Odhady v softvérových projektoch

- Techniky odhadu
- Náklady na softvérový projekt
  - *Produktivita*
- Odhad rozsahu (veľkosti) softvéru
  - *Dĺžka textu programu*
  - *Funkčné body (angl. function points)*
  - *Body prípadov použitia (angl. use case points)*
- Procesy odhadu nákladov
- Jednoduché modely odhadu nákladov
  - *Verzia 1 COCOMO*
  - *Verzia 2.0 COCOMO*
  - *Úpravy modelu COCOMO – znovupoužitie*
  - *Validácia modelu*
  - *Vyváženie modelu*

Odhady v softvérových projektoch

> „A leading executive was once asked what single characteristic was most important in a project manager. His response: '…a person with the ability to know what will go wrong before it actually does…' We might add: '…and the courage to estimate when the future is cloudy…'" (Pressman, 1992)

Pri plánovaní treba odpovedať na tieto otázky:

- aké úsilie treba vynaložiť na vykonanie určitej činnosti
- koľko času treba na vykonanie činnosti
- aké sú náklady na vykonanie činnosti

Odhad: najpravdepodobnejšia hodnota, ktorú bude nadobúdať určitý ukazovateľ. Odhad môže byť

- *empirický:* analýza údajov s cieľom stanovenia vzťahov medzi jednotlivými charakteristikami
- *analogický:* použitie známych meraní na odhad
- *teoretický:* vytvorenie numerického modelu, ktorý sa overuje spravidla empiricky
- *heuristický:* rozšírenie ostatných metód, použitie expertných systémov

Náklady a zdroje treba poznať na začiatku, ale presne určiť ich potrebu možno až pri vykonávaní projektu, resp. pri jeho ukončení.

Odhad je vždy zaťažený určitou chybou, upresňuje sa s postupom projektu. Kvalitu odhadu možno stanoviť až porovnaním očakávaných a skutočných hodnôt (ak je chyba menšia ako 20% už sa hovorí o dobrom odhade).

Cieľom odhadu je predpoveď najpravdepodobnejšieho budúceho vývoja sledovanej charakteristiky projektu (nákladov, času trvania,…). Splnenie odhadu nie je cieľom projektu.

S odhadom súvisí meranie (priraďovanie čísel alebo symbolov atribútom objektov). Meranie je dôležité pri rozhodovaní a aj pri určení presnosti odhadu. Podrobnejšie pozri kapitolu Meranie v softvérovom projekte.

Proces predpovede a jej vylepšovania musí zahŕňať tieto činnosti:

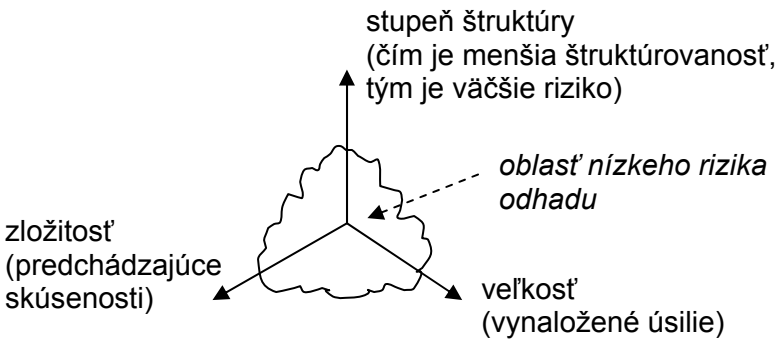
1. Výber charakteristík, ktoré sa budú odhadovať
2. Vykonanie odhadu (predpovede)
3. Vyhodnotenie správnosti odhadu (spätná väzba)

Metódy odhadu v softvérovom projekte:

- *zhora nadol:* možnosť podcenenia „drobných" problémov (tzv. implementačné detaily)
- *zdola nahor:* možnosť podcenenia činností týkajúcich sa celého systému (napr. integrácia, riešenie rizík); je náročnejšia, treba mať už nejaký návrh systému, t.j. rozdelenie do komponentov.

