

26 A SZOFTVER KÖLTSÉGEINEK BECSLÉSE

TÉMAKÖRÖK

A fejezet célja, hogy bevezessen a költségek és a szoftvergyártáshoz szükséges munka becslésének technikáiba. A fejezet elolvasása után:

- megértjük a szoftverköltségek árazásának alapelemeit, valamint azt, hogy a szoftver ára nincs közvetlen kapcsolatban a fejlesztési költségekkel;
- megismerünk három, a szoftvertermelékenység becslésére használt metrikát;
- meglátjuk, hogy a szoftverköltség és az ütemezés becslésére több különböző technika is alkalmazható;
- megértjük az algoritmikus költségbecslésre használt COCOMO II modell alapelveit.

TARTALOM

- 26.1. A termelékenység
- 26.2. Becslési technikák
- 26.3. Az algoritmikus költségmodellezés
- 26.4. A projekt időtartama és résztvevői

Az 5. fejezetben bemutatott projekttervezés folyamata során a projektet számos, párhuzamosan vagy szekvenciálisan végrehajtható tevékenységre osztják fel. A projekttervezés korábbi tárgyalásánál a fő szempontok azok voltak, hogy hogyan lehet ábrázolni ezeket a tevékenységeket és függőségeiket, és hogyan lehet szétosztani ezeket a tevékenységeket az emberek között. Ebben a fejezetben a projekt azonosított tevékenységei, valamint a munka- és időbecslés kapcsolatának problémájával foglalkozom. A projektvezetőknek az alábbi kérdésekre kapott válaszokat kell mérlegelniük:

1. Mekkora munkát igényel a tevékenység elvégzése?
2. Hány naptári napra van szükség a tevékenység elvégzéséhez?
3. Mekkora a tevékenység teljes költsége?

A projekt költségbecslését és a projektütemezést normális esetben együtt végzik el. A fejlesztés költségénél elsődlegesek a munkaráfordítások, így a munkaráfordítások kiszámítása felhasználható mind a költség, mind az ütemezés becslésénél. De már a részletes ütemtervek elkészülte előtt is szükség van valamilyen költségbecslésre, amelyet a projekt költségvetésének felállításánál vagy a szoftver eladási árának megállapításánál használnak fel.

A szoftverfejlesztési projekt összköltsége három paramétertől függ:

- a hardver és a szoftver költsége, beleértve a karbantartás költségét is;
- az utazási és a képzési költség;
- a munkaköltség (a szoftvermérnökök bérezésének költsége).

A legtöbb projektben a költségek túlnyomó részét a munkaköltség teszi ki. Viszonylag alacsony azoknak a számítógépeknek az ára, amelyek teljesítménye megfelelő a szoftverfejlesztéshez, és bár a különböző helyszíneken fejlesztett projekteknél az utazási költség jelentős lehet, a legtöbb projekteknél ez a költség aránylag nem magas, ráadásul elektronikus levelezéssel, megosztott weblapokkal és videokonferenciákkal csökkenthető az utazások száma. Az elektronikus konferencia csökkenti az utazási időt, és ez az idő produktívan felhasználható a fejlesztésben. Egy projektben, ahol dolgoztam, az elektronikus konferencia alkalmazása a személyes találkozó helyett több mint 50 százalékkal csökkentette az utazási költséget és időt.

A munkaköltség nem egyszerűen a projekten dolgozó szoftvermérnökök fizetésének költségét jelenti. A szervezetek a munkaköltséget az általános költség alapján számítják ki, azaz a szervezet fenntartásának teljes költségét elosztják a termelő munkaerő számával. Ezek alapján a következő költségek mind részét képezik a teljes munkaköltségnek:

1. az irodai helyiség fenntartásának, fűtésének és világításának költsége;
2. a kisegítő személyzet (a könyvelők, a titkárság, a takarítók és a technikusok) költsége;
3. a hálózathasználat és a kommunikáció költsége;
4. a központi szórakozási lehetőségek (könyvtár, pihenési lehetőségek stb.) költsége;
5. a társadalombiztosítási költségek (nyugdíj- és betegbiztosítás).

Az általános költségtenyező jellemzően a szoftvermérnökök fizetésének kétszerese körül van, de függ a szervezet méretétől és a szervezethez kapcsolódó általános költségektől. Ezért ha egy szoftvermérnök 90 000 dollárt keres egy évben, akkor a szervezet teljes költsége évi 180 000 dollár vagy havi 15 000 dollár.

A szoftverek költségelemzését objektíven kell elvégezni. A költségelemzés célja az, hogy pontosan jelezze előre a szállítónak a szoftverfejlesztés költségét. Ha a projekt költsége a vevőnek tett árajánlat része, akkor döntenie kell arról, milyen árat közöljenek a vásárlóval. A klasszikus eset szerint az ár egyszerűen a költség és a profit összege, de a projekt költsége és a vásárlónak tett ár közötti kapcsolat általában nem ilyen egyszerű.

A projekt működése közben a projektvezetőnek folyamatosan felül kell vizsgálnia a betervezett árat és ütemezést. Ez segít a folyamat tervezésében és az erőforrások hatékony felhasználásában. Ha az aktuális felhasználás lényegesen nagyobb a becslétnél, akkor a projektvezetőnek közbe kell lépnie. Ez jelentheti további erőforrások bevonását a projektbe vagy a munka módosítását.

A szoftver árazásánál figyelembe kell venni a szélesebb szervezeti, gazdasági, politikai és üzleti megfontolásokat, ezért a szoftver eladási ára és a fejlesztési költségek között a kapcsolat nem egyszerű. A figyelembe vehető tényezőket a 26.1. ábrán foglaltam össze. Mivel a szervezeti megfontolásokat is számításba vesszük, a projekt árképzésébe a szervezet felsőbb vezetését (vagyis a stratégiai döntéshozókat) általában ugyanúgy bevonják, mint a szoftverprojekt vezetőit.

Tényező	Leírás
Piaci lehetőségek	Egy fejlesztéssel foglalkozó szervezet alacsony árat állapíthat meg, ha a szoftverpiac új szegmensébe szeretne belépni. Ha vállalják, hogy a projekt alacsony profitot hozzon, akkor később magasabb profitra is szert tehetnek. Az így nyert tapasztalatok új termékek fejlesztését segíthetik elő.
Költségbecslési bizonytalanság	Ha egy szervezet nem biztos a költségbecslésében, akkor előre nem látható kiadások miatt megnövelheti az árat valamivel a normál profit fölé.
Szerződésbe foglalt feltételek	A vásárló megteheti, hogy engedélyezi a fejlesztőnek a forráskód tulajdonjogának megtartását és más projekteken való felhasználását. Ekkor a felszámított ár alacsonyabb lehet, mint abban az esetben, ha a szoftver forráskódját is átadták volna a vásárlónak.
Követelmények változékonysága	Ha nagy a valószínűsége a követelmények változásának, akkor a szervezet, szerződésekhez való hozzájutás reményében, csökkentheti az árait. A szerződés megszerzése után a megváltoztatott követelményekért magas árat lehet felszámolni.
Pénzügyi állapot	A pénzügyi nehézségekkel küzdő fejlesztők alacsonyabb árat ajánlhatnak, hogy így jussanak szerződésekhez. Még mindig jobb az átlagosnál kevesebb profitot termelni, vagy nem ráfizetésesnek lenni, mint kiszállni az üzletből.

