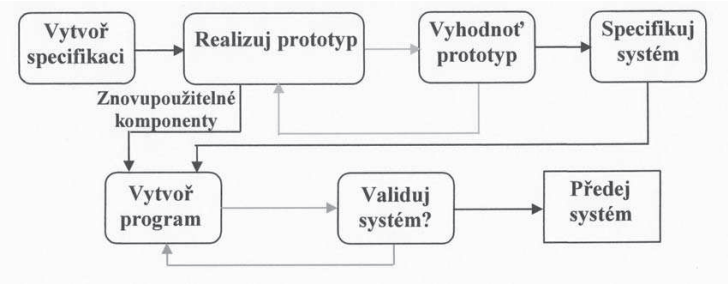
- systém je dobře přizpůsobitelný i dodatečným požadavkům zákazníka
- jednotlivé etapy lze stěží plánovat časově, finančně i personálně
- dokumentace je většinou zhotovena po dokončení projektu (málokdy psána průběžně)
- při údržbě není jasné, které požadavky jsou původní (byly stanoveny při zadání) a které jsou nové



Obrázek 27 Model výzkumník – evoluční prototypování

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



Obrázek 28 Specifikační protypování

Výhody a nevýhodu modelu prototyp

- prototyp je testován a ověřován budoucími uživateli -> je upřesňována specifikace požadavků
- náročný na vedení v případě existence třeba i několika nezávislých vývojových skupin (zvýšené náklady projektu)
- může vyžadovat speciální nástroje a techniky -> problematický výstup, např. nekompatibilní s jinými nástroji a technikami
- kvalitativní požadavky (mimofunkční) nelze testovat
- nebezpečí, že bude prototyp dopracován a použit

Evoluční vývoj SW vhodný pro

- malé systémy
- středně velké systémy s krátkou dobou života
- ve velkých systémech by evoluční vývoj působil zřetelné problémy, varianta smíšeného přístupu
    - špatně srozumitelné části - throw-away prototyping => zpřesnění DSP
    - dobře srozumitelné části – iterativní model (řada menších vodopádů)
    - uživatelské rozhraní - model výzkumník

Formální modely vývoje SW

- určitým způsobem podobné vodopádovému modelu
- na počátku je vytvořena specifikace požadavků; ta je zpřesněna do podrobné formální specifikace systému (pojmem formální se zde míní "matematická")
- postupné zpřesňování specifikace formálními transformacemi, konečným výsledkem spustitelný program
- v každém kroku se přidávají podrobnosti, ale můžeme dokázat, že odpovídá reprezentaci systému z předchozího kroku
- proces designu, implementace a testování jednotek je nahrazen formálními transformacemi

Výhody a nevýhody formálního modelu:

- zejména tam, kde je třeba dokázat zákazníkovi bezpečnost a spolehlivost systému
- nepoužitelný pro klasické problémy a klasického zákazníka
- vyžaduje specializované znalosti - výběr transformace vyžaduje zkušenosti lišící se od klasického programování
- někdy vyžaduje transformaci do speciálního programovacího jazyka (často čím nižší programovací jazyk, tím obtížnější převod)
- v praxi se takřka nepoužívá

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

2 modely navržené speciálně pro iterativní procesy vývoje:

- inkrementální vývoj SW (Lehman 1969, Mills 1980)
- spirálový model vývoje (Boehm 1988) a WinWin spirálový model (Boehm a Bose, 1994)

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

Průběh procesu

- definují se přehledové požadavky (někdy pouze cíl projektu)
- požadavky se přiřadí jednotlivým inkrementům
- navrhne se architektura systému

- opakovaně pro každý inkrement - dokud nevytvoříme finální systém: vyvineme a validujeme (otestujeme) inkrement, integrujeme inkrement a validujeme systém

Model přepokládá, že

- na začátku zákazník identifikuje přehledově služby, které od systému požaduje
- zákazník určí priority, tj. které služby jsou pro něj nejdůležitější a které jsou nejméně důležité
- pak je definována množina inkrementů, každý inkrement poskytuje podmnožinu celkové funkčnosti (alokace služeb do inkrementů závisí na prioritě služby, nejdůležitější služby mají být zákazníkovi předány jako první)
- v dalším kroku jsou detailně specifikovány požadavky na první inkrement
- 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