⏩ Odhad nie je hádanie ! (mal by byť podložený faktami)

Charakteristiky projektu, ktoré ovplyvňujú odhad:



Techniky odhadu

Dekompozícia

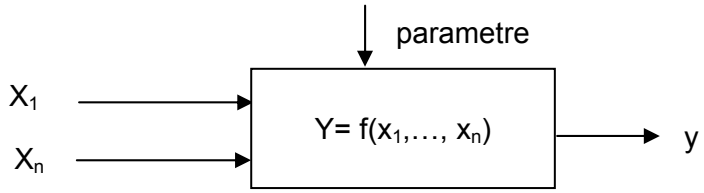
Rozdelenie projektu na jednotlivé etapy.

Modelovanie

Modely:

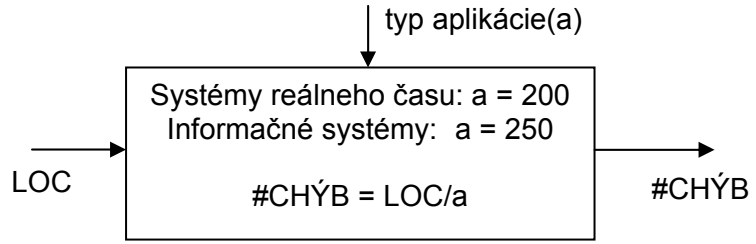
- parametrické modely
- neparametrické modely (napr. metódy strojového učenia, neurónové siete)

Snaha o určenie, ktoré (merateľné) parametre vplývajú na dĺžku trvania činností a odhaliť vzťah medzi týmito parametrami a dĺžkou trvania činností. Modely závisia od použitého procesu, t.j. ak sa zmení proces treba spravidla zmeniť aj model.



Problémom je, že X_1, …X_n v skutočnosti nie sú nezávislé premenné.

Príklad: odhad chýb



Posúdenie expertov

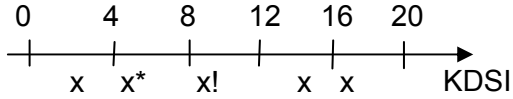
Použitie analógie pri expertnom odhade

Použijú historické údaje z predchádzajúcich ukončených projektov. Často dosť nespoľahlivý postup. Možno zlepšiť použitím údajov zo skutočne podobného projektu a zabezpečením, aby jednotlivci vypracúvajúci odhad boli experti v danej oblasti. Postupuje sa spravidla zhora nadol.

Použitie DELPHI metódy pri expertnom odhade nákladov

1. Experti dostanú špecifikáciu softvéru a formulár odhadu
2. *Diskusia* o produkte a metódach odhadu
3. Vytvorenie individuálnych odhadov
4. Zápis odhadov do tabuľky a vrátenie expertom (identifikujú sa iba expertove osobné odhady)
5. Distribuovanie odhadov a *diskusia*
6. Prehodnotenie odhadov - cyklus pokračuje bodom 3, kým nedôjde k dohode

Odhad veľkosti vytváraného systému
Projekt: bankový softvérový systém
Expert: MC
Kolo: 1
Dátum: 2.8. 1998



Kľúč: x … odhad
x* … váš odhad
x! … stredný odhad

Možnosti dodávateľa

Parkinsonov zákon: práca sa vždy dá naplánovať tak, aby vyplnila všetok čas, ktorý je k dispozícii. Náklady sa určia pomocou dostupných zdrojov a nie objektívneho odhadu.

⏩ Neexistuje najlepšia alebo najhoršia technika.
Používa sa viac techník súčasne.

Náklady na softvérový projekt

- náklady na softvér (programovacie prostredia, CASE,…), hardvér (prípadne ďalšie zariadenia) vrátane údržby
- náklady na cestovanie a školenia
- náklady spojené s vynaloženým úsilím na vytvorenie výrobku alebo zabezpečenie služby
  - výplaty softvérovým inžinierom
  - réžia: prevádzka priestorov, podporný personál, komunikácia v rámci projektu, doplnkové služby (knižnica, kopírovacie služby, …), zákonné povinnosti (poistenie,…)

Réžia môže predstavovať až dvojnásobok platu softvérového inžiniera.

Na náklady vplýva:

- veľkosť (zložitosť) projektu (vytváraného výrobku alebo služby)
- schopnosti členov tímu a skúsenosti v problémovej oblasti
- použitie podporných prostriedkov
- proces vývoja
- dané ohraničenia na výrobok (napr. spoľahlivosť, rýchlosť, pamäťové nároky a iné charakteristiky výpočtového prostredia)

Treba odlišovať náklady a cenu výrobku pre zákazníka.