26.1. ábra. A szoftver árat befolyásoló tényezők

Például egy kisméretű, olajcégeknek szolgáltatásokat fejlesztő szoftvervállalat az év elején 10 mérnököt foglalkoztat, de folyamatban csak 5 fejlesztőt igénylő szerződése van. Azonban árajánlatot tehet egy nagy olajcégnek olyan szerződésre, amely két éven keresztül 30 emberévnyi munkát igényel. A projekt legfeljebb 12 hónap múlva indul el, de ha megnyerik, rendbe hozhatja a vállalat pénzügyeit. A vállalat olyan ajánlatot tesz, amely 6 embert igényel 10 hónapra. Az árat (beleértve a projekt járulékos költségeit is) 1,2 millió dollárra becsülik, de figyelembe véve a versenyhelyzetet, a vállalat 0,8 millió dolláros árajánlatot tesz az ügyfélnek. Ez ugyan azt jelenti, hogy ezen a szerződésen veszít, de a specialisták megmaradnak későbbi, jobban jövedelmező projektekre.

26.1. A TERMELÉKENYSÉG

Egy ipari rendszerben a termelékenység a legyártott egységek számának és a gyártásukhoz szükséges emberórák számának hányadosával mérhető. A szoftvereknél viszont több különböző, más-más jellemzőkkel rendelkező megoldás létezik, amelyek közül az egyik az, hogy hatékonyabban hajtható végre, a másik viszont az, hogy olvashatóbb és könnyebben karbantartható. A különböző jellemzőkkel rendelkező megoldásoknál a termelékenységi arányok összehasonlítása nem mond igazán sokat.

Mégis előfordulhat, hogy a vezetőknek meg kell becsülniük a szoftverfejlesztési folyamatban dolgozó mérnökök termelékenységét. A termelékenységi becslésekre a projektbecsléseknél és a folyamat- vagy technológiajavítás hatékonyságának megállapításához lehet szükség.

A becslések alapja általában az, hogy mérik a szoftver valamilyen jellemzőjét, és ezt osztják a fejlesztéshez szükséges teljes munkával. A méréseknek két típusát szokták használni, ezek a következők:

1. *Mérethez kapcsolódó mérések.* Valamilyen tevékenységből származó kimenet méretéhez kapcsolódnak. Legegyszerűbb fajtája az, amikor a leszállított forráskód sorait számlálják meg, de idetartozik az átadott tárgykód utasításainak vagy a rendszer-dokumentáció oldalainak a megszámlálása is.
2. *Funkcióhoz kapcsolódó mérések.* Az átadott szoftver általános funkcionalitáshoz kapcsolódnak, a termelékenységet az adott idő alatt előállított hasznos funkcionalitás mennyiségével fejezik ki. Az ilyen típusú mérések közül a legismertebbek a funkciópont- és az objektumpontmérések.

A termelékenység mérésének egyik széles körben alkalmazott mértéke az egy programozóhónap alatt írt forráskódsorok száma. Ezt úgy számítják ki, hogy az átadott forráskód összes sorainak számát (LOC) elosztják a projekt befejezéséhez szükséges, programozóhónapban megadott teljes idővel, ezért ez az idő magában foglalja az elemzésre, tervezésre, kódolásra, tesztelésre és dokumentálásra szükséges időt is.

Ezt a módszert akkor fejlesztették ki, amikor a legtöbb program FORTRAN, assembly vagy COBOL nyelven készült. A programokat kártyákra írták, és minden kártyára egyetlen utasítás került, ezért könnyű volt kiszámolni a kód sorainak számát, mert az megegyezett a program kártyacsomagjában a kártyák számával. Ezzel szemben az olyan nyelveken megírt programok, mint a Java vagy a C++, tartalmazznak deklarációkat, végrehajtható utasításokat és megjegyzéseket és esetleg makróutasításokat is, amelyek számos sorral megnövelik a kódot. Mivel egy sorban több utasítás is megengedett, ezért a program utasításai és a forráslista sorai között nem egyszerű a kapcsolat.

A különböző programozási nyelvek termelékenységének összehasonlítása szintén félrevezető képet adhat a programozói termelékenységről, mert minél kifejezőbb egy programozási nyelv, annál alacsonyabb a kézzelfogható termelékenység. Ez a visszasság onnan ered, hogy a termelékenység számításánál min-

den szoftverfejlesztői tevékenységet együtt vesznek számításba, de a használt LOC-mértéket csak a programozási folyamatra alkalmazzák. Ezért, ha az egyik nyelvben ugyanazt a funkciót több sorban lehet implementálni, mint a másikban, akkor a termelékenység becslése hibás eredményre vezet.

Vegyünk például egy olyan rendszert, amelynek assembly nyelven 5000, C nyelven 1500 sorból állna a kódja. A különféle fázisok fejlesztési időtartama a 26.2. ábrán látható. Az assembler-programozónak a termelékenysége 714 sor/hónap, a C nyelvet használó programozónak pedig 300 sor/hónap, vagyis kevesebb, mint az előző fele, ezért a C nyelven megírt rendszerek fejlesztési költsége alacsonyabb és elkészítési idejük is rövidebb.

	Elemzés	Tervezés	Kódolás	Tesztelés	Dokumentálás
Assembly kód	3 hét	5 hét	8 hét	10 hét	2 hét
Magas szintű nyelv	3 hét	5 hét	4 hét	16 hét	2 hét

	Méret	Munka	Termelékenység
Assembly kód	5000 sor	28 hét	714 sor/hónap
Magas szintű nyelv	1500 sor	20 hét	300 sor/hónap

26.2. ábra. A rendszerfejlesztési idők

A kód méretének mint a becsült termék egy jellemzőjének másik felhasználási módja, hogy a kód funkcionalitásának mérésére használunk valamilyen mértéket. Ezzel a módszerrel kiküszöbölhető a fenti visszasság, mivel a funkcionalitás független attól a nyelvtől, amelyen az implementálás történik. MacDonell (MacDonell, 1994) több különböző funkcióalapú mértéket hasonlított össze és adott róluk rövid leírást. Ezen mértékek közül a legismertebb a funkciópont-számítás. A mérték javaslata Albrechtől (Albrecht, 1979) ered, és a finomítása Albrecht és Gaffney (Albrecht és Gaffney, 1983) munkásságának az eredménye. Garmus és Herron (Garmus és Herron, 2000) leírja a funkciópontok szoftverprojektekben történő gyakorlati felhasználását.

A termelékenységet az egy emberhónapban előállított funkciópontokban fejezi ki. Egy funkciópont nem egyedülálló jellemző, hanem a programjellemzők kombinációja. A program összes funkciópontjainak száma a következő programjellemzők mérése vagy becslése alapján számítható:

- külső bemenetek és kimenetek;
- felhasználói interaktivitás;
- külső interfészek;
- a rendszer által használt állományok.

Nyilvánvalóan egyes bemenetek, kimenetek, interakciók stb. összetettebbek, mint mások, így implementálásuk hosszabb időt vesz igénybe. A funkciópont-

jellemzők összetettségét egyenként felméri, és egy súlyt rendelnek hozzájuk, amelynek a felvehető legkisebb értéke 3 (egyszerű külső bemenetnél), legmagasabb értéke pedig 15 (bonyolult belső állományoknál). Az Albrecht által javasolt súlyok vagy a helyi tapasztalatokon alapuló értékek egyaránt alkalmazhatók.

