Tým. K čemu slouží? Výhody a nevýhody.

- Jednotlivec nestačí v daném termínu udělat svou práci, tak se o ni podělí více lidí.
    - Synergie (vzájemné pozitivní ovlivňování) a motivace spoluprací.
    - Předávání znalostí a dovedností, menší nebezpečí ztráty znalostí a dovedností.
    - Dělba práce a rolí → lepší využití talentů (superprogramátor se nebabrá s prkotinami a věcmi, které neumí, využití i méně kvalifikované síly), resp. kombinace talentů.
    - Menší závislost na jednotlivcích (s odchodem pracovníka neodejde znalost, vydírání).

- Špičkových lidí je málo. Týmová práce umožňuje lépe využít jejich kapacitu. Vývoj SW je zvláště náročný na špičkové pracovníky.
- Přítomnost talentů je výhodou pro všechny. Tým umožní využití talentů různých typů.
- Tým umožňuje zrychlení získávání dovedností a znalostí. To je zvláště důležité pro špičkové profesionály.
- Bez špičkových profesionálů nelze v globálním světě obstát, to by měl zohlednit podnik i stát. To se týká i školské politiky včetně politiky zásad školného.
- Talenty jsou omezený přírodní zdroj.

Nevýhody týmu:
- Přibývají náklady na budování a údržbu týmu a jsou potřeba specifické znalosti a dovednosti. Ve velkých týmech dramaticky klesá produktivita.
- (Zbytečná) administrativa.
- Jsou třeba specifické dovednosti a práce nutné pro budování a údržbu týmu.
- Mnoho špičkových programátorů se těžko podřizuje týmové práci.
- Problém vymezení a přijímání rolí a volby jejich nositelů.

Typy talentů (aspeak), které je nutno podporovat.

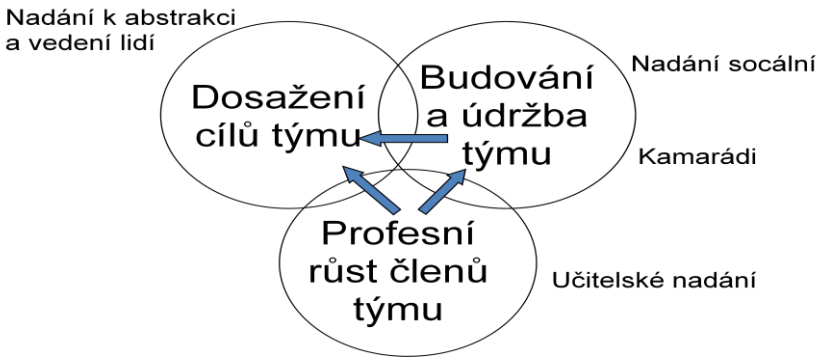
- Talent a lidé všeobecně je důležitý zdroj bohatství a rozvoje společnosti, je to přírodní **omezený,** ale zčásti obnovitelný zdroj.
- Vzdělávací soustava by ho měla zhodnocovat a také zajistit, aby se vůbec talenty zachytily a rozvíjely.

Druhy talentů (inteligencí):
- MI – multiple intelligence, různé dimense
    - **A**bstraktní (matematická apod.), měřeno IQ.
    - **S**ociální (práce s lidmi, např. řídit tým).
    - **P**raktická (něco vyrobit, zručnost), manuální.
        - Důležitá u výrobních týmů, trochu se kryje se schopností kódovat, ta je korelována s abstrakcí.
    - **E**mocionální (sebeuvědomění/ovládání, empatie).
    - Výtvarná, umělecká (**a**estetic), důležitá pro UI.
    - **K**inestetic (pohybová, sportovní).
    - ASPEAK – zkratka všech.

- Pro IT práce, především pro programování a specifikace hlavně ABSTRAKTNÍ.
- Pro vedení SOCIÁLNÍ.
- Pro budování týmu a vedení jednání EMOCIONÁLNÍ.
- Pro některé práce UMĚLECKÁ (návrh obrazovek).

Jaké jsou tři oblasti činnosti v týmu?

1. Tým se buduje s nějakým (pracovním) cílem.
2. Aby tým mohl existovat, je třeba se o jeho existenci starat (stanovení rolí, komunikace a vztahy mezi členy, společenská podpora): Tým musí být nutný pro dosažení cílů.
3. Aby tým mohl dosahovat požadované cíle, je nutno se starat o odborný růst (i dočasných) členů týmu. To zároveň zlepšuje vnitrotýmové vztahy takže to přispívá k plnění bodu 2. I zde platí zásada vítěz-vítěz, nikdo na nikoho nedoplácí.



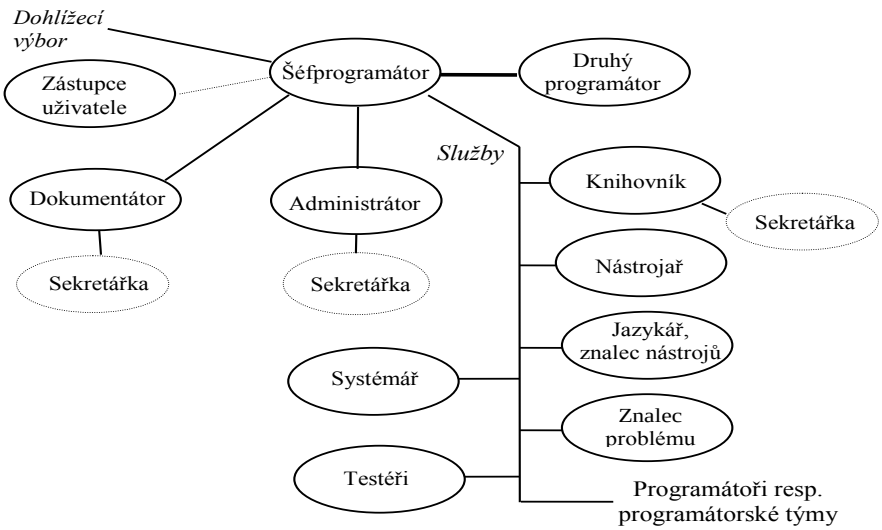
Demokratický tým – jak se v něm pracuje a nač se hodí?

- Pracuje na principu ad-hoc-kracie, role se explicitně dohodnou na základě dobrovolnosti (na jistou, mimo agilní vývoj na poměrně dlouhou dobu).
- Pro kvalitní pracovníky nové problémy a menší až střední úkoly nejefektivnější organizace (VÚMS).
- V malých firmách velmi často používán.
- U velkých firem management nerad vidí
  - závislost na několika lidech,
  - nestandardní řešení, méně používány normy,
  - obavy, jak se uskuteční revitalizace (úplné přepsání).
- Problém zajištění služeb, mnoho věcí si dělají všichni sami.
- Mimo agilní formy je to vhodné spíše pro stálý tým, hrozí ale zkostnatění.
  - Vhodný kompromis: agilní formy vývoje (role jen na krátkou dobu).
- Vhodné je volit role podle principů známých z jiných forem týmů.
- **Při kvalitních lidech velmi efektivní.**
- **Spíše skoro stálý tým, jádro týmu je velké a stálé pro různé projekty (existují i kompromisy).**

- Typické pro menší firmy a iterativní vývoj.
- Vhodné pro významně nové typy úkolů a produkty, které nejsou příliš kritické.
- Úspěšné pro kvalitnější pracovníky.
- V ČR existují příklady velmi kvalitních týmů.
- Je běžný při agilním vývoji.
- U déle existujících týmů sklon k týmovému šovinizmu a konzervativizmu.

Tým šéfprogramátora.

- Je vhodný v situaci, kdy je k dispozici několik vynikajících odborníků a pak poměrně mnoho relativně nezkušených pracovníků.



- Při vhodných lidech fantastická výkonnost.
- Problém přijetí rolí.
- Vhodné spíše pro středně velké *nekritické* projekty.
- Nevýhodou je silná závislost na šéfprogramátorovi, jeho talent se dá ale plně využít.
- Nevhodné pro agilní formy vývoje.
- Použitelné spíše výjimečně.
- *Historický přínos – vymezení služeb, prokázání podpora superprogramátora*.

Pevný a najímaný tým. Na které projekty se hodí.

Pevný tým:
- Úkoly se hledají k týmu.
- Víceméně nutnost u malých firem.

Najímaný tým:
- Vedení týmu se jmenuje pro daný projekt, ostatní se z pracovníků firmy „najímají“ podle potřeby (nejde u malých firem).
- Vhodné pro spíše rutinní práce, pak použitelné i pro velké systémy.
    - I proto velcí výrobci nepřichází s razantními inovacemi, raději je koupí.

Kdy najímaný tým?
- Velký spíše rutinní úkol (podmínka pro smysluplnost najímání).
- Větší firma (je odkud najímat, efekt velkých čísel – je velká šance, že se vždy někdo najde vhodný pro danou práci, kombinace talentů).

Vlastnosti dobrého vedoucího. Typy manažerů, potřebné vlastnosti.

- Vliv vedoucího je zásadní.
    - Buduje tým, najímá členy týmu (de facto).
    - Obsazuje role (často neformálně a přesto jasně).
    - Zabezpečuje všechny tři typy činností v týmu.
- Působení vedoucího je založeno na:
    - odborné a manažerské kvalitě, výběr lidí,
    - výkonnosti,
    - charismatu a sociálním talentu,
    - vzájemné důvěře (dovede držet slovo, podrží při problémech, dovede tým hájit, ale také motivovat,…).
- U větších týmů nemusí být vedoucí nutně odborně nejlepší (řízení vyžaduje specifické autonomní dovednosti – viz činnost dvojic).