⏩ **Cena = náklady + zisk**

Cena často závisí od strategického zámeru organizácie. Pri jej určovaní treba uvažovať širšie súvislosti.

Na cenu vplýva:

- *vytvorenie príležitosti na trhu* (ak sa presúva na nový trh, často je nižšia cena)
- *podmienky zmluvy* (ak napr. dodávateľovi ostanú zdrojové texty programov a bude ich môcť použiť na iný projekt, stanoví sa nižšia cena)
- *stupeň neurčitosti odhadu nákladov* (často zvýšenie ceny v prípade neistoty)
- *pravdepodobnosť premenlivosti požiadaviek* (iná cena sa môže stanoviť vtedy, ak sa počíta so zmenami a sú v nej zahrnuté ako keď sa cena zmien stanovuje zvlášť)
- *finančná situácia dodávateľa a strategické rozhodnutia* (niekedy sa stanovuje nižšia cena, dokonca nižšia ako sú samotné náklady, aby sa spoločnosť udržala na trhu, cenový dumping, konkurencia)
- *možnosti zákazníka* (cena sa určí podľa rozpočtu zákazníka a potom sa rokuje o špecifikácii, zahrnie sa toľko funkčnosti, aby sa dodržali plánované náklady)

Produktivita

Náklady, čas potrebný na projekt a vynaložené úsilie (prácnosť) významne ovplyvňuje produktivita.

⏩ **Produktivita označuje množstvo *výstupu* na jednotku času.**

Možno ju merať ako:

- veľkosť výstupu (napr. počet riadkov textu programu) za mesiac
- množstvo funkčnosti (funkčné body) za mesiac

Produktivita a kvalita:

- Som produktívny, ak vyprodukujem veľa zlého softvéru?
- Produktivita nie je to isté ako kvalita. Treba však zvážiť celý životný cyklus softvéru – bez testovania by sme boli vcelku produktívni.

Produktivitu nemožno merať priamo. Treba sa vyhnúť zjednodušeniu odhadu a merania produktivity. Na produktivitu jednotlivých inžinierov vplýva množstvo faktorov:

- *ľudia:* znalosti a skúsenosti využiteľné v projekte, pracovné prostredie
- *proces:* kvalita procesu vývoja softvéru (formálne prehliadky, použitý model životného cyklu)

- *požadované vlastnosti výrobku:* spoľahlivosť a výkonnosť výpočtového prostredia
- *problém:* veľkosť projektu, náročnosť problematiky
- *zdroje:* podpora technológie (CASE)

Rozdiely v produktivite softvérových inžinierov môžu byť až 10 násobné. Menej produktívni softvéroví inžinieri však môžu vytvárať spoľahlivejšie programy, ktoré sa ľahšie udržujú a netreba ich toľko testovať.

⏩ Pozor na porovnávanie jabĺk s hruškami.
Produktivita má viac povahu manažmentu ako technickú.

Vždy možno zvýšiť počet riadkov programu.

Odhad rozsahu (veľkosti) softvéru

Rozsah výrobku (softvéru) priamo ovplyvňuje ďalšie atribúty ako napr. náklady, produktivitu, zložitosť, testovateľnosť, atď.

Zistilo sa, že manažéri sú lepší v odhade vyžadovaného úsilia (prácnosti) ako v rozsahu výrobku (napr. dĺžke programu).

Dĺžka textu programu

Je to najjednoduchšia metrika rozsahu výrobku (programu). Sú rôzne prístupy pre meranie dĺžky textu programu. Určitý spôsob sa spravidla viaže na jednu organizáciu a jeden programovací jazyk. Na odhad sa používa najmä analógia a posúdenie expertov.

Najčastejšie sa používajú tieto spôsoby vyjadrenia dĺžky programu:

- počet oddeľovačov (;)
- počet dodaných príkazov (DSI – Delivered Source Instructions)
- počet vytvorených riadkov, komentáre a prázdne riadky sa nepočítajú (LOC – Lines of Code)
- počet znakov v texte programu
- počet bajtov potrebných na uchovanie programu
- počet strán výpisu programu

Výhody

- ✓ jednoduchosť a široká použiteľnosť
- ✓ možnosť použitia historických údajov (lebo existujú)
- ✓ možnosť automatického sledovania

Nevýhody

- × ťažkosti pri sledovaní rôznych jazykov
- × nepostihuje akosť a môže závisieť od štýlu zápisu.