A kiegyensúlyozatlan funkciópont-számításnál (unadjusted function-point count, UFC) minden egyes funkciópont-darabszámot szoroznak a becsült súlylyal, majd az így kapott szorzatokat összeadják.

$$\text{UFC} = \sum (\text{az adott típus elemeinek száma}) \times (\text{súly})$$

A funkciópont-számításnak ezt az alapvető formáját különböző összetettségű tényezők tovább módosítják. A tényezők értékét a projekt teljes összetettsége alapján határozzák meg, ahol számításba veszik az osztott folyamatok fokát, az újrafelhasználás mértékét, a teljesítményt stb. A funkciópont végső kiszámításához az UFC-t a projekt összetettségi tényezőivel kell megszorozni.

Symons (Symons, 1988) jegyezte meg, hogy mivel az összetettség szubjektíven becsülhető, ezért egy program funkciópont-számítása a becslő személyétől függ, hiszen a különböző emberek másképpen ítélik meg az összetettséget. A funkciópontok összeszámlálására eltérő variációk elképzelhetők, a becslő ítéletétől vagy az éppen fejlesztett rendszer típusától függően. Ráadásul a funkciópontok jobban alkalmazhatók az adatfeldolgozó rendszereknél, ahol sok az input-output művelet. Nehezebb összeszámlálni a funkciópontokat egy eseményvezérelt rendszernél. Mindezek alapján sokan úgy gondolják, hogy a funkciópontok nem túlságosan jók a szoftver termelékenységének mérésére (Furey és Kitchenham, 1997; Armour, 2002). Ugyanakkor a funkciópontok alkalmazói mellett érvelnek, hogy ezek hibái ellenére a gyakorlati helyzetekben hatékonyan használhatók (Banker és mások, 1993; Garmus és Herron, 2000).

A funkciópontok alternatíváját az objektumpontok jelentik (Banker és mások, 1992). Ezek az adatbázis-programozási nyelvekben és a szkriptnyelvekben használhatók. Az objektumpontok nem egyeznek meg az objektumorientált szemléletű szoftverfejlesztés során létrehozott objektumosztályokkal, a program objektumpontjainak száma inkább a következő elemek súlyozott becslése:

1. *A külön megjelenítendő képernyők száma.* Az egyszerű képernyők 1, a közepesen bonyolult képernyők 2, a nagyon bonyolult képernyők 3 objektumpontnak számítanak.
2. *Az elkészített jelentések száma.* Az egyszerű jelentések 2, a közepesen bonyolult jelentések 5 objektumpontot érnek, a feltehetőleg nagyon bonyolult módon előállítható jelentések pedig 8 objektumpontot érnek.
3. *Az imperatív nyelveken (mint a Java vagy a C++) kifejlesztett, az adatbázis-programozás kódját kiegészítő modulok száma.* Minden modul 10 objektumpontot ér.

Az objektumpontokat használja a COCOMO II becslő modell is (ott alkalmazáspont a neve), ezt később tárgyalom a fejezetben. Az objektumpontok használatának előnye a funkcióponttal szemben, hogy az objektumpontok könnyebben

becsülhetők a magas szintű szoftverspecifikáció alapján, mert csak a képernyőkhöz, a jelentésekhez és a hagyományos programozási nyelvekhez kapcsolódnak. Nem vonatkoznak implementációs részletekre, és az összetettségi faktor becslése így jóval egyszerűbb.

Funkciópontok vagy objektumpontok használata esetén a pontok száma a fejlesztési folyamat korai szakaszában megbecsülhető, mielőtt a program méretét befolyásoló döntések megszületnek. Ezeknek a paramétereknek a becslését a külső interakciók tervezése előtt meg kell tenni. Ebben a szakaszban nagyon nehéz pontosan megbecsülni, hogy a program forráskódja hány sorból fog állni.

A funkciópont- és objektumpont-számítás használható azokkal a technikákkal együtt, amelyek a kód sorait becslik. A funkciópontok számát a végső kód-méret becslésére használják. Egy olyan nyelvben, amelyben megvalósíthatók a funkciópontok, a korábbi adatok elemzésével meg lehet becsülni a kódsorok átlagos számát (average number of lines of code, AVC). Az AVC értéke assembly nyelv esetén 200 és 300 LOC/FP, adatbázis-programozási nyelv (mint amilyen az SQL) esetén 2 és 40 LOC/FP között van. Egy új alkalmazás kódjának becsült méretét a következőképpen számítják ki:

$$\text{Kódméret} = \text{AVC} \times \text{funkciópontok száma}$$

A szervezetben dolgozó mérnökök egyéni termelékenységét számos tényező befolyásolja, amelyek közül a legfontosabbakat a 26.3. ábra foglalja össze, de az egyéni képességbeli különbségek bármely más tényezőnél fontosabbak. A termelékenység mérésének korai szakaszában Sackman és mások (Sackman és mások, 1968) arra jöttek rá, hogy néhány programozó termelékenysége több mint tízszerese volt másokéhoz viszonyítva. Tapasztalataim szerint ez ma is fennáll. Nagy csapatban nagyobb az esély arra, hogy vegyes képességű emberek kerülnek össze, ezért a termelékenység „átlagos” lesz, míg a kisebb csapatnál az általános termelékenység főként az egyéni adottságoktól és képességektől függ.

Tényező	Leírás
Alkalmazási terület ismerete	A hatékony szoftverfejlesztéshez elengedhetetlen az alkalmazási terület ismerete. A legeredményesebb mérnökök valószínűleg azok lesznek, akik már értenek a területhez.
Folyamat minősége	A használt fejlesztési folyamatnak jelentős hatása lehet a termelékenységre. Ezzel a 28. fejezetben foglalkozom.
Projekt mérete	Minél nagyobb egy projekt, annál több időt igényel a csapatkommunikáció, annál kevesebb idő áll rendelkezésre a fejlesztéshez, így csökken az egyének termelékenysége.
Technológiai támogatás	Alkalmas támogató technológiákkal (például CASE-eszközökkel, konfigurációkezelési rendszerekkel) növelhető a termelékenység.
Munkakörnyezet	Ahogy azt a 25. fejezetben tárgyaltam, a termelékenység növekedését elősegíti a csendes munkakörnyezet, ahol lehetőség van saját munkahelyiségre.

26.3. ábra. A szoftvertervezés termelékenységét befolyásoló tényezők

Az „átlagos” termelékenységi érték fogalma alkalmazási területre vagy szervezetre nem létezik. Nagy, komplex beágyazott rendszereknél a termelékenység csökkenhet 30 sor/programozóhónapra is, a lényegre törő, jól megértett alkalmazási rendszereknél pedig akár a 900 sor/programozóhónapot is elérheti. Objektumpontos mérés esetén Boehm és mások (Boehm és mások, 1995) szerint a termelékenység az eszköztámogatás és a fejlesztői képességek függvényében 4 és 50 objektumpont/hónap között változik.