Vlastnosti vedoucího:
- Kompetentnost odborná i organizační.
- Vědomí vlastních předností a nedostatků (v čem nejsem dobrý, na to někoho najdu).
- Zdravá sebedůvěra až tvrdohlavost, ne bezhlavá a zároveň schopnost naslouchat a přebírat nápady, inovovat.
- Formulace cílů i způsobů řešení.
- Najít správné lidi, motivovat je, optimálně zatěžovat a kontrolovat.
- Plánování a dělba úkolů.
- Být příkladem pracovním i morálním.
- Vychovávat nástupce.
- Předvídavost, odhad rizik.
- Loajalita k podniku i k týmu (vyrovnaná).
- Schopnost správně oceňovat a aplikovat nové poznatky, podporovat rozumnou míru inovací.
- Dovést uznat a napravit svoje chyby.
- Diplomatické schopnosti.
- Psychologicky silná osobnost.

Manažerské typy:
- **Charismatický –** rychle se nadchne, ale nevydrží (25% případů).
  - Při spolupráci dbát o to, aby neztratil zájem (tlačit).
- **Hloubavý** – je nutné mít připraveny eventuality, varianty a data (11%).
  - Pokud ho ukecáme, bude držet slovo.
- **Skeptik** – nutno vše doložit a vyčíslit, dá na rady lidí z blízkého okolí (19%).
  - Často se bez nich nerozhodne, tendence přeceňovat detaily.
- **Následovník** – bere jen to, co se někde někomu už osvědčilo (36%).
  - Snaží se mít alibi.
- **Kontrolující** – má tendenci opakovaně silně prověřovat detaily (5%).

Vedoucí de-facto, de-jure. Jak se řeší, když v jednom týmu je víc de-facto vedoucích?

- Vedoucí de facto – všichni jeho odborné manažerské a osobní kvality uznávají a jeho doporučení jsou přijímána.
- Vedoucí de jure – je jmenován. Vedoucí de jure by měl být i vedoucím de facto.

- Je velký problém, je-li více vedoucích de facto (různí lidé uznávají kvality různých členů týmu), než když se nekryje vedoucí de jure a de facto. Je lepší takový tým rozdělit.

Spolupráce ve dvojici. Výhody a nevýhody

- Práce ve dvojici je velmi efektivní varianta spolupráce (minitým).
  - Spíše partneři než nadřízený a podřízený.
    - Druhý funguje i jako zástupce a případně řeší specifické úkoly.
  - U agilních týmů základní typ spolupráce.
  - Role se mohou vyměnit.
  - Mnohé výhody (sdílení znalostí a dovedností, snížení nebezpečí monopolu znalostí, organizační jednoduchost).
- Samostatná dvojice (čtení kódu).
- Vedoucí týmu a jeho zástupce!
- Nutné u větších týmů.

- Vedoucí ve dvojici „vše“ píše a o všem rozhoduje.
- Zástupce (alter ego):
  - Je u všech rozhodnutí.
  - Dělá permanentní oponenturu.
  - Někdy samostatně píše pomocné dokumenty.
  - Je partnerem, vztah podřízenosti potlačen.
  - Ve větším týmu se často stará o chod týmu.
- Role se mohou někdy prohodit.

Loajalita v týmu, jak se projevuje, jak se delegují pravomoci?

- Identifikace s cíli a postupy týmu.
- Ochota pro tým něco udělat a případně za tým i bojovat (např. s šéfy).
- Přátelství s členy týmu případně hrdost na dosažené výsledky.
- Záporné aspekty:
  - ponorková nemoc,
  - týmový šovinismus (nekritická obhajoba zájmů týmů a jeho činnosti),
  - nevhodné přístupy (dominance agresorů, hraný zájem o názory,…).

Delegace pravomocí:
- Sníží závislost na přítomnosti vedoucího.
- Snižuje zátěž vedoucího.
- Zlepšuje klima v týmu, zlepšuje obvykle komunikaci v týmu.
- Výchova nástupce.
- Partnerství.
- *Náročné přijmout, ohrožuje postavení vedoucího (spíše toho horšího), něco nelze delegovat (např. odpovědnost za úkol).*

Neformální role v týmu (co to je, příklady pozitivních a negativních), psychohry, jak na ně reagovat?

Psychohry:
- Podvědomá schémata jednání (často v ad hoc skupině) narušující práci týmu.
  - Nesměšovat s psychohrami jako prostředku léčby neuróz až psychóz.
  - Nutno včas zarazit.
  - Teoretici týmové práce vymezili skoro 40 psychoher.

Neformální role:
- Člen týmu by měl splňovat řadu podmínek. Při hodnocení členů týmu je vhodné hodnotit pracovní schopnosti a nadání, ale také zlozvyky.
- V psychologii práce se rozeznávají následující pracovní role:

1. **Využitelné role:**
   - Iniciátor čili rozjížděč (intuice na to, co je vhodné dělat, formulace vizí, nerad dotahuje detaily).
   - Hledač informací, inovátor (použít novinky, informace o novinkách, vhodný jako oponent).
   - Hodnotitel názorů, soudce (proč tak a ne jinak).
   - Encyklopedista (zdroj poznatků a znalostí o zkušenostech vlastních i cizích).
   - Cizelér (jde do detailů, dolaďuje detaily, dbá na „krásu“ řešení).
   - Koordinátor (dává věci do širších souvislostí, shrnuje znalosti).
   - Navigátor (intuice, zda se neodchylujeme od vizí, intuice pro volbu správných metod).
   - Šťoural (dovede najít nedostatky, cit pro rozpory).
   - Provozář (zajišťuje provoz, je schopen organizovat a udržovat pořádek).
   - Hecíř (dovede motivovat).
   - Harmonizátor (dovede tlumit rozpory).
   - Realizátor (rychlá implementace).
   - Moderátor (vedení debat, shrnování výsledků).
   - Normovač (dovede odhadnout náklady a termíny).
   - Pozorovatel, kronikář (pamatuje si vše, co se semlelo, pozdější hodnocení práce).
   - Tahoun (dovede řešení dokončit).
2. **Nežádoucí role:**
   - Agresor (závidí, nesouhlasí, útočí).
   - Negativista (zpochybňuje).
   - Exhibicionalista, sobec (předvádí se, chlubí, prosazuje se).
   - Kecal (tlachá, zdržuje).
   - Playboy (svět je sexuální loviště, žádný jiný zájem).
   - Vládce (autoritativní, intrikuje).
   - Populista (pasuje se na ochránce členů týmu).
   - Kanaďan (miluje drsné vtipy).

   Jeden člen může plnit více rolí. Vyloučit nebo omezit škodlivé role. Řešit nekompatibilitu rolí.

Motivační typy v týmu, workoholik, sobec, kamarád, které funkce jsou snadno nahraditelné a přitom mají vysokou cenu?

- Workoholici (motivováni prací).
- Sobci (kariéristé – motivovaní svou leností případně svou kariérou).
- Kamarádi (motivování interakcí a prací v týmu).
- Snadno nahraditelní – manažeři, dělníci.
- Obtížně – vývojáři, vizionáři, specialisté.

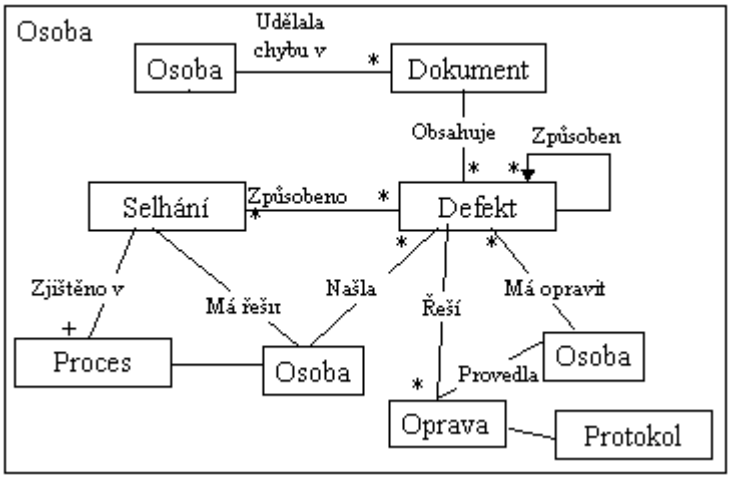
Inspekce, složení týmu, na co je vhodná, jak probíhá?

Struktura týmu:
- Moderátor (vybírá se z osvědčených, klíčová osoba).
- 2 až 3 oponenti, oponované materiály dostanou dlouho předem.
- Předčitatel (může to být i autor).
- Zapisovatel.
- Zasedání se neúčastní administrativní vedoucí!!!

Etapy inspekce:
- Jmenování moderátora.
- Plánování (moderátor):
  - Prověrka připravenosti dokumentů (uvolněny do inspekce, splňují kriteria pro inspekci).
  - Výběr členů týmu).
  - Termín inspekce.
- Úvodní seznámení:
  - Co je úkolem inspekce.
  - Stanovení rolí.
- Příprava:
  - Rozdání materiálů ke studiu.
  - Zajištění organizace (místnost, vybavení).
- Vlastní inspekce:
  - Předčitatel čte pomalu celý dokument, je-li to autor, může doplnit krátké komentáře.
  - Zapisovatel zapisuje poznámky členů týmu a vyhotoví celý zápis.
  - Platí obecná pravidla pro oponentury.
  - Lze požadovat následnou oponenturu.
- Přepracování:
  - Pověřené osoby opraví dokumenty.
- Kontrola:
  - Každou zjištěnou závadu je nutno prověřit, zda byla opravena a oprava nevyvolala další chyby (1/6 oprav bývá chybných).
  - Lze vyvolat následnou inspekci, tu by měl provádět částečně obměněný či úplně nový tým.

Lidská pochybení, defekty a selhání – vysvětlení a diagram včetně oprav. Co je a co není cílem testování?

- Chyba (fault) – pochybení člověka, výsledkem je závada (závady) v některých dokumentech, ty se nazývají defekt.
- Defekt – místo v nějakém dokumentu, které se musí opravit, aby nedocházelo k selháním.
- Selhání (failure) – systém pracuje (bude pracovat) jinak, než má.



Zásady SW metrik, co zásadně ovlivňuje kvalitu metrik?