Funkčné body (angl. function points)

Allan Albrecht, 1979

Vychádza zo špecifikácie požiadaviek (pre aplikácie komunikujúce s používateľom) a určuje počet a rozsah používateľovi prístupných služieb systému (tieto aspekty možno stanoviť dosť presne už na začiatku projektu).

Používa sa na odhad rozsahu softvérového systému, najmä pre interaktívne databázovo-orientované systémy (napr. obchodné aplikácie, angl. business applications). Nehodí sa pre systémy so zložitým vnútorným spracovaním.

Technika funkčných bodov nezávisí od programovacieho jazyka.

Postup:

1. Analýza požiadaviek a určenie týchto charakteristík:
   A. *počet externých vstupov* (vstupy od používateľov, výber z menu)
   B. *počet externých výstupov* (výstupné zostavy, výsledky, ktoré sa oznamujú používateľovi; počíta sa celá obrazovka a nie jednotlivé položky)

C. *počet logických interných súborov* (logické zoskupenia údajov, databáz (resp. db tabuliek), ktoré používa program, napr. zoznam používateľov)

D. *počet externých rozhraní* (objekty v programe, ktoré program používa na komunikáciu s okolím; napr. aj konfiguračné súbory)

E. *počet externých dopytov* (napr. reakcie na externé podnety, ktoré vyžadujú transakcie; hľadanie v texte; dopyty do databázy)

2. Klasifikácia charakteristík podľa zložitosti (3 úrovne: jednoduchá, stredná, zložitá). Na určenie zložitosti sa definujú ďalšie pomôcky, sleduje sa napr. frekvencia prístupov do súborov, typy záznamov.

3. Výpočet neupravených funkčných bodov – UFC (Unadjusted Function Count)

$$UFC = \sum_{i=1}^{5} \sum_{j=1}^{3} w_{ij} c_{ij}$$

i – typ charakteristiky (A, B, C, D, E)
j – zložitosť (jednoduchá, stredná, zložitá)
c_{ij} – počet (odhadnutá, nameraná hodnota)
w_{ij} – váha z tabuľky

	jednoduchá	stredná	zložitá
A	3	4	6
B	4	5	7
C	7	10	15
D	5	7	10
E	3	4	6

4. Úprava počtu funkčných bodov na základe 14 rôznych (technických) charakteristík projektu (pridelí sa v_i v rozsahu 0-5 bodov) a výpočet hodnoty faktora zložitosti – CAF (Complexity Adjustment Factor).

$$CAF = 0.65 + \left(0.01 * \sum_{i=1}^{14} v_i \right)$$

Ako charakteristiky projektu sa uvažujú:

1. Vyžaduje sa dátová komunikácia? (batch mód, nepriamy vstup údajov, interaktívny mód)
2. Sú v systéme distribuované funkcie?
3. Je výkonnosť systému kritická? (odozva a priepustnosť systému)
4. Vyžaduje systém spoľahlivé zálohovanie a obnovu?
5. Bude sa program vykonávať v existujúcom a používanom operačnom prostredí?
6. Vyžaduje systém on-line vstup údajov?
7. Vyžaduje on-line vstup údajov zložité transakcie (viaceré obrazovky a operácie)?
8. Umožňuje sa on-line modifikácia údajov?
9. Je vnútorné spracovanie zložité?
10. Sú moduly navrhnuté pre znovupoužitie?
11. Bude treba viacnásobnú inštaláciu v rôznych organizáciách?
12. Uvažuje sa jednoduchosť inštalácie?
13. Uvažuje sa jednoduchosť použitia?
14. Uvažuje sa zapracovanie budúcich zmien?

5. Výpočet DFP (Delivered Function Points)

$$DFP = CAF * UFC$$

Niekedy sa dopĺňajú aj pesimistické a optimistické odhady. Hodnota vypočítaná technikou funkčných bodov predstavuje najpravdepodobnejšiu hodnotu (*m*), pesimistický a optimistický odhad sa vykonáva najčastejšie na základe údajov z predchádzajúcich projektov.

Function Points II – funkčné body sa priraďujú na základe

$$E = \frac{p + 4 * m + o}{6}$$

vykonávaných transakcií v systéme; doplnenie ďalších charakteristík aj pre iné druhy programov.