Klady a výhody inkrementálního přístupu

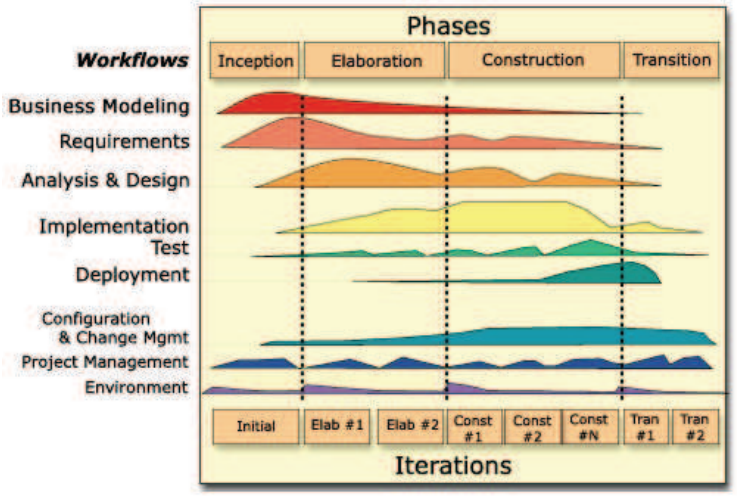
- zákazníci nemusejí čekat na dokončení celého systému, aby z něj něco měli
- počáteční inkrementy jim mohou pomoci získat zkušenost pro definici dalších inkrementů
- je menší riziko selhání celého projektu; i když některé inkrementy mohou být problematické, je pravděpodobné, že většina jich bude úspěšně předána zákazníkovi
- protože zákazníkovi předáváme jako první inkrementy s nejvyšší prioritou, budou nejdůležitější části systému nejlépe otestovány; jinými slovy - zákazníka pravděpodobně nepotká havárie nejdůležitější části systému

Problémy inkrementálního modelu

- inkrementy by měly být relativně malé (do cca 20 000 řádek kódu)
- každý inkrement by měl poskytovat nějakou navenek viditelnou funkčnost - proto může být obtížné namapovat požadavky zákazníka na inkrementy správné velikosti
- problematický inkrement může zpětně ovlivnit již hotovou část systému (při detailnější specifikaci požadavků inkrementu se zjistí, že se předtím na něco zapomnělo), oprava nemusí být triviální
- většina systémů obsahuje množinu základních služeb, které jsou vylaďovány různými částmi systému, v inkrementálním modelu vývoje se ale požadavky nespecifikují detailně, dokud se nedostaneme k jejich vývoji => je obtížné identifikovat služby potřebné pro všechny inkrementy - proto je pro větší projekty vhodné doplnit inkrementální model o fázi analýzy, podobnou jako ve vodopádovém modelu

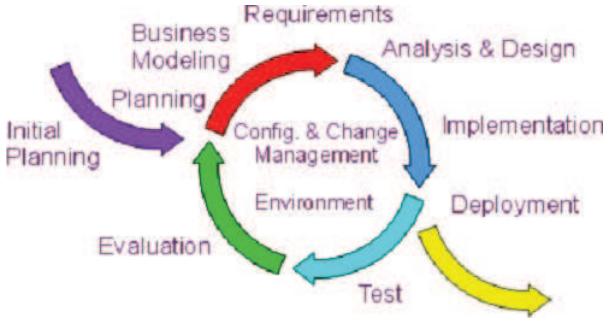
Rational Unified Process (RUP)

- produkt firmy Rational + dalších velkých firem
- podrobný popis najdete na **http://www.kiv.zcu.cz/studies/publikace/RationalUnifiedProcess** - podrobně pak v ASWI
- 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 29) – 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 architektura sw 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



Obrázek 29 Rational Unified Process – přehled fází

- každá fáze rozložena do iterací (výsledek iterace: spustitelná verze sw produktu, která je podmnožinou cílového sw produktu)
- iterace vývoje sw produktu (Obrázek 30)



Obrázek 30 Iterace vývoje softwarového produktu

- toky aktivit (činností) – tzv. workflow – reprezentují posloupnosti aktivit, které vytvářejí požadované produkty (zahrnují orto-, paraprocesy i metaprocesy)
    - základní toky aktivit – jejich výstupem jsou modely na dané úrovni abstrakce
        - byznys modelování – odpovídá zjištění podnikatelských požadavků a pravidel Kapitola 2.
        - Specifikace požadavků – odpovídá zejména uživatelským a funkčním požadavkům viz Kapitola 2
        - Analýza a návrh – architektura + podrobnější návrh systému
        - Implementace – kódování + jednotkové testy, případně integrační testování
        - Testování – ověřuje, jestli je sw produkt správný (proti specifikaci)
        - Nasazení – konfigurace výsledného sw produktu v cílovém prostředí
    - podpůrné toky aktivit
        - Konfigurační management a řízení změn – správa verzí všech vytvářených meziproduktů, často také nazývaných artefaktů (kódu, dokumentací, testů,…)
        - Projektové řízení – plánování pořadí a časové náročnosti aktivit, přidělení zdrojů a jejich vytíženosti, dodržení rozpočtu, kontrola výsledků, řízení rizik,…
        - Prostředí – zajištění zdrojů (tvrdých i měkkých), budování pyramidy vitality a pyramidy kultury