- Přesnost SW metrik je malá.
- Hodnoty metrik závisí:
  - Na typu projektu, projekty se málo opakují (výjimka: SAP atp.).
    - Velikost projektu a ostrost termínů.
    - Typu úlohy RT*dávka bez přímých následků.
  - Na stavu oboru.
    - Výkon HW, sítě, vývojová prostředí.
    - Paradigmata.
  - Dosažitelné technologii.
  - Kvalitě programátorů (produktivita 1:20).
    - Kvalitnější vývojáři píší lepší programy (rychlost 1:10 a to s méně chybami).
    - Programátoři jsou schopni silně ovlivnit určitou SW metriku, je-li jim dovoleno nebrat ohled na jiné metriky (rychlost a úkor délky).
  - Do jaké míry se řeší podobné problémy.

SW metriky jsou dvojího druhu:

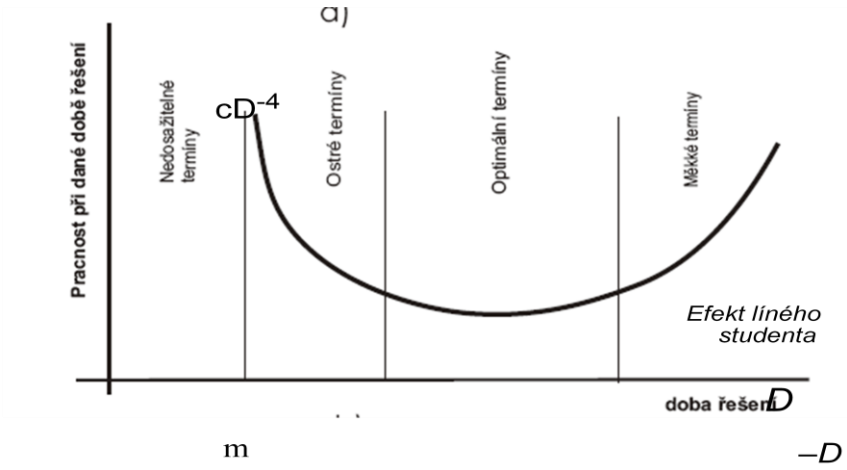
1. Explicitní (externí)
   - Metriky jako délka programů, počet podprogramů/metod atd.
   - Lze je zjistit kdykoliv po skončení vývoje → after process metrics.
   - Zjistitelné formální analýzou textů – jsou tedy zjevné – explicitní.
2. Implicitní (interní)
   - Lze je zjistit většinou pouze během vývoje SW → in process metrics. Jsou známy jen týmu.
   - Metriky jako:
     - celková spotřeba prací (*Prac*),
     - doba řešení (*Doba*),
     - průběh velikosti týmu v čase (*team*),
     - produktivita (*Prod*) – počet jednotek délky (řádků) za jednotku času (měsíc)
     - a počet selhání či výpadků za jednotku času (*Fail*).

Nedosažitelná oblast v rovině, jak se chovají náklady (pracnost) při přibližování se k ní?

Vliv napjatých termínů:

- Dobu řešení nelze v praxi libovolně zkracovat.
- Pro každý projekt existuje tedy mez *m*, pod níž se nelze prakticky dostat. Interval $\langle 0,m \rangle$ se nazývá ***nedosažitelná oblast*** pro daný projekt. *m* je funkcí *Prac*.
- Pro více projektů je nedosažitelná oblast ***oblast roviny*** *(Prac, Doba)*, kde $Doba < \frac{3}{4}\, Prac^{1/3}$

Pracnost u nedosažitelné oblasti:



Chování pracnosti u nedosažitelné oblasti:

- Uvažujme dvě realizace *A*, *B* stejného projektu s atributy

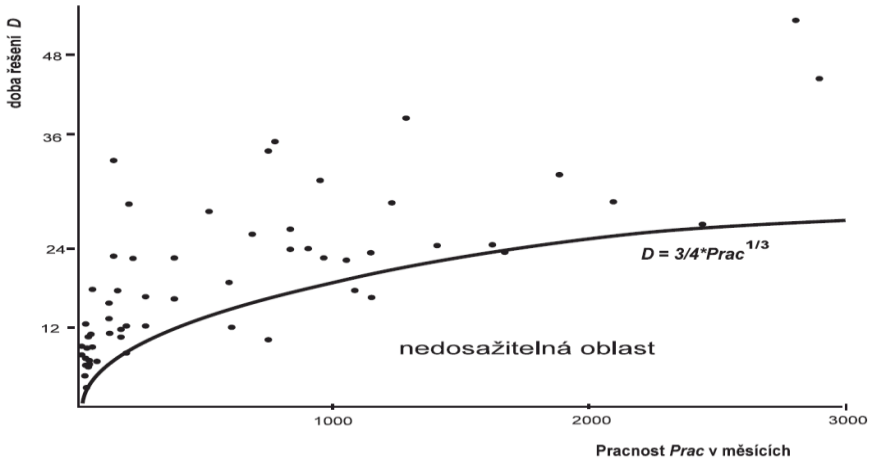
$$Del_A, Doba_A, Prac_A$$

$$Del_B, Doba_B, Prac_B.$$

- Vydělením Putnamových rovnic pro obě realizace dostaneme

$$Del_A/Del_B = (Prac_A/Prac_B)^{1/3} \cdot (Doba_A/Doba_B)^{4/3}$$

Některé starší výsledky pro SW projekty:



- Nechť $Doba_A > Doba_B$. Poněvadž programy psané ve spěchu bývají delší, je možné předpokládat

$$Del_A \leq Del_B \text{ tj. } Del_A/Del_B \leq 1$$

- Po úpravách dostaneme z poslední rovnice

$$1 \geq Del_A/Del_B =$$

$$= (Prac_A/Prac_B)^{1/3}\,(Doba_A/Doba_B)^{4/3}$$

- Z toho po úpravách, považujeme-li hodnoty s indexem *A* za konstantní, dostaneme obdobu Stefan-Botzmannova zákona

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

Čtyřpolní tabulky – princip a příklad (např. dřevo).

Rozhodovací tabulky:

- Umožňují přehledně zapsat, za jakých podmínek učinit příslušnou akci/akce.
- Do horního pole se zapisují požadované pravdivostní hodnoty jednotlivých podmínek (ano A, ne N, na podmínce nezáleží X).
- Do dolního pole se zapisuje značkou x, zda se má příslušná akce pro kombinaci podmínek uvedenou v horní části sloupce provést.

Starý zákazník	A	A	A	A	A	…	N	N
Běžný leasing	A	A	A	A	N		x	x
Rušení nájmu	A	A	N	N	A		x	x
Nový nájem	A	N	A	N	A		A	N
Zařadit zákazníka							x	x
Test platby	x	x	x	x	x			
Zrušení smlouvy	x	x						
Nová smlouva	x		x		x		x	
Faktura	x	x	x	x	x			
Úprava smlouvy	x	x	x	x				

- Vhodné spíše pro dávky a menší úlohy.
- Osvědčuje se pro vyjasnění všech možností.
- Podmínek nesmí být příliš mnoho.
- Spíše jen okrajová metoda.
- Použitelné při specifikaci požadavků i při návrhu systému.
- Používá se v podnicích.

Jaké znáte typy oponentur? Proč používat výstup podobný testování?

- **Inspekce –** jednofázová nebo vícefázová oponentura menších celků podle přísných pravidel. Dobrá inspekce odhalí 80% chyb.
- **Revize (review) –** standardní oponentura většího celku. Možné techniky:
  - běžná oponentura,
  - čtení kódu,
  - strukturované procházení.
- **Simulace –** procházení a napodobování provádění.

- Oponentury jsou nutné u velkých a kritických aplikací.
- Méně formalizované oponentury jsou dobré i u malých projektů.
- Oponentury najdou méně problémů než testy, ale i takové problémy, které se týkají vizí a specifikací.
- Pokud správně provedeno, je nalezení defektu a především jeho odstranění oponenturou mnohonásobně levnější, než jeho nalezení testováním.
- Existují problémy, které lze jen velmi obtížně detekovat testováním, bývají to nedostatky týkající se koncepce.

Strukturované interview / dotazníky / interview.

Specifikace požadavků na IS vždy (i v případě použití customizovaného IS) zahrnuje i zjišťování požadavků u zákazníka. Používají se při tom následující metody:

a) *Interview*: dobře připravený a provedený rozhovor (co nejpřátelštější) o tom, co zákazník dělá a co potřebuje a co si myslí, že by mohl IS zlepšit.
   - Interview může být realizováno ve skupině (*skupinové interview*).

b) *Strukturované interview:* interview s vyplňováním dotazníku. Rozhovor, při kterém se odpovídá na otázky předem připraveného dotazníku.

c) *Dotazníky*: požadavky se shromažďují pomocí dotazníků, které se rozesílají a které budoucí uživatelé vyplňují sami.

d) *Studium dokumentů resp. IS* používaných zákazníkem a studium jeho IS, pokud existuje.

e) *Společný vývoj požadavků*: formulace požadavků skupinou pracovníků zákazníka a konzultantů uživatele. Důležité pro agilní formy vývoje.

f) *Pozorování chodu prací* u zákazníka.

g) *Zapojení pracovníků dodavatele do pracovních procesů zákazníka.*

Obvykle se používá kombinace metod a), e), (případně a), b), e)), ostatní metody se používají řídčeji, obvykle jako metody doplňkové.

Interview:

- Podle jistých pravidel vedený rozhovor mezi:
  - moderátorem – tím, kdo se snaží získat nějaké informace, tím kdo se hlavně ptá
  - a respondentem – tím kdo hlavně odpovídá.
- Business interview je interview mezi byznys partnery. Je to nejčastěji používaná metoda zjišťování požadavků.
- Dlouhodobá spolupráce, prvky výslechu (nesmí to tak vypadat), sledovat logické nekonzistence a souvislosti, hledat nedořešené otázky, jen málo hodin (tak jedna, když víc – přestávky) nesedět tváří v tvář, příjemné prostředí + organizační zajištění (včas oznámit, vhodné prostředí, vhodný čas pro respondenta, vhodné oblečení a chování).