Feature Points (1986) – upravuje techniku funkčných bodov na použitie pre systémy s vnútorným zložitým spracovaním (operačné systémy, vnorené aplikácie, atď.). Pridáva šiestu charakteristiku – *algoritmy* so štandardnou váhou 3. Tiež redukuje váhu pre charakteristiku *počet logických interných súborov* z 10 na 7.

Pre aplikácie, kde počet algoritmov a počet logických interných súborov je rovnaký, táto technika dáva rovnaké výsledky ako technika funkčných bodov.

Výhody techniky funkčných bodov:

- ✓ možnosť odvodenia priamo zo špecifikácie
- ✓ nezávislosť na programovacom jazyku a technológii (používajú sa tabuľky prevodu funkčných bodov na riadky textu zdrojového programu pre jednotlivé programovacie jazyky)

Nevýhody techniky funkčných bodov:

- × dosť sa spolieha na používateľské posudzovanie rozsahu (nemusí vždy zodpovedať skutočnosti)
- × ťažkosti a subjektívnosť pri určovaní charakteristík a ich stupňa zložitosti
- × nedá sa automatizovať
- × pri opätovnom používaní softvérových súčiastok nevyjadrí skutočnú produktivitu, resp. náklady

⏩ Pri porovnávaní viacerých programovacích jazykov treba brať do úvahy skutočnosť, že hoci počet riadkov sa môže zredukovať pri použití vyššieho programovacieho jazyka napr. na polovicu, celkové úsilie sa môže zmenšiť iba o niekoľko percent.

Empiricky vytvorená tabuľka prevodu funkčných bodov na riadky textu programu (Capers Jones, Software Productivity Research, Inc., 1996):

Jazyk	**Úroveň**	**LOC/FP**
Asembler	1	320
Ada 95	6.5	49
Arity Prolog	5	64
C	2.5	128
C++	6	53
CLOS	15	21
Cobol	3	107
DBase IV	9	36
Delphi	11	29
Eiffel	15	21
Excel 1-2	51	6
Excel 5	57	6
Flex	7	46
Fortran 95	3	71
G2	6.5	49
HTML 3.0	22	15
Ingres, Informix	8	40
Java	6	53
Lisp	5	64
Modula2	4	80
MAKE	15	21

Pascal	3.5	91
Perl	15	21
prirodzený jazyk	0.1	3200
Smalltalk	15	21
SQL	25	13
Tabuľkové procesory	50	6
Visual Basic 5	11	29
Visual Age	15	21
Visual C++	9.5	34

Orientačný vzťah medzi úrovňou jazyka a produktivitou:

Úroveň jazyka	**Priemerná produktivita za mesiac**
1 – 3	5 až 10 funkčných bodov
4 – 8	10 až 20 funkčných bodov
9 – 15	16 až 23 funkčných bodov
16 – 23	20 až 30 funkčných bodov
24 – 55	30 až 50 funkčných bodov
viac ako 55	40 až 100 funkčných bodov

Body prípadov použitia (angl. use case points)

Gustav Karner, 1993

Rozšírenie techniky funkčných bodov na odhad rozsahu softvérového projektu.

Technika nezávisí od programovacieho jazyka.

Postup:

4. Klasifikácia hráčov podľa zložitosti (3 úrovne: jednoduchá, stredná a zložitá). Jednoduchá – iný systém s definovaním API, stredná – iný systém komunikujúci prostredníctvom protokolu, zložitá – človek komunikujúci pomocou GUI alebo webového rozhrania

Typ hráča	Váha
Jednoduchý	1
Stredný	2
Zložitý	3

5. Klasifikácia prípadov použitia podľa zložitosti (3 úrovne: jednoduchá, stredná, zložitá) podľa počtu transakcií v opise prípadu použitia vrátane bočných či rozširujúcich scenárov.

Typ prípadu použitia	Počet transakcií	Váha
Jednoduchý	≤ 3	1
Stredný	4 do 7	2
Zložitý	≥ 7	3

Iné spôsoby zisťovania zložitosti prípadu použitia:

- Počet analytických tried, ktoré prípad použitia implementujú
- Odlíšenie na základe expertného odhadu (jednoduchý = jednoduchý algoritmus, jeden používateľ, jednoduché rozhranie a pracuje len s jednou entitou v databáze)

6. Výpočet neupravených bodov prípadov použitia – UUCP (Unadjusted USe Case Points)

$$UUCP = \sum_{i=1}^{3} w_i a_i + \sum_{j=1}^{3} w_j uc_j$$

UUCP = UAW + UUCW
UAW ... unadjusted actor weights
UUCW ... unadjusted use case weights