A mennyiség/idő formában kifejezett mértékekkel a gond az, hogy nem veszük számításba a nemfunkcionális szoftverjellemzőket, mint például a megbízhatóságot, a karbantarthatóságot stb., és azt sugallják, hogy mindig a több a jobb. Beck (Beck, 2000) extrém programozást tárgyaló munkájában kiválóan rámutatott, hogy ha a kód folyamatos egyszerűsítésén és javításán alapuló szemlélet szerint dolgoznak, akkor a sorok számlálásának nincs sok értelme.

Az ilyen mértékek a kódgenerátorral vagy más eszközzel előállított szoftver újrafelhasználásának lehetőségét sem veszik figyelembe. Mi igazából egy adott funkcionalitású, minőségű, teljesítményű, karbantarthatóságú stb. konkrét rendszer előállításának költségét szeretnénk megbecsülni, ez pedig csak közvetetten kapcsolódik az olyan fizikai mértékekhez, mint a rendszer mérete.

Ha a vezetők a termelékenységi mérésekből ítélik meg a résztvevők képességeit, akkor lehet, hogy a mérnökök termelékenységük fokozásához engednek a minőségből, és ekkor előfordulhat, hogy egy „kevésbé termelékeny” programozó megbízhatóbb kódot ír, ami könnyebben érthető és olcsóbban karbantartható. A termelékenységi méréseket a fenti okok miatt csak útmutatásként szabad figyelembe venni, és csak gondos elemzés után következhet az alkalmazásuk.

26.2. BECSLÉSI TECHNIKÁK

Nem létezik olyan egyszerű módszer, amellyel egy szoftverrendszer kifejlesztéséhez szükséges munka mennyisége pontosan becsülhető lenne. A kezdeti becsléseket a magas szintű felhasználói követelmények definíciója alapján kell elvégezni. Lehet, hogy a szoftvernek idegen számítógépen kell futnia, hogy új fejlesztési technológiát is alkalmazni kell, és talán a projektben dolgozó emberek és szakértelmük sem ismert. Ezek a tényezők azt jelentik, hogy a projekt kezdeti szakaszában nehéz pontos becslést adni a rendszerfejlesztés költségeire.

A fentiekén túl nehézséget okoz annak megállapítása is, hogy mennyire hitelesek a különböző költségbecslési technikák. A projektköltségek becslései gyakran önmagukat valósítják meg: a projekt költségvetését a becslés alapján határozzák meg, a terméket pedig úgy igazítják ehhez, hogy teljesüljenek a költségvetésben előírt számok. Nincs tudomásom egyetlen olyan projektköltséget vizsgáló, irányított kísérleti jelentésről sem, ahol a becsült költségek ne befolyásolták volna a kísérletet. Egy irányított kísérletben a költségbecslést nem kellene ismertetni a projektvezetővel, és ekkor összehasonlíthatók lennének a tényleges költségek és a projekt becsült költségei.

Mindemellett a szervezeteknek meg kell becsülniük a szoftverre fordított munkát és a költségeket. A becsléshez egy vagy több, a 26.4. ábrában összefoglalt technika használható (Boehm, 1981). Ezen technikák mindegyike a projektvezető gyakorlatán alapul, aki a megelőző projektekben szerzett ismereteit használja fel a projekthez szükséges erőforrások becslésére. Viszont lényeges különbségek lehetnek a múltbeli és jövőbeli projektek között. Sok új fejlesztési módszer és technika került bevezetésre az elmúlt 10 évben. Néhány példa a változásra, amelyek hatással lehetnek a gyakorlaton alapuló becslésekre:

1. elosztott objektumrendszerek nagyszámítógépes rendszerek helyett;
2. webszolgáltatások használata;
3. ERP- és adatbázis-központú rendszerek;
4. kész szoftverek használata eredeti rendszer kifejlesztése helyett;
5. újrafelhasználás-alapú fejlesztés, a rendszer minden részének új kifejlesztése helyett;
6. fejlesztés szkriptnyelvek (például TCL vagy Perl) használatával (Ousterhout, 1988);
7. CASE-eszközök és programgenerátorok használata nem támogatott szoftver-fejlesztése helyett.

Ha a projektvezetők nem élnek ezekkel a technikákkal, akkor a korábbi tapasztalataik nem segítenek nekik a szoftverprojekt költségeinek becslésénél. Ez pedig bonyolultabbá teszi a pontos költségbecslést és az ütemezés megbecslését.

A költségbecslés 26.4. ábrán ismertetett technikai mind „fentről lefelé”, mind „lentől fölfelé” szemléleteknél alkalmazhatók. A fentről lefelé megközelítés a rendszer szintjén kezdődik: a becslő személy először a termék átfogó funkcionalitását vizsgálja meg, és azt, hogy ezt a funkcionalitást hogyan szolgálják az egymással kapcsolatban álló alfunkciók. Számításba veszik a rendszerszintű tevékenységeket is, mint például az integrációt, a konfiguráció- és dokumentációkezelést.

Ezzel ellentétben a lentől fölfelé megközelítésnél a becslés a komponensek szintjén kezdődik. A rendszer komponensekre bontása után mindegyik komponenshez ki kell számítani a fejlesztéséhez szükséges munkát, és a teljes rendszer kifejlesztéséhez szükséges munkát mindezen költségek összeadásával kapjuk meg.

A fentről lefelé megközelítés hátrányai ugyanazok, mint a lentől fölfelé megközelítés előnyei, és fordítva. A fentről lefelé becslés alábecsülheti a speciális komponensekkel (például a nem szabványos hardverelemek interfészeivel) kapcsolatos bonyolult technikai problémák megoldásának költségét, és az elvégzett becslésnek nincs részletes indoklása. A lentől fölfelé becsléshez ugyan készül ilyen részletes indoklás, és a rendszer minden komponensét beleszámítja, de ezzel a módszerrel nagyobb az esély arra, hogy alábecsüljük a rendszertevékenységek (például integráció) költségét, és többre is kerül: kezdeti rendszertervet kell készíteni azoknak a komponenseknek az azonosítására, amelyeknek a költségét meg kellene állapítani.

Minden becslési technikának megvan a maga erőssége és a maga gyenge pontja is. Mindegyikük különböző információkat használ fel a projektről és a fej-

Technika	Leírás
Algoritmusos költségmodellezés	A korábbi költségbecslések felhasználásával egy modellt fejlesztenek ki, amely kapcsolatot állít fel valamilyen szoftvermérték (általában a szoftver mérete) és a projektköltség között. Ezzel a metrikával végzik el a becslést, és a modell előre jelzi a szükséges munkát.
Szakértői vélemény	Többen is szakvéleményt adnak a javasolt szoftverfejlesztési technikákról, és megvizsgálják az alkalmazási területet. Mindannyian megbecslik a projektköltséget, majd összehasonlítják és megvitatják ezeket a becsléseket. A becslési folyamat addig ismétlődik, amíg egy becslést közös megegyezéssel el nem fogadnak.
Becslés hasonló eset alapján	Ez a technika akkor alkalmazható, ha az alkalmazási területen belül a szervezetnek van már egy befejezett projektje. Az új projekt költségeit a befejezett projekttel való hasonlóság alapján becslik. A szemlélet egy nagyon tiszta leírását Myers (Myers, 1989) készítette el.
Parkinson törvénye	Parkinson törvénye kimondja, hogy a munka kitölti a rendelkezésre álló időt. A költséget a rendelkezésre álló erőforrásoknak kell meghatározniuk, nem a becslésnek. Ha a szoftverrel 12 hónap alatt kell elkészülni és 5 ember áll rendelkezésre, akkor a szükséges munka 60 emberhónap.
Nyerő ár	A projekt költségének becsült értéke az az összeg, amelyet a vevő a projektre képes költeni. A becsült munka nem a szoftver funkcionalitásától, hanem a vevő költségvetésétől függ.