Strukturované interview:

- Výhody: méně závislé na moderátorovi, promyšlené dotazy, vhodné pro opakované implementace (customizace), velmi časté u instalace technologií, důležité se neopomene, lze lépe kontrolovat a vyhodnocovat, lze použít elektronickou podporu, ± větší rychlost.
- Nevýhody: nevyužijí se přednosti moderátora, neodhalí se nečekané souvislosti, obtížná kontrola kvality (lze zlepšit pomocí IS projektu), dosti pracné, nevhodné pro nové oblasti, vhodné spíše pro IS řízení technologií, pracnost vytvoření dotazníků a jejich rigidita.

Průběh interview, čemu se vyhýbat?

- Platí obecné zásady mezilidské komunikace. Zvláště důležité je zachovávat následující zásady:
  - Pozorně naslouchat, neskákat do řeči, reagovat v rozumné míře pokyvováním a i slovně (ano?, pravdu?) a i jinak (oči, mimika), aby byl patrný zájem o to, co se říká (nemusí to být snadné, ale je to efektivní).
  - Požádat občas po podrobnější vysvětlení, nebo shrnout téma svými slovy (zdvořile!).
  - Dávat čas na rozmyšlenou při formulaci odpovědi, případně reformulovat otázku (ale tak, aby to nebylo pociťováno nepříjemně).
  - Nepřipustit dojem, že se jedná o ztrátu času.
  - Necháme prostor na vyjádření postojů a pocitů, snažíme se uplatnit i další aspekty z brainstormingů (střídat podtémata: nápady, fakta, přínosy, kritika, dojmy, jak zařídit).
  - Lze krátkodobě odlehčit jednání odbočením od tématu, slušně se k tématu vrátit (stálá plná pozornost je možná tak asi 20 minut), udělat kratší i delší přestávky (viz zásady hygieny práce a vedení porad).
  - Střídat otázky uzavřené (ano/ne nebo výběr několika možností) s otevřenými.
  - Zapisovat (elektronické záznamníky jsou efektivní, respondent s nimi ale musí souhlasit, pokud o nich neví nebo s nimi ne zcela souhlasí, mohou ohrožovat účinnost interview), dobré mohou být flipcharty (zápis utřídí myšlení, věci se lépe pamatují, lze následně zpracovávat).

Rozdíl mezi auditem a recenzí (interview)?

- Audit – soubor kontrolních akcí prováděných obvykle pracovníky *mimo tým* (management, auditoři – často z poradenských firem, členové týmu jako kontrolované osoby).
- Interview – vnitřní oponentura, *uvnitř týmu*.

Co je SW proces. Životní cyklus SW procesu. Proces vývoje s prototypováním.

- *SW proces* – síť kroků potřebných k vytvoření SW systému. (Obdoba byznys procesu.)

Životní cyklus procesu:

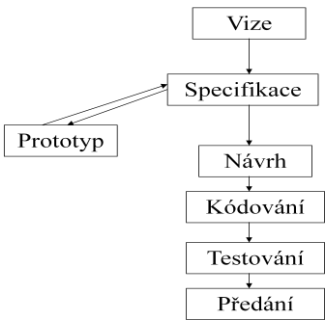
- *Analýza požadavků na softwarový proces*.
  - Na základě vlastností cílového produktu a prostředí, vlastností týmu a prostředků, podmínek smlouvy se zákazníkem se zvolí základní vlastnosti procesu, např. inkrementální model.
- *Vývoj modelu.*
  - Cílem je vytvořit proveditelný softwarový proces. Vývoj modelu procesu obvykle zahrnuje plán tvorby modelu, volbu architektury procesu, návrh modelu, instanciaci a validaci modelu provedením inspekcí, případně ověřením na pilotním projektu. Často stačí instanciace existujícího modelu.

- *Přizpůsobení (tayloring)*.
  - Adaptace modelu pro konkrétní projekt zpřesněním významu jednotlivých kroků a doplněním kroků.
- *Plánování*.
  - Stanovení termínů provedení jednotlivých kroků procesu.
- *Instanciace*.
  - Doplnění údajů o agentech (kdo co kde provede) a zdrojích (co je k provedení třeba). U velkých projektů se připouští instanciace počátečních kroků procesu v situaci, kdy definice celého procesu ještě není dokončena.
- *Evoluce*.
  - V průběhu provádění procesu dochází obvykle ke změnám v důsledku nově vzniklých nebo nově zjištěných skutečností. To může ovlivnit definici softwarového procesu, případně i model procesu.
- *Provedení (enactment)*.
  - Podle modelu, který nyní slouží jako plán činnosti, se uskuteční vývoj softwaru. Při tom se připouštějí úpravy modelu při výskytu neočekávaných skutečností.

Proces vývoje s prototypováním:

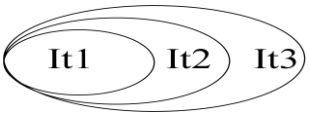
- Softwarový prototyp se jako částečně funkční model cílového řešení vytváří z následujících důvodů:
  - ověření správnosti a úplnosti specifikace požadavků,
  - ověření úplnosti a správnosti funkcí a návrhu struktury systému,
  - předběžný odhad nákladů a rizik realizace.
- Existují následující základní typy SW prototypů:
  - *Potěmkin* (obrazovkový prototyp): Model cílového systému, který simuluje obrazovky dialogů. Vlastní výkonná část systému zcela (nebo téměř zcela) chybí. Tento typ prototypu vlastně simuluje budoucí rozhraní systému.
  - *Neúplný*: Modeluje pouze některé funkce.
  - *Jiný kůň*: Systém je téměř úplný, ale funguje na jiném hardwaru resp. nad jiným základním softwarem. Časté pro software pro jednočipové počítače.
  - *Hlemýžď*: Prototyp je realizován v jazyce, který neumožňuje cílovou efektivnost (např. v jazyce PROLOG).
  - *Nerudný*: Prototyp není uživatelsky příjemný, nemusí být ani dostatečně stabilní.
  - *Záludný*: Nereaguje správně na chyby v datech a chyby obsluhy.
  - *Samotář*. Není schopen propojování (PROLOG).
- Prototyp je určen k ověření specifikace funkcí a není určen k cílovému řešení.

- Použití prototypu – klasická varianta:

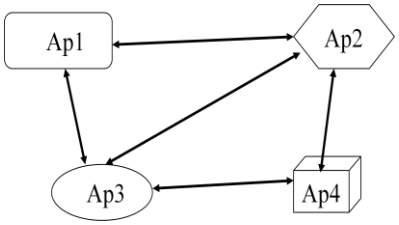


Typy vývoje systému (inkrementální a iterativní). Který se více využije pro agilní vývoj?

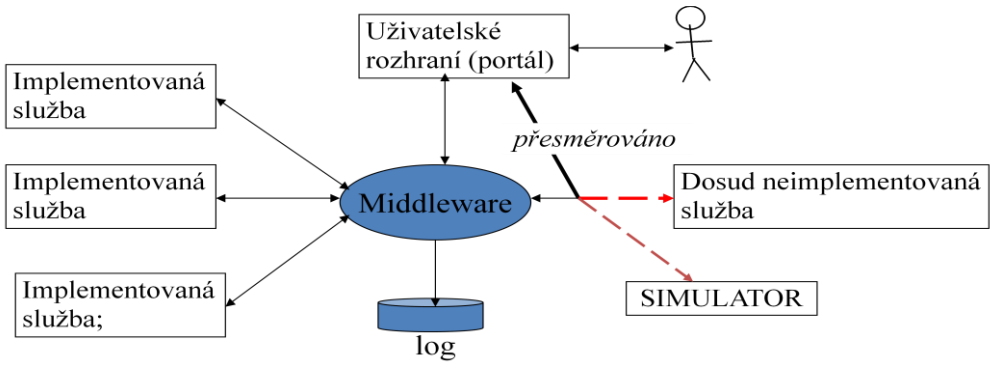
1. **Iterativní vývoj** – jedna rostoucí aplikace (může být distribuovaná), pro vývoj iterace potřebuji mít již vyvinuté jádro a **to potřebuji při vývoji iterace používat. Typické pro agilní vývoj.**



2. **Inkrementální vývoj** – propojují se (různorodé) aplikace Ap1, Ap2,… Většinou pomocí výměny zpráv nebo dat přes middleware, nebo přístupu ke společným datům (datovým úložištím). Při vývoji přírůstku **nepotřebuji mít k dispozici zbytek systému!!! Agilita potřebuje doladit.**

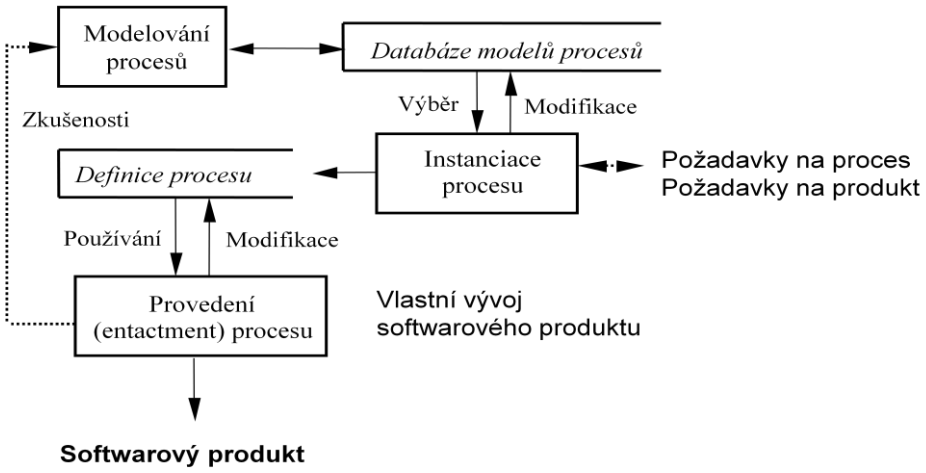


Dá se upravit prototypování v SOA? Proč to často vypadá, že uživatel neví, co chce, i když tomu tak často není.