6. Úprava na základe charakteristík projektu (pridelí sa v_i v rozsahu 0-5 bodov: 0 žiadny vplyv, 3 priemer, 5 významný vplyv) a výpočet hodnoty faktora zložitosti
   – TCF (Technical Complexity Factor)
   – EF (Environmental Factor)

$$TCF = 0.6 + \left(0.01 * \sum_{i=1}^{13} v_i w_i \right) \qquad EF = 1.4 + \left(-0.03 * \sum_{i=1}^{8} v_i w_i \right)$$

$$UCP = UUCP * TCF * EF$$

Výhody

- možnosť automatizácie procesu
- možnosť stanovenia priemeru času implementácie na jeden prípad použitia v rámci organizácie
- dobré oddelenie metriky rozsahu od veľkosti tímu

Nevýhody

- potrebujeme mať vytvorené prípady použitia (môže to byť až 10-20% úsilia v projekte)
- veľké jednotky pre podrobné plánovanie
- rôzne odhady zložitosti prípadov použitia (čo sa počíta za transakciu)
- niektoré z faktorov neovplyvňujú celý projekt a teda vnášajú do odhadu chybu

Procesy odhadu nákladov

Bailey and Bassili

Boehm – zahŕňa aj charakterizáciu projektu a explicitné stanovenie cieľov odhadu

DeMacro (zdola nahor)

Heemstra – určuje presne čo treba odhadovať (veľkosť, úsilie a produktivita)

Humprey (metriky a modely založené na osobných údajoch, hodí sa na zlepšenie práce jednotlivcov)

Jednoduché modely odhadu nákladov

Cieľom modelu odhadu nákladov je určiť, ktoré parametre významne vplývajú na výšku nákladov a odhaliť vzťah medzi nákladmi a týmito charakteristikami.

Vychádza sa z odhadu vynaloženého úsilia –

$p * V$ [človeko-mesiac]

p – koeficient produktivity (1/produktivita)
V – veľkosť systému

$$úsilie = C * VM^b * M$$

Doplnenie exponenciálneho komponentu (náklady sa nevyvíjajú lineárne s veľkosťou projektu – väčšie tímy, viac komunikácie):

C	zložitosť
VM	metrika výrobku (často napr. veľkosť, lebo odhad veľkosti je jednoduchší ako odhad úsilia)
b	odráža nelinearitu úsilia vzhľadom na veľkosť projektu, je to číslo blízke 1
M	koeficient, ktorý kombinuje proces, výrobok a atribúty vývoja (napr. aj maximálny počet pracovníkov)

- problémy sú s ťažkosťami s odhadom *VM* na začiatku projektu;
- odhad *c* a *M* je subjektívny aj napriek tomu, že existujú rôzne klasifikácie charakteristík spolu s pravidlami priradenia hodnôt
- príklad odhadu získaného na základe analýzy 60 projektov v IBM

$$úsilie = 5.25 * (KDSI)^{0.91}$$

COCOMO (COnstructive COst MOdel)

Boehm, 1. verzia r. 1981; 2. Verzia r. 1996

Verzia 1 COCOMO

Vychádza z odhadu veľkosti programu v KDSI.

Má tri úrovne podľa toho, koľko podrobností o projekte máme k dispozícii:

- základná (hrubé odhady),
- stredná (berie do úvahy aj atribúty projektu),
- podrobná (rozdelenie úsilia medzi jednotlivé etapy)

COCOMO definuje potrebné úsilie a čas. Čas a úsilie nemožno zamieňať. Pridaním ľudí sa môže zmenšiť úsilie/človekomesiace, ale čas môže dokonca vzrásť.