Agilní metodiky

- metodiky odpovídající inkrementálnímu modelu
- http://www.agilealliance.org
- Podrobněji KIV/ASWI

- Manifest agilního programování
- Umožnit změnu je mnohem efektivnější, než se jí snažit zabránit
- Je třeba být připraven reagovat na nepředvídatelné události, protože ty nepochybně nastanou
- Aneb:
    - 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

Agilní přístup

- společnými rysy všech agilních metodik jsou
    - Iterativní a inkrementální vývoj s krátkými iteracemi
    - Komunikace mezi zákazníkem a vývojovým týmem
    - Průběžné automatizované testování

Konkrétní metodiky

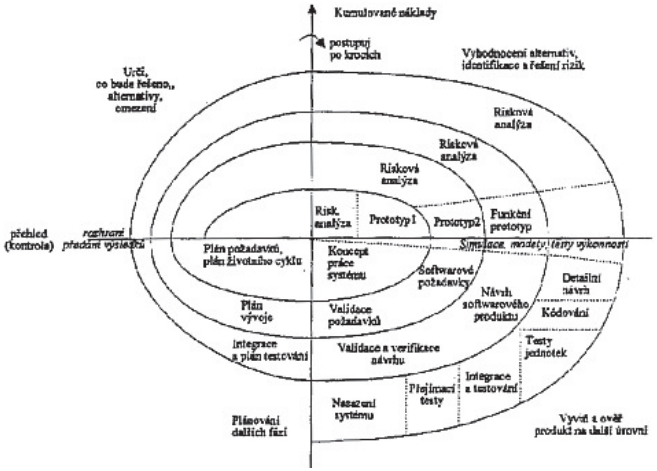
- Extreme Programming (XP) - Kent Beck, Ward Cunningham, Ron Jeffries (www.xprogramming.com)
- Feature-Driven Development (FDD) - Jeff de Luca, Peter Coad (www.featuredrivendevelopment.com)
- SCRUM Development Process - Ken Schwaber, Mike Beedle (www.controlchaos.com)
- Test-Driven Development (TDD) - Kent Beck (www.testdriven.com)

Extrémní programování (Beck 1999)

- rozporuplné názory
- krátký průzkum, případy užití
- rychlá zpětná vazba
- předpoklad jednoduchosti
- vývoj malých inkrementů
- zatažení zákazníka do procesu
- průběžné vylepšování kódu
- pro případy, kdy požadavky na sw jsou vágní nebo se rychle mění
- použitelná pro malé týmy (2-10 lidí)
- veškerý kód psán povinně ve dvojicích (jeden kóduje, druhý kontroluje)

Spirálový model SW procesu

- spirálový model SW procesu původně navrhl Boehm (1988)
- realistický přístup k vývoji rozsáhlých SW systémů (není vhodný pro malé projekty), orientován na rizika, místo sekvence (kde se někdy musíme vracet) je proces
- reprezentován spirálou, v každé otočce vytváříme úplnější verzi SW
- SW projekt rozděluje do miniprojektů
    - každý miniprojekt řeší jedno nebo více rizik, dokud nejsou všechna hlavní rizika vyřešena
    - pojem "riziko" je definován poměrně široce, může znamenat špatně pochopené požadavky, nesrozumitelnou architekturu, potenciální problémy s výkonností, potíže s technologií apod.
    - po vyřešení rizik pokračujeme v miniprojektu, např. podle vodopádového modelu