- Zprávy lze přesměrovat **beze změny implementovaných** služeb buď na uživatelské rozhraní (efektivní prototyp, použitelné i za běhu systému), nebo dokonce na simulátor (ladění RT systémů).

Schéma vývoje procesu, modelování,…



Capacity Maturity Model (CMM) pro SW procesy. Proč není vhodný pro malé firmy?

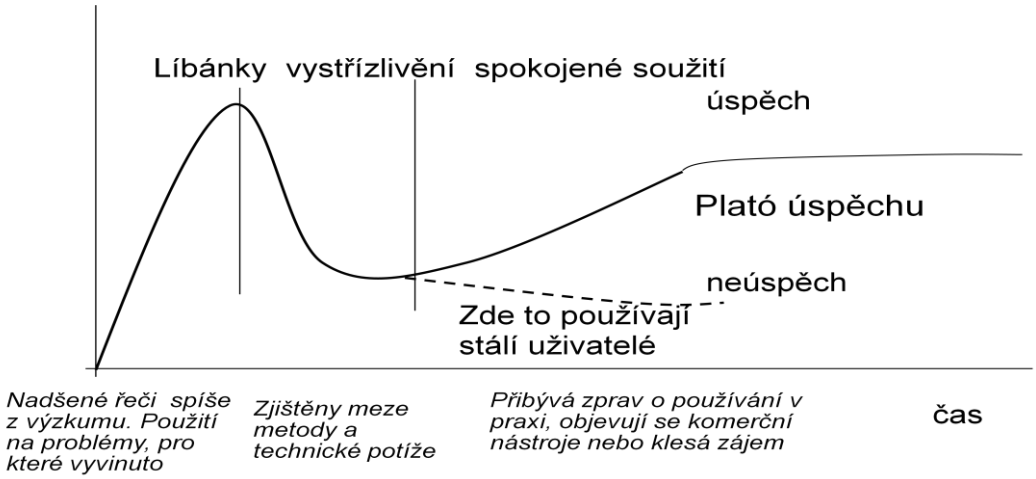
- Zdokonalování procesů.

1. Počáteční úroveň – jak si kdo co pamatuje, když odejde, znalost se ztratí.
2. Opakovatelná úroveň – standardy a DB v počátcích a na jednotlivosti. DB průběhu projektů. Není statistická analýza.
3. Definované procesy – standardy od specifikací požadavků až po managementu procesů a jejich provázanost.
4. Řízené procesy – metriky – sběr a vyhodnocování, trendy, chybová místa, analýza s používáním matematické statistiky.
5. Optimalizované procesy – procesy neustálého vylepšování.

- *Nezaručuje úspěch, lidé vždy klíčem, dobrým pomáhá, lemplové se mohou schovávat za papíry. Funguje spíše u větších podniků.*
- *Pravidlo:* vyšší úroveň zahrnuje vždy všechny nižší úrovně.
- Úrovně 4. a 5. a do značné míry i 3. jsou dosažitelné jen u velkých organizací.
- Problém dostatečně velkých souborů dat.
- **Nebezpečí byrokratizace vývoje softwaru.**

Pozitivní ohlasy přijetí nového nástroje v jednotlivých etapách.

Funkce množství pozitivních ohlasů:



Etapy zvládání nové metodiky:

- *Líbánky:*
  - Metodika se používá v oblastech, pro které byla především vyvinuta, **tam** se osvědčuje.
  - Pracují s ní kvalitnější lidé, kteří se snaží touto cestou získat slávu, nejsou komerční nástroje velkých výrobců.
- *Vystřízlivění:*
  - Nehodí se na vše, výrobci SW jsou stále opatrní, náraz na meze.
  - Má mouchy (implementační, metodické).
  - Vyžaduje zaškolení a změnu návyků (to někdy až při změně generace vývojářů, viz objektovou orientaci).
  - Inovátoři se zajímají o nové hity.
- *Soužití* (ne vždy k němu dojde):
  - Většina nedostatků se odstraní.
  - Systém používají stálí uživatelé a ti si s problémy nějak poradí, nebo si na ně zvyknou. Přibývá komerční podpora od velkých firem a výuka na školách.
  - Časem dojde k ustálenému stavu (plató).

Problémy s používáním CPM (critical path method – metoda kritické cesty) při řízení SW projektů (proč selhává metoda kritické cesty). Jak lze léčit efekty líného studenta? Zpevňování norem a multitaskingu. Souvisí to nějak s týmovou loajalitou?

- Metoda kritické cesty se dá zobecnit tak, aby byl zohledněn problém soutěže a společné zdroje.
- V praxi se osvědčuje při plánování činností, které se opakují a kde lze doby řešení dobře odhadnout, dokonce změřit.
- V případech, kdy jsou odhady nejisté, jak je pravidlem při vývoji SW, nebývají výsledky dobré.
- MS Project je založen na metodě kritické cesty. MS Project využívá rozhraní založené na Ganttových grafech.

Problémy kritické cesty, boj o měkké normy:

- V SW bývá obvykle málo spolehlivých dat potřebných pro dobrý odhad (malá opakovatelnost, nové typy požadavků, změny technologie) – je nutno se spolehnout na odhady řešitelů jednotlivých úkolů.
- Pokud někdo nesplní termín, má potíže.
- *Důsledek: odhady se licitují a volí tak, aby byla malá pravděpodobnost, že se termín nesplní (tj. volí se horní hranice konfidenčního intervalu).*

Problémy kritické cesty, boj proti zpevňování norem:

- V dobách řešení jsou velké rezervy. Přesto se i pak se termín obvykle prošvihne. Proč?
- Pokud někdo skončí dříve, práci neodevzdá z obavy, že příště by nevylicitoval měkký termín, dali by mu tvrdší termín a on by čelil riziku, že ho nesplní.

Problémy kritické cesty, syndrom líného studenta:
- V dobách řešení jsou velké rezervy, proč i pak se termín projektu obvykle prošvihne?
- Pokud má někdo rezervu, nezačne včas pracovat (dělá na něčem jiném). Pracovat začne, až když má dojem, že už je třeba začít. Pak ale často (v desítkách procent) nestihne svůj termín. Skluz už se obvykle nedohoní v důsledku boje proti zpevňování norem.

Problémy kritické cesty, efekt multitaskingu a vyšší moci:
- V dobách řešení jsou velké rezervy, proč i pak se termín projektu obvykle prošvihne?
- Pokud se neudělají opatření, vznikne často situace, kdy nelze na určitém úkolu začít pracovat, protože se dělá na jiném úkolu, jindy nelze začít pracovat, protože předchůdce neskončil včas, někdy se jen lenoší – efekt líného studenta.
- *Výsledek: nutně dojde často ke skluzům a je malá naděje, že se skluzy podaří dohnat.*

Metody kritického řetězce:
- Kritický řetěz funguje jen, když jsou řešitelé ochotni odhalit reserv a neskrývat, že jsou hotovi dříve, než se plánovalo.
- Odměny za dodržení termínu (za zkrácení není další bonus, vedlo by to opět k licitování).
- Termín se odvozuje z očekávané doby řešení.
- Navíc má každá činnost odhad doby, když jdou věci špatně (horní hranice konfidenčního intervalu).
- Termín pro celý projekt se určuje jako odhad horní hranice konfidenčního intervalu jeho řešení.
- Každý začne pracovat hned, jak je jeho předchůdce hotov. To ale znamená, že je nutné nějak vyloučit efekt multitaskingu, To se řeší tak, že se dané činnosti postupně stále přesněji oznamuje, kdy bude třeba začít pracovat na daném projektu (činnost je tedy spravována jako autonomní služba).
- Práce se odevzdává v okamžiku, kdy je hotova. Její začátek se ale postupně zpřesňuje.
- *Důsledek: dosti často se daří, aby práce trvala přibližně tak dlouho, jako kdyby byla zcela nezávislá.*

Vanová křivka. Jak poznat, kdy je třeba aplikaci vyměnit (příznaky, že SW je opotřebovaný)?

- Znázorňuje opotřebení SW, frekvence selhání.
- Záběh (etapa investic) – klesající křivka → opravy chyb, které prošly testováním, corrective maintenance, prvá vylepšení a přizpůsobení.
- Provoz (etapa maximálního užitku) – ustálená křivka.
- Opotřebeno (zmenšování užitku) – rostoucí křivka → SW se stále opravuje, vylepšuje a tím se zanáší další chyby. Za každý odstraněný defekt přibude nový. Růst počtu selhání v důsledku rozpadu integrity systému.

Kolik % nákladů na kódování, metriky kódování.

- Kódování není dnes hlavní problém, patří k nejméně pracným etapám vývoje SW.
- *Jedna sedmina pracnosti* vývoje a pracnost kódování se dále zmenšuje.
- Ví se, jak na věc a jak to učit a standardizovat.
- Systémy podpory programování (vizuálnost, CASE, Delphi, Power Builder, …), asi nevhodné pro SOA.
- Servisní orientace, objektová orientace.
- Problém je ve specifikacích, to je úzké místo.
- Při kódování je výhoda mládí. Nebezpečí, že se mladí omezí na kódování a nic dlouhodobějšího se nenaučí.

Typy testů. Jaké typy testů dělá kodér a na jaké se hodí specializovaní testeři? Variace kodér-tester.