26.4. ábra. Költségbecslési technikák

lesztőcsapatról, ha csak egyetlen modellt alkalmazunk, és az információk pontatlanok, a végső becslés hibás lesz. Nagy projekteknél ajánlott több költségbecslési technikát használni, és összevetni ezek eredményét. Ha ezek a becslések gyökeresen különböző költségeket jeleznek, akkor ez utalhat arra, hogy a költségek megállapításához nem áll rendelkezésre elég információ. Ilyen esetben még több információt kell beszerezni, és addig kell ismételni a költségfelmérési folyamatot, amíg a becslések nem kezdenek hasonlítani egymásra.

A fenti becslési technikák ott alkalmazhatók, ahol már elkészítették az összes felhasználói és rendszerkövetelményt meghatározó és azt rögzítő dokumentumot. Itt már elfogadható becslés adható arról, mekkora a rendszer kifejlesztendő funkcióinak kiterjedése. Általánosságban elmondható, hogy a nagy rendszerek tervezési projektjeinél rendszerint létezik ilyen követelményeket leíró dokumentum.

Nemritkán azonban sok projekt költsége csak a rendszer felhasználói követelményeinek körvonalai alapján becsülhető meg, ami azt jelenti, hogy a becslés készítőinek nagyon kevés információ áll a rendelkezésére. A követelmények elemzése és specifikálása költséges, és lehet, hogy egy vállalat vezetőinek a részletes követelmények vagy a rendszerprototípus kifejlesztése nélkül, a költségvetés tényleges jóváhagyatása előtt fel kell mutatniuk a rendszerre vonatkozó kiindulási költségbecslést.

Ilyen körülmények között a leggyakrabban alkalmazott stratégia a „nyerő ár”. Első ránézésre az elképzelés etikátlannak és üzletszerűtlennek tűnhet, mégis

vannak előnyei. A projekt költségében az ajánlatok vázlatos ismertetése alapján állapodnak meg, ezután ül le az ügyfél és a vevő tárgyalni a részletes projektspecifikáció felállításáról, amelyet a megállapított ár korlátoz. A vevőnek és a szállítónak meg kell egyeznie arról, mi az elfogadható rendszer-funkcionalitás. Sok projektben a rögzített tényező nem a projektkövetelmény, hanem a költség, és a követelmények megváltoztathatók, hogy ne lépjenek túl a költségeket.

Például egy vállalat ajánlatot tett egy olajtársaságnak új üzemanyag-ellátó rendszer kifejlesztésére, amivel az a saját állomásait ki tudja szolgálni. A rendszerhez nincs részletes követelmény dokumentum, így a fejlesztő az árat 900 000 dollárra becsüli, ez versenyképesnek tűnik, és beleillik az olajtársaság költségvetésébe. Miután elnyerték a szerződést, megbeszélik a részletes rendszerkövetelményeket, az alapvető funkcionalitások megvalósulnak; ezután megbecslik a további követelmények járulékos költségeit. Az olajtársaság nem feltétlenül veszít, mivel van érvényes szerződése, amit a vállalat be akar tartani. A további követelmények teljesítése egy jövőbeli költségvetésbe kerülhet bele, így az olajtársaság költségvetése nem fog összeomlani a nagyon magas induló szoftverkötség miatt.

26.3. AZ ALGORITMIKUS KÖLTSÉGMODELLEZÉS

Az algoritmikus költségmodellek a projekt méretének, a programozók számának és más folyamat- és terméktényezőnek becslésén alapuló költség-előrejelzést egy matematikai formulával írják fel. Egy algoritmikus költségmodell a befejezett projektek költségeinek és jellemzőinek elemzésével építhető föl úgy, hogy megtaláljuk az aktuális gyakorlathoz legjobban illeszkedő zárt formulát.

Az algoritmikus költségmodellek elsősorban a szoftverfejlesztés költségeinek becslésére valók, de Boehm (Boehm, 2000) tárgyalja az algoritmikus költségbecslés más alkalmazásait is, beleértve a szoftvercégek befektetőinek, a kockázatfelmérés alternatív stratégiáinak és az újrafelhasználás, újrafelállítás és kihelyezés döntési információinak becslését.

A szoftverek költségének algoritmikus költségbecslése legáltalánosabb formájában a következőképpen fejezhető ki:

$$\text{Munka} = A \times \text{Méret}^B \times M$$

Az A egy konstans tényező, a helyi szervezeti szokásoktól és a fejlesztett szoftver típusától függ. A Méret lehet a szoftver kód méretének becslése vagy a funkció-, illetve objektumpontokban kifejezett funkcionalitás becslése. A B kitevő értéke általában az 1 és 1,5 által behatárolt intervallumból kerül ki, és a nagy projektekhez szükséges többletmunkát jelzi. Az M szorzó a különböző folyamat-, termék- és fejlesztési jellemzőkből tevődik össze.

A legtöbb algoritmikus becslési modell tartalmaz egy kitevőt (B a fenti egyenletben), amely a méret becslésére vonatkozik. Ez azt a tényt tükrözi, hogy a költség normális esetben nem a projekt méretével lineárisan nő. Ahogy a szoftver

mérete nő, extra költségek jelennek meg a nagyobb csapat kommunikációs többlete, a bonyolultabb konfigurációkezelés, a bonyolultabb rendszer-integráció stb. miatt. Így minél nagyobb a rendszer, annál nagyobb a kitevő.

Minden algoritmikus modellnél ugyanazok az alapvető nehézségek merülnek fel:

1. *Ahol csak a specifikáció áll a rendelkezésre, ott a projekt kezdeti szakaszában gyakran nehéz megbecsülni a Méret-et.* A funkciópont- és objektumpontbecslések ugyan könnyebben elkészíthetők, mint a kódméret becslései, de pontatlanabbak is lehetnek.
2. *A B-t és az M-et alkotó tényezők becslése szubjektív.* Az értékeket az előismeretek és tapasztalatok függvényében mindenki másképp becsli meg.