Obrázek 31 Spirálový model vývoje sw produktu

*

- model začíná ve středu spirály, prozkoumáme rizika a pak zvolíme způsob řešení další iterace
- každá otočka spirály je rozdělena do 4 sektorů (sektory nemusejí trvat stejnou dobu)
    - určení záměrů pro současnou fázi projektu (= otočku spirály)
        - definujeme záměry dané fáze projektu (funkčnost, výkonnost apod.)
        - určíme alternativní možnosti realizace záměrů (např. design A, design B, použití existující komponenty, nákup komponent)
        - identifikujeme omezení daných alternativ (cena, časový plán, přenositelnost)
        - vyhodnotíme alternativy vzhledem k záměrům a omezením - často se tím ukáže nějaká nejistota, která je zdrojem rizik
        - riziko = něco, co se může nepovést
        - pokud vyplynula rizika, měli bychom se jimi zabývat v další fázi
    - vyhodnocení a snížení rizik
        - pro každé z identifikovaných rizik je provedena podrobná analýza a kroky ke snížení rizika
        - prototypování, simulace, vypracování dotazníků apod. (např. riziko, že jsou nesprávně definovány požadavky, můžeme vytvořit prototyp části systému a konzultovat ho se zákazníkem)
    - vývoj a validace
        - provádíme vývoj a ověření produktu na dané úrovně
        - použitý model SW procesu je určen zbývajícími riziky (riziko integrace podsystémů -> vodopádový model, riziko nevhodné uživatelské rozhraní -> evoluční prototypování)
    - plánování
        - každá otočka je zakončena posouzením všech produktů vytvořených v předchozím cyklu a plánů pro další cyklus
        - naplánujeme další fázi projektu (výstupy, požadavky na zdroje, organizace zdrojů, rozdělení práce, časový plán)
        - cílem je, aby byly všechny zúčastněné strany srozuměny s plány na další fázi

- obrázek - příklad, jak může vypadat vývoj podle spirálového modelu:
    - nejvnitřnější otočka spirály - všeobecný záměr, studie proveditelnosti
    - druhá - definice požadavků
    - třetí - návrh (design) systému
    - čtvrtá - vytvoření všech programů
    - uvedený diagram je pouze příklad, konkrétní počet iterací nebo kroky v iteracích se mohou lišit
- **spirálový model je chápán jako "generátor modelů SW procesu"**, s jeho pomocí vytváříme podrobnější model SW procesu (tj. konkrétní obsah jednotlivých otoček může být odlišný)

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

WinWin spirálový model

- původní spirálový model neposkytuje podrobnosti o tom, co se má dělat v prvním kvadrantu (neříká, odkud se vezmou "záměry pro současnou fázi projektu") -> modifikován (Boehm a Bose, 1994)
- nový model je postaven na tzv. win-win přístupu: snaha, aby všechny zúčastněné strany (zákazníci, vývojáři, vedoucí atd.) byly "výherci" – tj. myslíme na potřeby zúčastněných
    - zákazník: mít produkt co nejdříve, zaplatit za něj co nejméně
    - vývojáři: odborný růst, pracovní a platové podmínky, preference pro používané nástroje (chci to psát v Javě), odložit psaní dokumentace apod.
    - nadřízení: zisk, nepřekročení rozpočtu, žádná překvapení

Postup

1. identifikace klíčových účastníků projektu - do úvah zahrneme všechny účastníky projektu důležité pro daný cyklus projektu
2. pochopení zájmů každého účastníka (viz výše v rámci specifikace, obvykle nejobtížnější část)
3. urovnání podmínek pro výhru, řešení konfliktů - např. zákazníci většinou neodhadnou, co je snadné nebo obtížné naprogramovat => mají nerealistická očekávání - nabídnout novou alternativu typu výhra-výhra

- **ostatní části jsou stejné jako u původního spirálového modelu**

Vývoj orientovaný na znovupoužitelné komponenty

- vývojáři znají kód nebo návrh podobný tomu, který se požaduje, ten vezmou, upraví a začlení do svého systému - děje se nezávisle na modelu vývoje SW
- spoléhá na existenci velké množiny SW komponent a rámce pro jejich integraci
- komponentou např. knihovní fce, třída, podsystém, aplikace
- rámcem např. příkazové jazyky (Tcl/tk), distribuované objektové architektury (CORBA, JavaBeans) apod.

Komponentově orientovaný proces vývoje zahrnuje:

- specifikaci požadavků
- **analýzu komponent**
- **modifikaci požadavků**
- návrh systému s využitím komponent
- vývoj a integraci
- validaci systému

Analýza komponent

- 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

Návrh systému s využitím komponent

- během této fáze je navržen rámec systému nebo se využije už existující rámec
- pokud některá potřebná komponenta neexistuje, musíme jí navrhnout

Vývoj a integrace

- vytvoříme komponenty, které nemáme (a nemůžeme koupit atd.)
- komponenty integrujeme do systému

Výhody a klady modelu

- omezené množství SW, který musíme sami vyvinout => snížení ceny produktu a menší nebezpečí, že nastanou potíže
- obvykle rychleji získáme výsledný SW