- Částí, (unit tests):
  - samostatných kusů programů,
  - dost často testují programátoři sami.
- Integrační:
  - shora/zdola,
  - selektivní, sendvič, jádro a programy.
- Regresní (zopakování většiny testů):
  - může být příliš náročné na uživatele a na provoz, v agilním vývoji spíše pravidlo.
- Funkcí:
  - ucelených akcí systému.
- Systému:
  - v simulovaném provozu jako celek.
- Předávací:
  - podle smlouvy.
- Test užíváním:
  - zkušební provoz.
- Test simulací nebo prototypem

Programátoři a testeři → tři varianty „spolupráce“:

1. Programátor je současně tester:
   - populární, rychlé, málo účinné (vadí u kritických aplikací).
2. Tester je specifická role, bílé skříňky:
   - tester spolupracuje s programátorem při nápravě selhání.
3. Tester testuje černé skříňky, testeři nemají zdrojové kódy ani kontakt s programátory:
   - nejúčinnější je 3, ale je to velmi drahé a vyžaduje to profesionály.

Integrační testování, typy testů podle dekompozice (shora/zdola). Proč nelze testy zcela automatizovat?

- Integrace zdola:
  - Je třeba mnoho pomocných dat a programů.
  - Funkce systému se testují poměrně pozdě, což opožďuje předvedení systému uživateli.
  + Moduly jsou obecněji použitelné (méně závisí na změnách funkcí systému).
  + Ověřují se možnosti implementace.

- Integrace shora:
  + Je třeba méně pomocných dat a programů.
  + Funkce systému a rozhraní se testují a mohou předvádět poměrně brzy.
  - Moduly jsou použitelné jen v daném prostředí (někdy je to výhoda).
  - Chyby na vyšších úrovních mohou být fatální (až příliš pozdě se zjistí problémy s implementací).

- Jde automatizovat jen zčásti. Důvody: algoritmicky nerozhodnutelné problémy, mrtvý kód, otestování všech částí/větví, nekonečný cyklus.
- Nelze plně automatizovat, tvůrčí problém – heuristiky, vždy nějaký defekt zůstane.

Zásady a problémy vývoje uživatelského rozhraní.

Zásady vývoje rozhraní:

1. Považovat vývoj rozhraní za relativně autonomní problém.
   - Vytvořit v systému autonomní komponentu/vrstvu pro implementaci rozhraní.
2. Přesvědčit management i zúčastněné, že je rozhraní skutečně problém.
   - Upozornit na nutnost postupného vývoje.
3. Zajistit prostředky a účast lidí.
4. Naplánovat etapy (před oživováním systému, během oživování, při provozu).
5. Použít pokud možno to co existuje a poučit se od úspěšných realizací.

Obtížnost vývoje rozhraní:
- Závisí na psychologii a znalostech uživatelů.
- Souvisí s řadou oborů (ergonomie, výtvarné umění, psychologie).
- S časem se požadavky mění.
  - Nové objevy a techniky (GUI, okna).
  - Různé typy aplikací (př. CAD).
  - Začátečníci a pokročilí.
  - Všeobecný vývoj, rozvoj počítačové gramotnosti a v neposlední řadě módy.
  - Technický pokrok (viz např. vynález myši, nově hlasový vstup).
- Nutno ověřovat na lidech.

Uživatelské rozhraní – co preferují začátečníci / pokročilí. Zásady používání nápovědy.

- Začátečník preferuje snadnost naučení a snadnost pamatování, pokročilý spíše rychlost.
- Začátečníci preferují GUI a učení metodou pokusů a omylů. Je třeba najít cesty přechodu ke klávesovým zkratkám a znakovému rozhraní (mnohdy efektivnější a rychlejší).
- Nespoléhat na nápovědy, nápovědy chápat spíše jako pomoc v nouzi, při zaučování a při nečekaných situacích.

Principy testování uživatelského rozhraní.

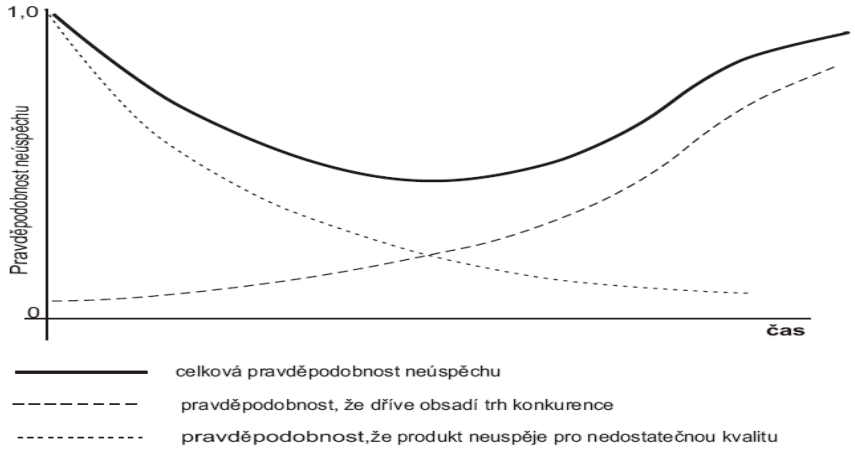
- Předvedení a zkoušení prototypů několika uživateli
  - Někdy se vyplatí prvý test dělat s „brigádníky“ To je zvláště vhodné pro ověřování variant pro začátečníky.
  - Ne vždy lze, např. je-li nutná věcná znalost toho, co systém podporuje.
  - Log průběhu

- Zahájení testu:
  - Příprava nástrojů a lidí, místa.
  - Test provádí obvykle člen skupiny uživatelů za přítomnosti vývojáře.
    - Je třeba brát ohled na velké individuální rozdíly mezi jednotlivými uživateli a také mezi nezkušenými a zkušenými pracovníky. Ověřování UI je proto třeba provádět s větší skupinou pečlivě vybraných budoucích uživatelů.
    - Bývá výhodné provést nejprve zaškolení v ovládání systémových prostředků.
    - Optimální počet testujících je 3 až 6.
  - Testeři pracují s tou částí UI, se kterou budou skutečně pracovat při provozu systému.
  - Při organizaci testů je žádoucí navodit atmosféru spolupráce:”Pracujete na tom, aby vám to, co budete používat, sloužilo dobře. Testuje se návrh, ne vy“.
  - Uživatelé nemají pracovat ve stresu.
  - Výsledky testů by měly být anonymní.
  - Upozornit, že se systém na základě požadavků testerů může změnit.
  - Testování lze do jisté míry provádět na částečně funkčních prototypech. Vždy je však nutné nakonec provádět testy i na oživeném systému. Důvodem je potřeba testovat doby odezvy.
  - Je obvyklé, že se na základě analýzy výsledků testů UI podstatně mění. Testovat je nutné i nové verze UI.
  - U testování jen uživatelé, hlavně ne jejich šéfové.

- Provedení testu:
  - Je důležité, aby testování nebylo přerušováno telefony, návštěvami atd.
  - Je výhodné, když uživatel provede na počátku rychle a úspěšně nějakou část dialogu, je pak méně nervózní.
  - Atmosféra při testování by měla být uvolněná.
  - Nedávat najevo, že uživatel je pomalý nebo že dělá chyby.
  - Vyloučit kibice, včetně (to především) šéfů testujícího.
  - Zastavit testování, vznikne-li pocit, že tester toho má již dost.
  - Uživatel při testování „myslí nahlas“ (říká, co zrovna dělá a proč, vyjadřuje překvapení, kritiku, nebo potěšení, případně že neví).
    - Problém je, že to zdržuje při práci, zvláště ty zručnější.
    - Pokud uživatel říká, že neví nebo příliš dlouho váhá, lze mu pomoci (nemělo by být často, nutno zaznamenat).
  - Je dobré na konec testování s konkrétním uživatelem se zeptat, co si o systému jako celku myslí.

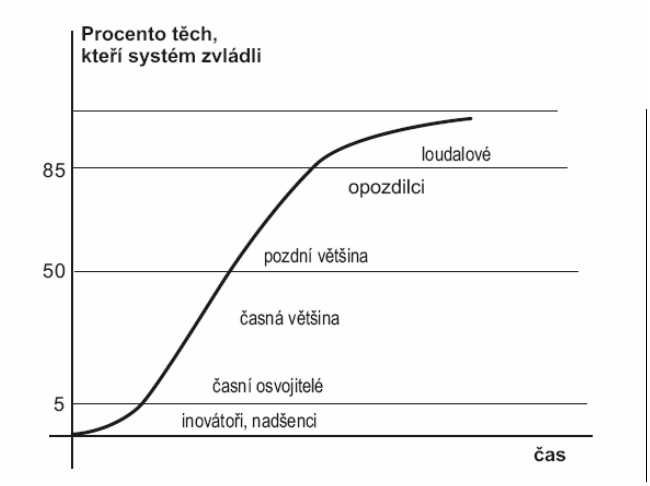
- Vyhodnocení testu:
  - Požádat uživatele o celkový názor, především o to, jak je spokojen.
  - Výsledky testů zaznamenat v dohodnuté formě.
  - Provést analýzu žurnálu (ten je dodré sledovat i za (zkušebního) provozu
  - Výsledky zavést do databáze projektu (pokud existuje).
  - Sestavit vlastní hodnocení.
  - Hodnocení problémů:
    - 0 – celkem OK.
    - 1 – kosmetická – opravit, bude-li čas.
    - 2 – méně závažná – opravit, pokud nebudou závažnější.
    - 3 – závažná – opravit co nejdříve, ale není překážkou pro zahájení provozu.
    - 4 - katastrofická – systém nelze provozovat.

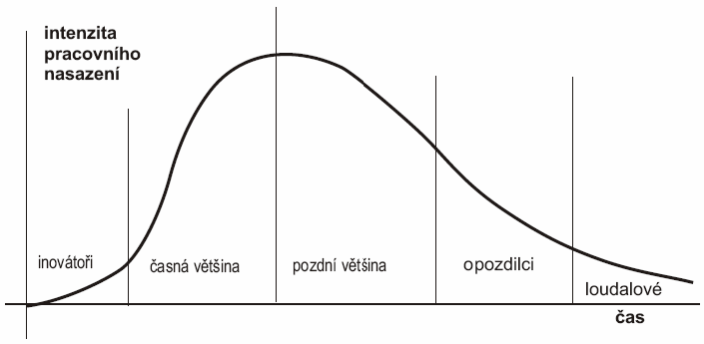
Kdy je optimální ukončit testování?