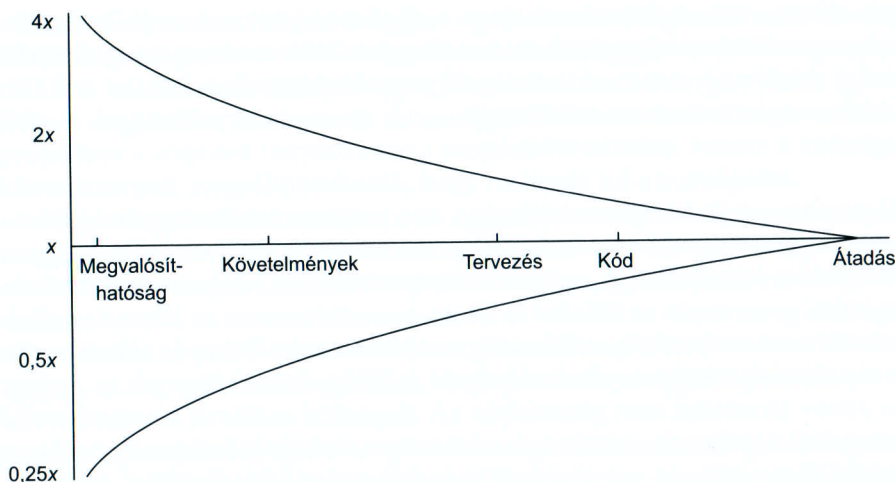
A legtöbb algoritmikus költségmodell alapmetrikája a befejezett rendszer forráskódjában található sorok száma. A méretet meg lehet becsülni más projektekkel való hasonlóság alapján, a kódméret funkciópontokra való konvertálásával, vagy rangsorolhatják a rendszerkomponensek méretét, és egy ismert referenciakomponenssel megbecsülve a komponensméretet, megbecsülhető a rendszer mérete is, de a becslés lehet akár egyszerűen a mérnökök megítélésének kérdése is.

A kód méretének pontos becslése bonyolult, mivel a végleges rendszer méretét olyan tervezési döntések is befolyásolják, amelyek a kezdeti becslések idején még nem feltétlenül ismertek. Egy összetett adatkezelést igénylő alkalmazás például implementálhatja saját adatkezelőjét, de használhat kereskedelmi adatbázist is. Ha kereskedelmi adatbázist használunk, akkor kisebb lesz a kód mérete, de további munka válhat szükségessé a kereskedelmi termék hatékonyságának növeléséhez.

A rendszerfejlesztéshez használt programozási nyelv is befolyásolja a fejlesztett kód sorainak számát. Ha olyan nyelvet választanak, mint a Java, akkor a kód több sorból állhat, mintha (mondjuk) C nyelven programoztak volna, viszont ez a kódtöbblet több fordítási idejű ellenőrzést tesz lehetővé, így csökkenthetők a validálási költségek. Ez viszont hogyan vehető számításba a becslésnél? Ráadásul a méret becslésénél a szoftverfejlesztési folyamatban az újrafelhasználhatóság mértékének becsült értékét sem szabad figyelmen kívül hagyni.

A becslést végző személynek nem egy, hanem többféle becslést kellene készítenie (a legrosszabb, várt és legjobb esetekre), mindegyiknél a költségbecslő képletek alkalmazásával. A kezdeti becslésekben még valószínűleg több a hiba, a becslés várhatóan akkor válik pontosabbá, amikor jól megértik a terméket, hangolják a modellt a felhasználó szervezethez, és előre meghatározzák a választható nyelvek és hardver körét.

Az algoritmikus modell alapján tett becslések pontossága a rendelkezésre álló rendszer-információtól függ. A szoftverfejlesztés előrehaladtával egyre több információ gyűlik össze, ezért a becslések egyre pontosabbá válnak. Ha a szükséges munkát kezdetben x munkahónapra becsülik, akkor a rendszer első tervezeténél a becslés értéke $0,25x$ és $4x$ között mozoghat, a fejlesztési folyamat során



26.5. ábra. A becslési bizonytalanság

pedig fokozatosan egyre pontosabbá válik (26.5. ábra). A Boehm tanulmányából (Boehm és mások, 1995) vett ábra több szoftverfejlesztési projekt tapasztalatait használta fel. Természetesen, közvetlenül az átadás előtt egy nagyon pontos becslés készíthető.

26.3.1. A COCOMO-modell

Számos olyan algoritmikus modellt ajánlottak már, amelyeket alapként fel lehetne használni a szoftverprojektek munkamennyiségének, ütemezésének és költségeinek becsléséhez. Ezek a modellek hasonló elven működnek, de különböző paraméterértékeket használnak. Közülük az empirikus COCOMO-modellt mutatom be. A modell felállításához nagyon sok szoftverprojektből gyűjtöttek össze, majd elemeztek adatokat, hogy megtalálják a megfigyelésekre legjobban illeszkedő képleteket. Ezek a formulák a rendszer és a termék méretére, a rendszer fejlesztésére vonatkozó munka projekt- és csapattényezőire vonatkoznak.

A COCOMO-t a következő okok miatt választottam:

1. Jól dokumentált, mindenki számára szabadon hozzáférhető, és támogatják mind a szabadon hozzáférhető, mind a kereskedelmi eszközök.
2. Széles körben használják, és nagyra értékelik.
3. Hosszú múltja van az első, 1981-es megvalósításától (Boehm, 1981) kezdve az Ada nyelvű szoftverfejlesztésekhez tökéletesített verzió (Boehm és Royce, 1989) keresztül a legfrissebb, 2000-ben kiadott változatáig (Boehm és mások, 2000).

A COCOMO-modellek átfogóak, sok paraméterrel, melyek mindegyike értékek egy tartományát veheti fel. Olyan összetettek, hogy nem tudom a teljes leírá-

Projekt bonyolultsága	A képlet	Leírás
Egyszerű	$PM = 2,4 (KDSI)^{1,05} \times M$	A kis csapatok által fejlesztett jól megértett alkalmazások.
Közepesen bonyolult	$PM = 3,0 (KDSI)^{1,12} \times M$	A bonyolultabb projektek, ahol a csapattagok már lehet, hogy rendelkeznek valamennyi tapasztalattal a hasonló rendszerek területén.
Beágyazott	$PM = 3,6 (KDSI)^{1,20} \times M$	Bonyolult rendszerek, ahol a szoftver része a hardver, a szoftver, az előírások és a műveleti eljárások szorosan összetartozó egészének.

26.6. ábra. A COCOMO 81 alapmodell

sukat itt megadni. Inkább csak a lényegi jellemzőiket tárgyalom az algoritmikus költségmodellezés alapismereteinek megadásával.

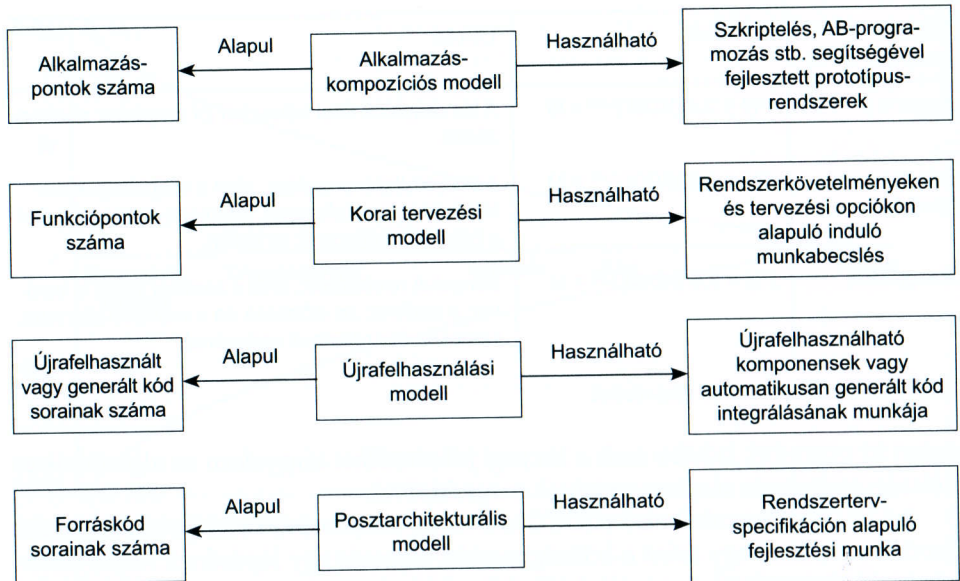
A COCOMO-modell első (COCOMO 81 néven ismert) verziója háromszintű modell volt, ahol egy szint a költségbecslési elemzés egy lépésének részletezettségét tükrözte. Az első szint (az alap) kezdeti, durva becslést adott, a második szint több projekt- és folyamatszorzó bevezetésével módosította az első szint becslését, a legjobban részletezett szint pedig a projekt különböző szakaszait becsülte. A COCOMO-modellnek a különböző típusú projektekhez használt alapképletei a 26.6. ábrán láthatók. Az M szorzó a termék, a projekt és a csapat jellemzőit tükrözi.