Nevýhody modelu

- kompromisy vůči požadavkům jsou nevyhnutelné => systém nemusí splňovat požadavky zákazníka
- vytvářené části musíme přizpůsobit možnostem komponent, což se projeví na kvalitě jejich návrhu (jejich flexibilitě apod.)
- vývoj není zcela v našich rukách, protože nové verze komponent vyvíjí někdo jiný (vývoj komponenty může být ukončen, nové verze nemusejí být kompatibilní se starými verzemi) => údržba systému se může prodražit
- rámce a knihovny komponent bývají tak rozsáhlé, že seznámení s nimi může vyžadovat několik měsíců (ve velkých organizacích mají pak vyčleněné konkrétní lidi)

Shrnutí

- 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")
- formální metody - pokud musím dokazatelně splnit specifikaci, mám podpůrné nástroje a tým seznámený s formálními metodami
- 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)
- win-win spirálový model - jako spirálový, vhodnější pro projekty na zakázku

Hlavním účelem modelů je určit správné pořadí kroků při vývoji SW a určit kritéria pro přechod k dalšímu kroku.

Řízení projektů

Co je to 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

Příklady IT projektů (zapojen hardware, software a/nebo sítě)

- Vaše týmová celosemestrální práce
- Upgrade hardware a software v laboratořích KIV
- Rozšíření technické infrastruktury univerzity o přístup k bezdrátovému internetu
- Tým (často dobrovolníků) vyvíjí standard pro určitou technologii
- Tým v podniku rozhoduje o tom, který software bude zakoupen a jak bude nasazen
- Firma vyvíjí webový e-shop pro podporu prodeje vlastních produktů

Vlastnosti projektu

- Má jednoznačně určený cíl – zdokumentovaný ve specifikaci požadavků
- Je dočasný – pevně stanovený začátek a konec
- Řešení projektu se postupně vypracovává
    - začátek – široké definování problémů (viz prvotní zadání vašich zadavatelů)
    - postupem času vyjasňování detailů
    - prvotní plány, nutnost aktualizace, řízení změn
- vyžaduje určité, často různorodé zdroje
    - tvrdé i měkké (hardware, software, lidé)
- má hlavního zákazníka nebo zadavatele (investora)
    - zadavatel určuje směr řešení projektu a uvolňuje zdroje (finance, zápočty)
- součástí projektu je neurčitost
    - obtížné definování cílů projektu, odhad doby potřebné k dokončení, náklady
    - vnější faktory (nemoc, krach dodavatelské firmy)

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)

Řízení projektů

- je „uplatnění veškerých poznatků, dovedností nástrojů a technik na aktivity (činnosti) projektu takovým způsobem, aby byly splněny požadavky na projekt“ (PMBOK® Guide 2004,strana 8)
- klíčové prvky – účastníci projektu (stakeholders), poznatky pro řízení projektů, nástroje a techniky pro řízení projektů, přínosy úspěšných projektů pro organizaci

Účastníci projektu

- do projektu zapojeni nebo jsou projektem ovlivněni: zadavatel (investor) projektu, vedoucí projektu a projektový tým, podpůrný personál, dodavatelé, oponenti,…viz kapitola 1

Poznatky pro řízení projektů

- základní oblasti
    - řízení rozsahu projektu – definování a řízení veškerých prací
    - řízení času v projektu – odhad doby potřebné k dokončení prací, návrh přijatelného časového plánu, zajištění včasného dokončení
    - řízení nákladů – rozpočet a jeho řízení
    - řízení kvality – projekt splní stanovené cíle nebo předpokládané potřeby v definované kvalitě
- podpůrné oblasti
    - řízení lidských zdrojů – efektivní nasazení lidí do řešeného projektu
    - řízení komunikací – generování, shromažďován, distribuce, ukládání informací projektu
    - řízení rizik – identifikace, analýza, reakce na rizika
    - řízení obstarávání – obstarávání zboží a služeb z vnějších organizací
- řízení integrace projektu – souvislost (vzájemné ovlivnění) se všemi předchozími

Nástroje a techniky pro řízení projektů

- pomáhají při práci ve všech devíti oblastech řízení projektu
- Ganttovy diagramy, síťové grafy, analýza kritické cesty (řízení času)
- Struktura rozpisu prací (WBS), analýza požadavků, řízení změn rozsahu (řízení rozsahu)
- Analýza doby návratnosti, odhady nákladů (řízení nákladů)
- Paretovy diagramy, modely vyspělosti (řízení kvality)
- Motivační techniky, týmová spolupráce (řízení lidských zdrojů)
- Řešení konfliktů, komunikační infrastruktura (řízení komunikací)
- Smlouvy, vyjednávání, analýza „make“ or „buy“ (řízení obstarávání)
- Hodnocení rizik, plán řízení rizik (řízení rizik)
- Metodologie řízení projektů, plán řízení projektů, software pro řízení projektu (řízení integrace)

Co je to úspěšný projekt?