- Obecně je pocit, že lépe před minimem než po minimu.
- Po minimu z dlouhodobého hlediska zvětším možnost, že vyroste konkurence.
- Také ne zcela správný pocit, že už je to dobré.
- Intuice manažera odhadne nejvhodnější dobu.

Křivka učení + typy lidí k ní.





- Inovátoři jsou motivováni novinkami, nikoliv zlepšováním své práce.
- Časní osvojitelé chtějí zlepšovat svou práci a novinky při tom vítají, učí se rychle, mívají vysokou reputaci u uživatelů, to jsou ti praví spojenci.
- Časná většina inovace spíše vítá, ale chce mít svůj klid.

- Křivka učení – nemá být příliš strmá (intensita příliš vysoká) ani příliš plochá.
- Je nutné chápat jako učení a zajistit: kvalitní vyučující, čas, pomůcky a prostředí, hodnocení pokroku (známkování).
- Optimální – rychlé pochopení základních funkcí a po zaškolení i osvojení složitých funkcí. Lepší výsledek než při ustrnutí na začátečnické úrovni, nebo příliš vysoká náročnost na začátku.

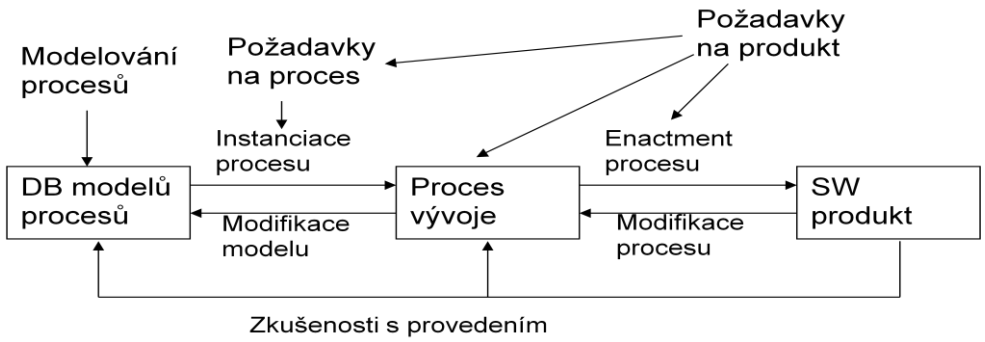
Jak probíhá čtení kódu, co je výsledkem, opravují se chyby?

- Zjednodušená inspekce – tým je dvojice.
- Jeden čte a říká, co je jaký význam daného místa.
- Druhý poslouchá a někdy také čte a snaží se porozumět.
    - Často stačí jen poslouchání (mluvení do vrby).
    - Důležité: snažit se oponovat.
    - Zaznamenávat zjištěné problémy, pokud možno je neřešit ihned.

Varianty:
- Kód se čte přímo z monitoru.
- Kód se rozesílá (mailem) s návrhy změn.
- Nutné pro vývoj na bázi open source.
- Pro distribuované týmy je to nutnost, lze zapojit širší skupinu.
- Problémy: obtížnější kontrola, větší nároky na disciplinu práce a také často pomalejší.

Co je enactment procesu + diagram modelování procesů.

- *Provedení procesu (enactment)* – provedení procesu může být úkolem lidí či softwarových nástrojů, nebo obou. Výsledkem provedení procesu je softwarový produkt.
- *Omezení provedení procesu* – proces a jeho kroky musí obvykle splňovat řadu podmínek, jako jsou povinnosti autorizace a dodržování standardů.



Co je to usability a na co má vliv, jaké jsou její pozitivní dopady (na produkci apod.)?

- Usability – je to faktor akceptovatelnosti SW.

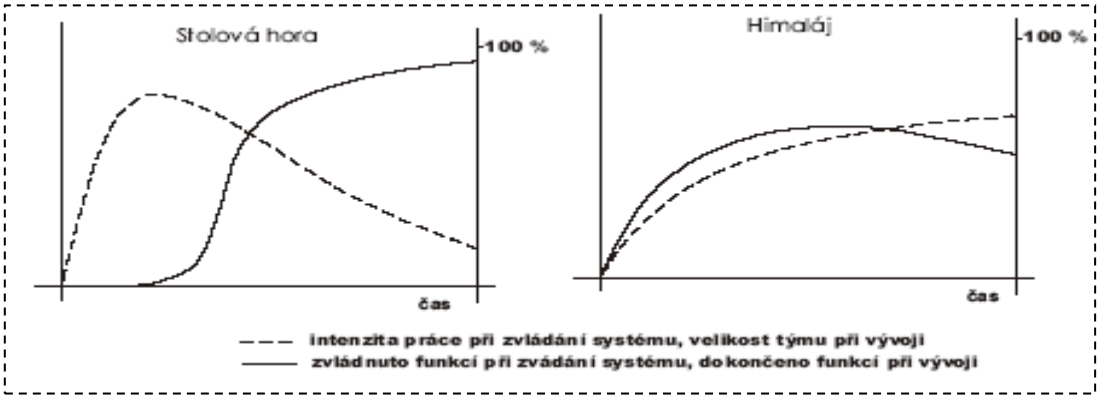
Faktory použitelnosti SW:
- Snadnost naučení.
- Efektivnost při používání.
- Dobře se pamatuje, jak systém používat.
- Málo chyb uživatele způsobených špatným ovládáním rozhraní.
- Subjektivní příjemnost práce se systémem pro uživatele.
- Dobré ergonomické vlastnosti.

Efekty uživatelského rozhraní:
- Ovlivňuje spokojenost zákazníka se systémem.
- Je důležité i z čistě obchodního hlediska. Je obalem softwaru.
- Podstatně ovlivňuje složitost ovládání IS.
- GUI je ergonomicky náročné.
- Čtvrtina požadavků se týká rozhraní.
- Získat prostředky (někteří doporučují i více než **10 %** nákladů na vývoj).

Himaláj a Stolová hora. Co jsou to neinvestiční výdaje. Jaký je manažerský úkol při Stolové hoře.

- Kolik dávat do „neproduktivních“ činností – takových, jejichž výstup není částí projektu (do lidí, nástrojů, prostředí) → školení, vývojové prostředí/nástroje (vlastní nákup), HW + základní SW.

- m/n = 1/2 – 1/(2c), kde n = člověkoměsíce k dispozici, m = investované člověkoměsíce



- Himaláj (jde se hned na věc) – jde pomalu a těžko odlaďuje – začneme brzo stoupat, ale konec v nedohlednu.

- Stolová hora – rizika: nákup šuntu, pomalý vývoj vlastních prostředků + vhodní lidé, jak poznat, že neproduktivní investice není neproduktivní = otázka kvalifikace.

Manažerský úkol:
- Na vývoj nových nebo na zvládnutí kupovaných nástrojů je vhodné věnovat téměř polovinu prostředků.
- Znatelný přínos pro daný projekt přinese nový nástroj jen tehdy, zvýší-li produktivitu práce alespoň dvakrát.
- Je třeba rozvíjet prostředí a nástroje (vývoj, nákup), musím ale na to mít schopné lidi. *Stejný efekt ale může mít investice do znalostí lidí.*

Vyplatí se studium již existujícího systému, dokumentace a procesů při zavádění nového? Výhody a nevýhody.

Výhody:
- Sleduje se to, co existuje.
- Dají se zjistit důležité typy činností a dat.
- Levné.
- Dá se použít jako inspirace.
- Lze části IS s výhodou použít.

Nevýhody:
- Často jde o zastaralé a i nepotřebné funkce.
- Může inspirovat zastaralá řešení.
- Snaha o znovupoužití existujícího nevyhovujícího systému.

- Pečlivě sledovat, kdo jakou funkci používá. Sledování existujícího IS má podobné výhody a nevýhody jako sledování dokumentů.
- Současný IS je ale cosi jako prototyp nového IS, pokud postupujeme rozumně.
- SOA umožňuje leccos ze starého IS integrovat do nových IS.

Jak využít stávající IS?
- Jako prototyp.
- Získání podkladů pro následující rozhodnutí:
  - Co stávající IS neřeší a je to třeba zajistit?
  - Jak modifikovat stávající použitelná data?
  - Jak využít a modifikovat stávající funkce?
  - Jak naplánovat přechod na nové (souběh, velký třesk, v SO lze postupné překlápění služeb)?
  - Výše uvedené skutečnosti by měly být součástí specifikace požadavků.

Správa konfigurace? Co to je, k čemu se používá? Postupy zjištění správné konfigurace v dokumentech? (nebo Proč se používá řízení konfigurace a jak probíhá?)

- Dosáhnout toho, aby byly vyvinuty, prověřeny a do celku se dostaly komponenty (prvky konfigurace), které pro danou verzi výrobku patří k sobě.
- Obecně technický problém (ISO normy).
- Stanoví se (někdy v etapách) prvky konfigurace, každý prvek má jméno a id, který určuje, do které konfigurace patří.
- Prvky projdou cyklem od zadání k převzetí.
- Správa konfigurace hlídá, zda jsou prvky k dispozici, tj. zda prošly i řádným vývojem a byly prověřeny.
- Do konfigurace (verze) *x.y.z.w* patří prvky se jménem v seznamu a z těch s id mající nejdelší společný prefix s id konfigurace.

Princip určování náročnosti a nákladovosti pomocí COCOMA + funkční body.

- Dva principy odhadu:
  - přes odhad velikosti – COCOMO,
  - pomocí odhadů složitosti interakce s okolím – Function Points.
- V obou případech opravné (Pišvejcovy) konstanty.
- V obou případech dvě varianty odhadu použitelné v různých etapách životního cyklu.