A COCOMO 81 modell feltételezte, hogy a fejlesztés a vízesésfolyamattal (4. fejezet) és a szabványos imperatív programozási nyelvek (C vagy FORTRAN) segítségével történik. Azonban a modell első ajánlása óta radikális változások történtek a szoftverfejlesztés területén. A prototípus-készítés és az inkrementális fejlesztés általánosan használt folyamatmodellek lettek. Adatintenzív rendszereket adatbázis-programozási nyelvek (például SQL) és kereskedelmi adatbázis-kezelő rendszerek használatával fejlesztenek. A szoftverek ma már valamilyen szkriptnyelvvel összekapcsolt, újrafelhasználható komponensekből is összeállíthatók, és sok esetben kész alrendszereket használnak fel. Létező szoftvereket is újra lehet tervezni, hogy új szoftvert állítsanak elő belőlük, és a szoftverfolyamat legtöbb tevékenységéhez már elérhetők a CASE-eszközök.

A COCOMO II modell, hogy alkalmazkodjon a változásokhoz, elismerte a szoftverfejlesztés különböző megközelítéseit, mint például a prototípus-készítést, a komponensek összeállításán alapuló fejlesztést, az adatbázis-programozási nyelvek használatát stb. A COCOMO II egy spirális fejlesztési modellt (4. fejezet) támogat, és beágyaz különböző almodelleket, amelyek növekvő részletezettségű becslésekre alkalmasak. Ezek a fejlesztési spirál ismétlődő köreibben használhatók. A 26.7. ábra mutatja a COCOMO II almodelljeit és azok alkalmazási területeit.

A COCOMO II modell almodelljei a következők:

1. *Alkalmazáskompozíciós modell.* Feltételezi, hogy a rendszer újrafelhasználható komponensekből, szkripteléssel vagy adatbázis-programozással jön létre.



26.7. ábra. A COCOMO II modell

Prototípusfejlesztés becslésére tervezték. A szoftver méretének becslése az alkalmazáspontokon alapul, a szükséges munka becslésére egy egyszerű méret/termelékenység formula használatos. Az alkalmazáspont jelentése ugyanaz, mint a 26.1. alfejezetben tárgyalt objektumponté, csak az objektumorientált fejlesztéssel való keveredés elkerülése miatt lett átnevezve.

2. *Korai tervezési modell.* Ez a modell a rendszer tervezésének korai szakaszában használható, amikor a követelmények már eldőlték. A becslések a funkciópontokon alapszanak, amelyeket a forráskód sorainak számává alakítanak át. A formula alakja marad a fentebb tárgyalt egységes formájú, de szorzóknak egy hételemű halmaza kapcsolódik hozzá.
3. *Újrafelhasználási modell.* Ez a modell az újrafelhasználható komponensek és a tervből vagy programfordító eszközökkel automatikusan generált kód integrálásához szükséges munkát számolja. Általában a posztarchitekturális modellel együtt használatos.
4. *Posztarchitekturális modell.* Ha elkészült a rendszer architektúrája, meglehetősen pontos becslés tehető a szoftver méretére. Ez a modell is a korábban tárgyalt formulát használja a költségbecslésre. De 17 szorzót tartalmaz, ezekkel a személyi kapacitások, a termék és a projekt jellemzői fejezhetők ki.

Természetesen nagy rendszerekben a különböző részek különböző technológiákkal fejleszthetők, és lehetetlen a rendszer minden részét azonos pontossággal becslőni. Ebben az esetben a rendszer minden részéhez a megfelelő almodellt kell alkalmazni, és az eredményeket kombinálni, létrehozva egy összetett becslést.

Alkalmazáskompozíciós modell

Az alkalmazáskompozíciós modell a COCOMO II-ben a prototípus-készítő projektek, illetve a már létező komponenseket összeépítő szoftverfejlesztő projektek számára szükséges munka becslésének támogatására vezették be. A súlyozott alkalmazáspontok (objektumpontok) becslését el kell osztani az alkalmazáspont-termelékenység standard becslésével. A becslést eztán finomítani kell az objektumpontok fejlesztésének nehézségi foka szerint (Boehm és mások, 2000). A programozó termelékenysége éppen úgy függ a fejlesztői tapasztalatától és a képességeitől, mint a fejlesztést támogató CASE-eszközök lehetőségeitől. A 26.8. ábra mutatja a modell kifejlesztői (Boehm és mások, 1995) által javasolt objektumpont-termelékenységi szinteket.

A fejlesztő gyakorlata és képessége	Nagyon kevés	Kevés	Közepes	Sok	Nagyon magas
A CASE érettsége és lehetőségei	Nagyon kevés	Kevés	Közepes	Sok	Nagyon magas
PROD (NOP/hónap)	4	7	13	25	50

26.8. ábra. Az objektumpontokban kifejezett termelékenység

Az alkalmazáskompozíció általában szignifikáns szoftver-újrafelhasználást tételez fel, és a rendszerben az alkalmazáspontokat újrafelhasználható komponensek implementálhatják. Következésképpen az összes alkalmazáspont számát becsülhetjük a várható újrafelhasználások számával. Így a rendszerprototípusok munkájának számítására az alábbi formula használható:

$$PM = (NAP \times (1 - \%reuse/100))/PROD$$

ahol a PM a munka emberhónapban megadva, a NAP az alkalmazáspontok száma a szállított rendszerben, a %reuse az újrafelhasznált kódok becsült mennyisége a fejlesztésben, a PROD pedig a 26.8. ábra szerinti termelékenység. A modell az egyszerűség kedvéért feltételezi, hogy az újrafelhasználás nem igényel járulékos munkát.

Korai tervezési modell

Ez a modell akkor használható, ha a követelmények már ismertek és a rendszertervezés folyamatának elején vagyunk. Viszont nem szükséges hozzá a részletes architekturális terv. A cél az, hogy egy közelítő becslést adjunk az esedékes munkára. Következésképpen egyszerűsítő feltételeket teszünk, például azt, hogy az újrafelhasználható kód integrálásának nincs munkaigénye. A korai tervezés becslései jól használhatók a felhasználói követelmények implementálásának különböző lehetőségeit összehasonlító elemzésekre.