- definuje manažer projektu spolu s klíčovými účastníky
- Studie společnosti Standish Group (zlepšení statistických ukazatelů pro IT projekty)
    - Podíl úspěšných projektů v IT se zvýšil z 16% (1994) na 34% (2002)
    - podíl neúspěšných projektů v IT se snížil z 31% (1994) na 15% (2002)
- V USA vynaloženo na IT projekty 250 miliard dolarů (1994), 255 miliard dolarů (2002), promrhané peníze se snížily ze 140 miliard USD (1994) na 55 miliard USD (2002)

Projektový manažer a jeho role

- Obecně a ideálně zvládnutí pyramidy vitality a pyramidy kultury
- Prakticky závidí na povaze konkrétního projektu
- Podle PMBOK® Guide 2004 by řídící tým projektu měl disponovat
    - Základními okruhy poznatků pro řízení projektů (viz výše)
    - Poznatky, standardy a předpisy pro příslušnou oblast aplikací
    - Poznatky ohledně prostředí projektu
    - Obecnými poznatky a dovednostmi řízení
    - Sociálními dovednostmi (soft-skills)
- v IT nemusí (a reálně ani nemůže) projektový manažer do detailu rozumět příslušným používaným technologiím (nemusí být expert), ale musí vědět tolik, aby měl věci pod kontrolou (IT background)
- Dostatečné odborné znalosti projektového manažera jsou důležité i proto, aby ho tým přijal a respektoval
- Dobrý organizační manažer může být dobrým manažerem IT projektů, protože se dokáže zaměřit na splnění potřeby organizace, technické detaily pak deleguje. Pro úspěšné dokončení projektu je prospěšnější být lepším projektovým manažerem než-li expertem na konkrétní informační technologie
- Význam vůdčích schopností – schopnost vybudovat tým, umění komunikovat, sebeúcta, sebevědomí, zaměření se na výsledky,…

Project Management Institute (PMI)

- Profesionální sdružení projektových manažerů
- www.pmi.org
- Zájmová skupina Student of Project Management SIG (www.studentsofpm.org)
- PMP certifikace (Project Management Professional)

Software pro podporu řízení projektů

- Stovky produktů
    - Nižší třída – základní funkce, do 200 USD na uživatele
- Střední třída – rozsáhlejší projekty, více uživatelů nad více projekty, 200 až 500 USD na uživatele
    - Vyšší třída – pro řízení podnikových projektů
- Microsoft Project (nástroj střední třídy)

Systémový pohled na řízení projektů

- Řešení projektů probíhá vždy v širším prostředí než je projekt samotný – přemýšlení o projektu v kontextu celé organizace
- Tři sféry řízení systému: obchodní, organizační (zákaznická), technologická - zaměření se pouze na jednu oblast vede minimálně k problémům s integrací projektu
- Obchodní pohled (učí se studenti ekonomických směrů) – typické otázky
    - Jaké potřeby a přání mají naši zákazníci?
    - Jaké výrobky či služby jejich přání uspokojí?
    - Proč si naši zákazníci koupí právě toto…?
    - Jakou máme konkurenci?
    - Kolik nás to bude stát?
    - Kolik bude potřeba lidí, jak je budeme platit?
    - Má smysl tuto technologii kupovat?
    - Není lepší koupit krabicový software, nežli vyvíjet vlastní?
- Technologický pohled (učí se studenti technických fakult) – typické otázky
    - Jaký a jak výkonný hardware potřebujeme?
    - Jaké jsou požadavky na výkonnost a spolehlivost celého systému?
    - Na jakých operačních systémech poběží aplikace?
    - Jaké technologie a jakou architekturu použijeme?
    - Jak ověříme, že systém má fungovat tak, jak má?
    - Jakým způsobem budeme provádět testy?
- Organizační/zákaznický pohled (učí život)
    - Jak daná organizace zákazníka funguje, jak tam lidé pracují (ne, jak si myslíme, že pracují nebo jak nám říkají, že tam pracují)
    - Dotkne se projekt všech zaměstnanců zákaznické organizace a v jakém smyslu?
    - Jak zákaznické organizaci zpříjemnit, zefektivnit práci?
    - Jaká je kultura zákazníkovy organizace?
- Do přípravy projekty zahrneme všechny tři složky – celý kontext projektu – hledáme nejlepší řešení