Vychádza zo vzťahu

$$úsilie = c * VM^{\,b} * M$$

Uvažuje 3 módy vyjadrujúce stupeň zložitosti systému (C):

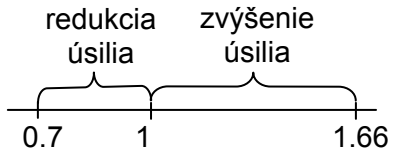
- *organický mód:* relatívne malý softvérový tím pracuje na aplikácii zo známej problémovej oblasti, v známom prostredí. Predpokladajú sa malé náklady na komunikáciu, stabilný hardvér, známe algoritmy a možnosť dekompozície problému na menšie ucelené časti (veľkosti do 50 KDSI).
- *prechodný mód:* predstavuje prechod medzi „jednoduchým“ organickým módom a „zložitým“ viazaným módom. Predpokladajú sa vyššie požiadavky na komunikáciu, čas a veľkosť programu (jeho ucelených častí). Podľa Boehma cca 300 KDSI.
- *viazaný mód:* rôzne ohraničenia projektu, napr. krátke termíny dodávky, neustále zmeny v požiadavkách, zásahy zákazníka, neznáma problémová oblasť, atď. Aplikácie majú zvyčajne stanovené požiadavky na časovú odozvu, často ide o interaktívne, riadiace a operačné systémy.

1. základná úroveň COCOMO

M = 1 $úsilie = 3.2 * (KDSI)^{1.05}$ – organický mód

$úsilie = 3.0 * (KDSI)^{1.12}$ – prechodný mód

$úsilie = 2.8 * (KDSI)^{1.20}$ – viazaný mód



2. stredná úroveň COCOMO

$M = K_1 * K_2 * \quad * K_{15}$

- K_i reprezentujú hodnoty koeficientov, ktoré upresňujú charakteristiky projektu, interval (0.7, 1.66),
  číslo sa priraďuje podľa danej tabuľky, kde pre každý koeficient je priradená hodnota pre každú z 5 kategórií (veľmi nízky, nízky, normálny, vysoký, veľmi vysoký)

Atribúty výrobku:

- požadovaná spoľahlivosť
- rozsah databázy
- zložitosť výrobku

Atribúty počítačového vybavenia:
- ohraničenia na čas vykonávania (využitie dostupného strojového času)
- ohraničenia na pamäť
- požadovaný čas odozvy
- stabilita hardvérovej/softvérovej platformy (frekvencia zmien)

Atribúty personálu:
- schopnosti analytikov
- skúsenosti s aplikačnou oblasťou
- schopnosti a skúsenosti programátorov
- skúsenosti s operačným systémom
- skúsenosti s programovacím jazykom

Atribúty projektu:
- použitie moderných programovacích techník
- použitie podporných softvérových prostriedkov
- ohraničenia na plán vývoja

Hodnoty atribútov možno stanoviť regresnou analýzou na základe vlastných údajov z predchádzajúcich projektov. Niekedy sa nehodnotia všetky atribúty alebo sa pridávajú nové.

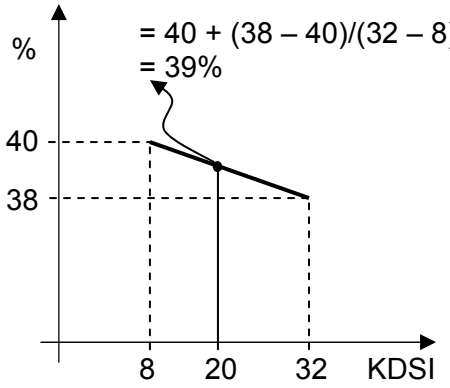
3. podrobná úroveň COCOMO

Rozpracúva úroveň jednotlivých **etáp projektu**. Nezahŕňa špecifikáciu, lebo predpokladá, že takýto podrobný odhad sa robí až po špecifikácii. Vychádza z tabuľky pre rozdelenie úsilia (v %) medzi jednotlivé etapy podľa veľkosti projektu (pre každú zložitosť projektu iná).

Tabuľka pre organický mód:

ORGANICKÉ PROJEKTY	2 KDSI	8 KDSI	32 KDSI	128 KDSI
Architektúra systému	16%	16%	16%	16%
Podrobný návrh	26%	25%	24%	23%
Implementácia a testovanie modulov	42%	40%	38%	36%
Integrácia a testovanie systému	16%	19%	22%	25%

Pre inú veľkosť systému sa percentuálny podiel vypočíta interpoláciou údajov z tabuľky.



Odhad času podľa COCOMO

$$\textit{čas vývoja} = 2{,}5 * \textit{úsilie}^{d}$$

d = 0.38 pre organický mód
d = 0.35 pre prechodný mód
d = 0.32 pre viazaný mód

Verzia 2.0 COCOMO

Berie do úvahy zmenené podmienky pri tvorbe softvéru. Napr.