Odhad COCOMO (Constructive Cost Model):

- Vychází z odhadu délky programu v tisících nově napsaných řádků.
- Prvý krok odhadu vychází z typu SW díla (organický typ, přechodný typ, vázaný (embeded) typ).
- Pro jednotlivé typy SW dostaneme následující odhady spotřeby práce *Prac* v člověkoměsících a doby vývoje *D* v měsících z délky *Del* programů v tisících řádků:
  - Organický typ …. $Prac_{čm} = 3{,}2 \times Del^{1{,}05}$ $D_{měs} = 2.5 \times Prac^{0{,}38}$
  - Přechodný typ … $Prac_{čm} = 3{,}0 \times Del^{1{,}12}$ $D_{měs} = 2.5 \times Prac^{0{,}35}$
  - Vázaný typ ……. $Prac_{čm} = 2{,}8 \times Del^{1{,}20}$ $D_{měs} = 2.5 \times Prac^{0{,}32}$

Funkční body (Function Points):

- Function points 1
  - Ze složitosti I/O se vypočtou neadjustované FP.
  - Takto získaný odhad se znásobí funkcí číselných hodnocení atributů projektu a kalibrační konstantou.
  - Tím se získá odhad pracnosti, resp. délky.
  - Dále se postupuje podobně jako v COCOMO.
  - $Del = c \times F$ (c … konstanta, F … informační objem) $Prac = d \times Del^{b}$

$$F = w_1 \cdot (IN) + w_2 \cdot (OUT) + w_3 \cdot (ENQ) + w_4 \cdot (FILE) + w_5 \cdot (FILEE)$$

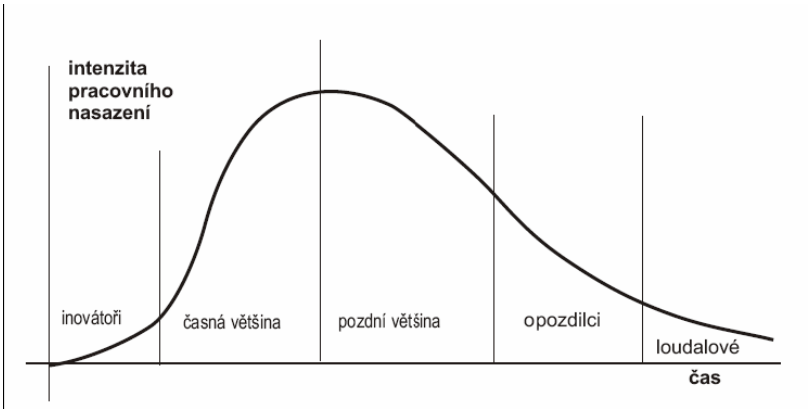
    - Pro nejjednodušší variantu odhadu je $w_1 = 4$, $w_2 = 5$, $w_3 = 4$, $w_4 = 10$, $w_5 = 7$.
    - IN a OUT je počet logicky nebo formátem odlišných vstupů a výstupů.
    - ENQ je totéž pro komunikaci s terminálem (jinou aplikací).
    - FILE obdoba pro zápisy a čtení do vlastních DB.
    - FILEE totéž pro společné DB.
  - $c = 2/3 + 0.001 \sum^{14} F_i$
    - F_i jsou atributy projektu (celkem 14 atributů) hodnocené od 0 do 5 podle následujícího hodnoceni:
      - 0 – daný faktor neexistuje nebo nemá vliv,
      - 1 – nevýznamný vliv,
      - 2 – mírný vliv,
      - 3 – průměrný vliv,
      - 4 – významný vliv,
      - 5 – velmi silný vliv na *celou* architekturu a programování.
- Function points 2
  - Neadjustované body součtem bodů elementárních akcí (transakcí).
  - Používá se více atributů projektů (celkem 19 – 14 převzatých z FP 1, 5 nových) a jiné hodnocení atributů.
    - Hodnocení atributů: 0 – žádný vliv, 1 – průměrný vliv, 3 – kritický vliv.

- ***Nechť a (p) je procento výsledků dosažené p procenty nejlepších. Co je známo o a (p)? Jakých výsledků dosahuje 1% nejlepších? Jak tyto poznatky ovlivňují stavbu týmu? (nebo Kolik % práce udělá 1% nejlepších? Matematicky vyjádřit…)***

- $a(p) = c \cdot p^{1/2}$ kde c nepříliš silně závisí na typu činnosti.
- **Procento nejlepších udělá 7,5 % výsledků**, 20 % nejlepších udělá polovinu práce, 40% nejhorších neudělá skoro nic.

- Vědomí důležitosti talentu a kvality lidí (nejlepší udělají nejen nejvíce výsledků na hlavu, ale také udělají i nejlepší výsledky).
- Pokud jsou ve větší skupině problémy na straně kvalitních lidí, je nutné zkoumat, čím to je a zda to vadí (např. zda se nejedná o rutinní práce, kde se mohou kvalitní lidé cítit nevytíženi).

- ***Skupiny uživatelů z hlediska jejich přístupu k zavádění systému (např. loudalové). Ve které skupině hledat spojence při zavádění systému a proč. (nebo Jaké rozlišujeme skupiny uživatelů při zavádění systému. Na kterou skupinu se zaměřit?)***



Koho získat jako spojence?

- Inovátoři jsou motivováni novinkami, nikoliv zlepšováním své práce.
- Časní osvojitelé chtějí zlepšovat svou práci a novinky při tom vítají, učí se rychle, mívají vysokou reputaci u uživatelů, to jsou **ti praví spojenci**.
- Časná většina inovace spíše vítá, ale chce mít svůj klid.

- ***Údržba, pracnost údržby, jak celkově hodnotíme projekt 100 body? Co je náplní údržby, co se dokončuje z vývoje?***

Druhy údržby:
- Corrective – odstranění defektů, které nebyly detekovány oponenturami a testováním.
- Enhancive – vylepšování.
- Adaptive – přizpůsobování změnám platformy, přenos, změny norem.

Etapy údržby:
- Převzetí.
- Etapa investic – prvá vylepšení a přizpůsobení.
- Etapa maximálního užitku – vylepšení požadované uživateli, regresní testy, stabilní provoz.
- Zmenšování užitku – vylepšení pro další uživatele, zlepšování výkonu, roste počet problémů.

Pracnost údržby:
- Složitost údržby roste s časem (důvod stoupající větve vanové křivky).
- Co snižuje pracnost údržby?
  - Správné specifikace (správné požadavky ale také správná dekompozice).
  - Servisní architektura.
  - Objektová orientace.
  - Správné procesy vývoje (inkrementálnost, agilnost) a kvalita dokumentace, kvalita oponentur a testů.
  - Přenositelné technologie (Java).
  - Vývojové nástroje (menší rozsah dokumentů, srozumitelnost, podpora korektnosti oprav,…).
  - Dobře vedené programování (používám, co je napsáno, používám moderní metodiky (OO, SOA,…), požívám dohodnuté standardy).
  - Kvalita technik údržby

Hodnocení projektu 100 body:

	Pracnost	Čas
Vize	5	8 – 10
Specifikace	15 – 25	25 – 40
Návrh / generace	15 – 20	cca 20
Kódování	15 – 20	10
Testování	35 – 40	25 – 30
Celkem	100	100
Údržba	200	
Celkem s údržbou	300	

Atributy kvality dat. Kdy je vhodné použít datové úložiště.

Kvalita dat:
- V managementu se musí používat data, která nejsou zcela spolehlivá a relevantní.
- Podpora managementu se stává hlavním úkolem informatiky. Operativa je už do značné míry vyřešeným úkolem (neplatí pro zábavu).

Nejčastěji používané tributy kvality dat:
1. *Relevantnost* – míra, do jaké míry data splňují účel, pro který jsou používána, týkají se daného problému.
2. *Přesnost* – jak přesná jsou používaná data (např. směrodatná odchylka). Kupodivu se neuvažují posunutá data
3. *Včasnost* – za jakou dobu lze data aktualizovat, jak jsou data aktuální.
4. *Dostupnost* – jak jsou již existující data dostupná. Obtížný problém díky nesmyslných předpisů pro ochranu privátních dat.
5. *Porovnatelnost* – metrika hodnotící možnost porovnávat, ale také spojovat data z různých zdrojů.
6. *Koherence* – metrika vyjadřuje, do jaké míry byla data vytvořena podle z hlediska výsledku kompatibilních pravidel.
7. *Úplnost* – metrika udávající jaká část potenciálních dat je zachycena v databázi, případně, zda výběr dat pokrývá „rovnoměrně“ celý výběrový prostor.

- Relevantnost a včasnost závisí na frekvenci zjišťování nebo na tom, jak je časově náročné data vytvořit.
- Kvalita dat může implikovat vytvoření datového úložiště v SOA, aby management mohl ovlivňovat chod systému.
- Přesnost SW metrik je malá.

Kdy je vhodné použít datové úložiště?
- Při rozvrhování činností nemůže mít všechna data ve vysoké kvalitě (některá data se nesbírají kvůli ceně sběru, nedostatek dat, nepřesnost dat).
- Algoritmus rozvrhování musí být pomalý (exponenciální složitost, nutné jsou dohody s lidmi) a nepřesný v důsledku nekvality dat → **použít úložiště dat** (nutné, i pokud chceme dát managementu možnost uplatnit své znalosti a intuici při řízení).

Čištění dat. Operace čištění.

- „Zlepšováním“ či čistěním dat (data cleaning) se míní takové operace zlepšování kvality dat jako odstranění okrajových dat, doplňování dat do časových řad, atd.

Čištění dat:
- *Okrajová data* (chyby měření).
  - Jde o postup, kdy se ze souboru vylučují data, která jsou zjevně nesprávná: úmyslně změněná, chybně zanesená (překlepy), nesprávného formátu.
- *Chybějící data*.
  - V tomto případě se do souboru doplní chybějící data, aby bylo možno soubor rozumně zobrazovat (například časové řady) a přitom nedošlo k chybným výsledkům (k významným změnám charakteristik daného souboru). Někdy se doplňují data v řecko-latinských čtvercích.
- *Vyloučení duplicitních dat*.
- *Sjednocení formátů*.