Zvláštnosti projektů v IT

- Různorodá povaha projektů
- Různorodé věcné oblasti
- Rychlá změna technologií
- Obtížně sestavitelný tým - rozdílné specializace jeho členů

Pro naše účely budeme potřebovat stanovení rozsahu projektu – rozsah projektu bude definován v dokumentu specifikace požadavků, strukturu rozpisu prací (WBS) a Ganttův diagram

Literatura:

- Schwalbe K., Řízení projektů v IT, Computer Press, 2007.
- Kadlec V., Metodiky efektivního vývoje softwaru, Computer Press, 2004
- Vondrák I., Úvod do softwarového inženýrství, VŠB TU Ostrava, http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf
- Zendulka I., Projektování programových systémů - 2 Životní cyklus programového díla, http://www.fit.vutbr.cz/study/courses/PPS/public/pdf/2_Ziv_cykl.pdf
- Project management Institute: www.pmi.org
- Agile alliance: http://www.agilealliance.org
- Rational Unified Process: **http://www.kiv.zcu.cz/studies/publikace/RationalUnifiedProcess**

4. 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
- Zde jen lehký úvod, podrobně v ASWI
- 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, zaznamenávání 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)

Prvek konfigurace

- Základní konstituující jednotka systému
- Nejsou to jen spustitelné či zdrojové soubory, ale i všechno „okolo“ – knihovny, skripty, dokumenty, modely, testovací data, verze překladače apod.
- Víme o jeho existenci, vlastníkovi, změnách, umístění v produktu (musíme se rozhodnout, na jaké úrovni abstrakce se pohybujeme – soubory, metody,…)
- Atomický z hlediska identifikace a změn
- Jednoznačně identifikovatelný (není často jednoduché to dobře vymyslet), např. APQ/D22/PRSPACE/SA/12.1
    - Zkratka firmy (+pobočka)
    - Označení projektu (+fáze projektu)
    - Typ prvku
    - Název prvku
    - Identifikátor verze

Softwarová konfigurace

- Souhrn prvků, který reprezentuje podobu daného sw systému
- Často jednoznačně identifikovatelná
- Musí obsahovat všechno, co je třeba k opakovanému vytvoření příslušné verze sw produktu (zdrojové soubory, dokumentace, testovací data ad.)
- Konfigurace, kterou lze použít (zdrojové soubory lze přeložit, funguje na daných testovacích datech…) = konzistentní konfigurace
- Popis konfigurace (např. v UML – diagram komponent, prvky a jejich vztahy)

Správa verzí

- Nástroj identifikace prvků i celé konfigurace
- Udržení přehledu o podobách prvků konfigurace
- Historická podoba verze = revize (Visual Studio 2005)
- Alternativní podoba verze = varianta (Visual Studio Team System)

Verzování

- podle stavu (identifikace prvků)
- podle změn (identifikace změny – výsledná verze prvku vznikne aplikací změn)
- celé konfigurace (SVN) – odvozené verze prvků
- jednotlivých prvků (CVS) – konfigurace nemá verzi
- produktu

Popis verze

- Extenzionální – každá verze má jednoznačné ID
    - jednodušší, nevhodné pro velký prostor verzí
    - např. verze 12.1
    - jedinečný identifikátor verze+metadata
- Intenzionální – každá verze popsána sadou atributů
    - vhodné pro velký prostor verzí, s touto podobou umí pracovat málo nástrojů
    - např. os = windows xp, jvm = 1.5.x

Úložiště (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

Pracovní prostor

- soukromý prostor pro změny neovlivňující podobu prvků konfigurace v repository

Přístup k úložišti

- Pesimistický (read-write kopie jen pro jednoho) vs. optimistický (read-write kopie pro všechny, pak řešení konfliktů) pohled na zamykání při operacích check in , check out

Vývoj aplikace

- Základní postup při vývoji: inicializace, check out, lokální testování (do úložiště se neukládá prvek, který např. nejde zkompilovat), check-in, integrační testování

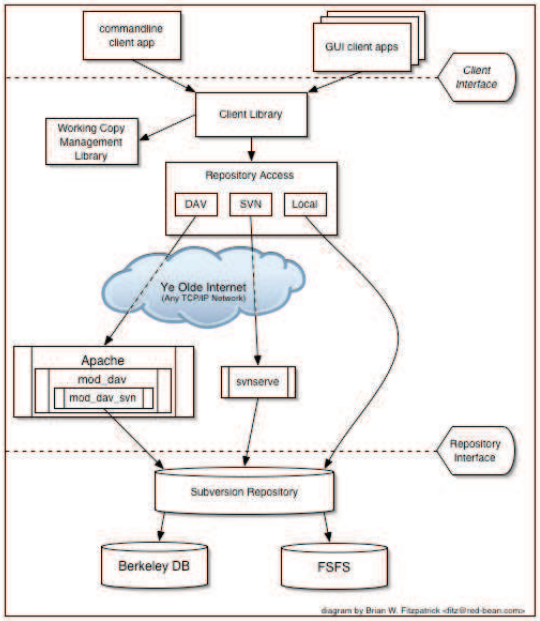
Některé systémy pro správu verzí

- SVN (Subversion)
- CVS (Concurerent versioning system)
- Rational ClearCase
- Microsoft Visual SourceSafe