- uvažuje rôzne modely životného cyklu softvéru
- rôzne prístupy k vývoju softvéru

Výsledkom je nové delenie úrovní COCOMO. Uvažujú sa tri etapy vývoja (a aj odhadu): analýza aplikácie, skorý návrh a etapa postarchitektonického návrhu. Ako základ sa neberie KDSI, ale veľkosť sa odhaduje v prvej etape počtom objektov (object points), v druhej etape počtom funkčných bodov a až v tretej etape riadkami textu programu.

Model sa stále vyvíja, konkrétne tvary vzťahov a koeficientov sa menia. Podrobnejšie informácie možno nájsť na webe napríklad:
http://sunset.usc.edu/COCOMOII/Cocomo.html

alebo v podobe ilustratívnej implementovanej „COCOMO web-kalkulačky“

http://csse.usc.edu/tools/COCOMOII.php.

Úpravy modelu COCOMO – znovupoužitie

N_1 … KDSI vyvinutého jedinečného kódu
N_2 … KDSI vyvinutého kódu na znovupoužitie
N_3 … KDSI znovupoužitého kódu (black-box reuse)
N_4 … KDSI modifikovaných súčiastok (white-box reuse),
a_i … zložitosť pre jednotlivé typy
b … ako v neupravovanom COCOMO

Znovupoužitie softvérovej súčiastky sa počíta v projekte iba raz aj pri viacnásobnom použití. Počítajú sa len lokálne (vnútropodnikové knižnice).

- **Relative Cost of Reuse**

- tá časť z nákladov potrebných na vývoj novej súčiastky, ktorú treba na znovupoužitie podobnej súčiastky bez modifikácie (black box reuse)
- v rôznych štúdiách sa RCR pohybuje od 0.05 do 0.4. Najčastejšie sa uvádza, že opätovné použitie softvéru zaberie asi 20% úsilia nového vývoja, t.j. RCR = 0.2

- **Relative Cost of Writing for Reuse**

- tá časť z nákladov potrebných na vývoj novej súčiastky, ktorú treba na vytvorenie podobnej znovupoužiteľnej súčiastky (treba viac testovať, zovšeobecniť súčiastku, vypracovať podrobnejšiu dokumentáciu, pripraviť distribúciu súčiastky)
- vytvorenie softvéru pre znovupoužitie zaberie asi 50% úsilia na vývoj, t.j. RCWR=1.5

Príklad IBM: 25% – ďalšie zovšeobecnenie

$$\textit{úsilie} = a_1 N_1^b + a_2 N_2^b + a_3 N_3^b + a_4 N_4^b$$

15% – ďalšia dokumentácia
15% – ďalšie testovanie
5% – knižničná podpora a údržba

60% ceny navyše, t.j. RWCR = 1.6

V zjednodušenom modeli sa uvažuje RCR = 0.1 a RWCR = 2.0, t.j. 20 krát viac nákladov treba na vytvorenie softvéru pre znovupoužitie ako na jeho znovupoužitie.

Zjednodušenie:

$$úsilie = aN_1^b + 20acN_2^b + acN$$

a,b … ako COCOMO
c … vzťah medzi úsilím vyvinúť jedinečný kód a úsilím znovupoužiť kód, číslo z intervalu (0.0909, 1.739)

Validácia modelu

- meranie chýb rozdielom medzi predpovedanou hodnotou a nameranou (skutočnou) hodnotou; neberie sa do úvahy veľkosť projektu, čo môže spôsobovať problémy
- výpočet chyby percentami

$$100/n \sum_{i=1}^{i=n} \frac{E_{pred} - E_{act}}{E_{act}}$$

  - problém je, že niektoré chyby sa kompenzujú (vysoký odhad sa kompenzuje nízkym), preto sa počítajú absolútne hodnoty

$$100/n \sum_{i=1}^{i=n} \frac{| E_{pred} - E_{act} |}{E_{act}}$$

- správnosť predpovede sa udáva stanovením percenta odhadov, ktoré spadajú spĺňajú stanovenú percentuálnu odchýlku

Vyváženie modelu

- ide o vyladenie modelu pre lokálne podmienky (organizácie, softvérových inžinierov, projekt)
- robí sa výberom hodnôt pre koeficienty modelov
- podľa údajov z predchádzajúcich projektov: porovnanie modelu odhadu nákladov a skutočných hodnôt, predpokladá sa určitá minimálna úroveň vyspelosti organizácie, t.j. procesy sú opakovateľné