Best practices (Subversion)

- Používejte v repository standardní adresářovou strukturu; kořenový adresář (jméno projektu) obsahuje podadresáře
    - /trunk – hlavní vývojová větev produktu
    - /branches – vedlejší či alternativní vývojové větve, na kterých se pracuje
    - /tags – ukončené vývojové větve (např. release)
- Do repository ukládejte změny, které odpovídají jednomu jednoduchému účelu (např. přidání vlastnosti, metody, oprava chyby); komentujte provedenou změnu
- Používejte systém pro správu požadavků a slaďte jej s použitím repository (např. uložení změnu do repository je reakcí na požadavek ID požadavku se např. objeví v komentáři změny); některé systémy vzájemnou provázanost repository a správy požadavků přímo vyžadují
- Při slučování (merge) důsledně pište podrobné komentáře
- Soubory a adresáře ve vašem pracovním prostoru mohou být obecně libovolné (tedy i rozdílné) revize z repository; to musíte brát v úvahu při ukládání do repository
- Neexistuje limit na velikost souborů, které můžete do repository uložit, přenos souboru mezi lokálním pracovním prostorem a repository je „streamovaný“, využívá konstantně malou velikost paměti

- Přenos velkých soborů trvá dlouho
- Existují různé přístupy ohledně ukládání kódů do adresářů /trunk a /branches



Obrázek 32 Subversion

Literatura a odkazy:

- Sussman B.C. et al., Version Control with Subversion, http://svnbook.red-bean.com/
- SVN, http://subversion.tigris.org/
- CVS, http://ximbiot.com/cvs/wiki/
- Microsoft Visual SourceSafe, http://msdn.microsoft.com/en-us/vstudio/aa700907.aspx
- IBM Rational ClearCase, http://www-01.ibm.com/software/awdtools/clearcase/support/index.html
- Origo, správa projektu, správa verzí, http://www.origo.ethz.ch/

5. Jazyk 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

Tři stavební bloky

- Předměty – elementy modelu
  - Strukturní abstrakce – podstatná jména – třídy, rozhraní, spolupráce, případy užití
  - Chování – "slovesa" – interakce, stav
  - Seskupení – balíčky pro seskupení sémanticky souvisejících prvků
  - Poznámky – anotace typu "zvýrazňovač"
- Relace – propojují jednotlivé předměty
  - Asociace – popis spojení mezi objekty
  - Závislost – změna v určitém předmětu ovlivňuje jiný předmět
  - Zobecnění (generalizace) – jeden element je specializací jiného elementu
  - Realizace – jeden klasifikátor určuje dohodu, jejíž realizaci zaručuje jiný klasifikátor
- Diagramy – znázorňují pohledy na model (model vs. diagram)
  - Diagram tříd
  - Diagram komponent
  - Diagram nasazení
  - Objektový diagram
  - Digram případu užití
  - Diagram posloupnosti (Sekvenční diagram)
  - Diagram spolupráce
  - Stavový diagram
  - Diagram aktivit

Čtyři obecné mechanismy

- Specifikace – textový popis sémantiky jednotlivých elementů modelu (jádro modelu, sémantický základ modelu) – udržuje model pohromadě a dává mu smysl
- Ozdoby – rozšíření informací o elementu modelu – při správném využití zvyšují čitelnost elementu modelu
- Podskupiny – popis různých pohledů na svět
  - Klasifikátor a instance – klasifikátor je abstraktní vyjádření typu předmětu (např. pivo), instance je specifický výskyt typu předmětu (např. moje pivo); příklady klasifikátorů – účastník (vnější uživatel systému), třída, komponenta, datový typ, rozhraní (kolekce operací k určení poskytované služby), uzel (fyzický element obsahující spustitelný kód, který zastupuje výpočetní prostředek), signál, případ užití, subsystém
  - Rozhraní a implementace – rozhraní je dohoda, která specifikuje chování předmětu, implementace specifikuje podrobnosti tohoto chování