Az ebben a szakaszban végzett becslések az algoritmikus modellek szabványos képletén alapszanak, nevezetesen a következőn:

$$\text{Munka} = A \times \text{Méret}^B \times M$$

Boehm a saját adathalmazára támaszkodva azt javasolja, hogy az A együtttható értéke az ezen a szinten végzett becslésekhez legyen 2,94. A rendszer méretét KSLOC-ban mérik (egy KSLOC ezer forráskódsornak felel meg), és a szoftver funkciópontjai becsült számának KSLOC-ká való konvertálásával számítható ki. A konvertáláshoz olyan szabványos táblázatokat használnak, amelyek a funkciópontok és a szoftver mérete közötti kapcsolatot adják meg különböző programozási nyelvekhez.

A B kitevő fejezi ki, hogy a projekt méretének növekedésével növekszik a szükséges munka is. A COCOMO 81-től eltérően értéke itt nem rögzített, hanem a projekt újszerűségétől, a fejlesztés rugalmasságától, a használt kockázatfeloldási folyamattól, a fejlesztőcsapat összetartásától és a vállalat folyamatfejlettségi szintjétől (28. fejezet) függően 1,1 és 1,24 között változhat. A kitevő értékének ezen paraméterek felhasználásával történő kiszámításával a COCOMO II poszt-architektúrális modelljének leírásában foglalkozom.

Az M szorzó értéke a COCOMO II-ben hét projekt- és folyamatényező egyszerűsített halmazából számítható ki. Ezek csökkenthetik vagy növelhetik a szükséges munkát. A korai tervezési modellben ezek a jellemzők a termék megbízhatósága (RCPX), a szükséges újrafelhasználás (RUSE), a platformmal járó nehézségek (PDIF), a személyzet képessége (PERS), a személyzet gyakorlata (PREX), az ütemezés (SCED) és a támogatóeszközök (FCIL). Ezek értéke közvetlenül megbecsülhető egy hatfokú skálán, ahol a skála 1. fokának nagyon alacsony, 6. fokának pedig nagyon magas szorzóérték felel meg.

A munka tehát a következőképpen számítható ki:

$$\text{PM} = 2,94 \times \text{Méret}^B \times M$$

ahol

$$M = \text{PERS} \times \text{RCPX} \times \text{RUSE} \times \text{PDIF} \times \text{PREX} \times \text{FCIL} \times \text{SCED}$$

Újrafelhasználási modell

Ahogy a 18. és 19. fejezetben már tárgyaltam, napjainkra a szoftver-újrafelhasználás általánossá vált, és a legtöbb nagyméretű rendszer kódjának nagy százaléka korábbi fejlesztésekből származik. Az újrafelhasználási modell az újrafelhasznált vagy generált kód integrálásához szükséges munka becslésére használható.

A COCOMO II az újrafelhasznált kód két típusáról beszél. A fekete doboz kód az a kód, amelyet annak megértése vagy megváltoztatása nélkül használunk fel újra. A fekete doboz kódnak nincs fejlesztési munkaigénye. Azt a kódot, ame-

lyet adaptálunk egy új kód vagy egy újrafelhasznált komponens integrálásához, fehér doboz kódnak hívjuk. Ennek újrafelhasználásához szükséges valamennyi fejlesztési munka, mivel meg kell érteni és módosítani kell a rendszeren belüli korrekt működés eléréséhez.

Ráadásul egyes rendszerek tartalmazznak automatikusan generált kódot, melyet a rendszer modelljéből programgenerátorok állítanak elő. Ez az újrafelhasználásnak egy esete, amikor a szabványminták a generátorba vannak beépítve. A rendszermodell elemzésre kerül, majd a szabványminták alapján, a rendszermodell részleteit figyelembe véve, megtörténik a generálás. A COCOMO II korai tervezési modellen belül van egy külön modell az ilyen generált kód költségének a becslésére.

Az automatikusan generált kódhoz a modell becsli az integrálásához szükséges emberhónapot. A munka becslésének képlete:

$$PM_{\text{Auto}} = (\text{ASLOC} \times \text{AT}/100)/\text{ATPROD} \quad // \text{ Becslés a generált kódhoz}$$

A képletben az AT az automatikusan generált felhasznált kód százalékban, az ATPROD pedig az ilyen típusú kódkészítés termelékenységi szintje. Boehm és mások (Boehm és mások, 2000) az ATPROD értékét nagyjából 2400 forrásutasításnak mérték havonta. Tehát ha egy rendszerben 20 000 soros fehér doboz kódot használunk fel újra, és ebből 30 százalék az automatikusan generált, akkor ennek integrálásához a szükséges munka:

$$(20\,000 \times 30/100)/2400 = 2,5 \text{ emberhónap} \quad // \text{ Generált kód példa}$$

Az újrafelhasználási modell más komponensei akkor használhatók, ha a rendszerbe új kódokat vagy újrafelhasznált fehér doboz komponenseket kell integrálni. Ekkor az újrafelhasználási modell nem adja meg közvetlenül a munkát. Ehelyett számolja az újrafelhasznált kód sorainak száma alapján az új kód sorainak számát.

Például 30 000 soros újrafelhasznált kód esetén az új, becsült ekvivalens sorszám 6000. Lényegében 30 000 újrafelhasznált kódsor megfelel annak, mintha 6000 új kódsort írnánk. Ezt a kalkulált értéket kell hozzáadni a COCOMO II poszt-architekturális modelljében kifejlesztett új kód sorainak számához.

A becslések az újrafelhasználási modellben a következők:

ASLOC – az adaptált komponens kódsorainak száma;

ESLOC – az új forráskód ekvivalens sorszáma.

Az ESLOC számításánál figyelembe kell venni a szoftver megértéséhez, az újrafelhasználáshoz szükséges módosítás elvégzéséhez és a kód rendszerbe való integrálásához szükséges munkát. A fejlesztési munka meghatározásánál ehhez még hozzá kell venni az automatikusan generált kód mennyiségét, ahogy azt az előbb kifejtettem.

A következő formula használható az ekvivalens forráskódsorszám számításhoz:

$$ESLOC = ASLOC \times (1 - AT/100) \times AAM$$

Az ASLOC-ot csökkentjük az automatikusan generált kód százalékaival. Az adaptációs finomító szorzó (Adaptation Adjustment Multiplier, AAM) három komponens összege:

1. Egy adaptációs komponens (AAF), amely az újrafelhasznált kódban történő módosítások költségét jelzi. Ez tartalmazza a tervezés, kódolás és integrálás komponensét.
2. Egy megértési komponens (SU), amely az újrafelhasznált kód megértésének és a mérnök kódban való jártasságának költségeit reprezentálja. A SU értéke 50-től (nem strukturált kód) 10-ig (jól megírt, objektumorientált kód) terjedhet.
3. Egy felbecsülési faktor (AA), amely az újrafelhasználási döntéseket reprezentálja. Valamennyi elemzés mindig szükséges annak eldöntéséhez, hogy egy kód újrafelhasználható-e, ez épül be a költségbe az AA-n keresztül. Az AA értéke 0 és 8 között van, az elemzéshez szükséges munka mértékétől függően.

Az újrafelhasználási modell egy nemlineáris modell. Ahhoz is szükséges bizonyos munka, hogy felmérjük az újrafelhasználhatóság lehetőségét. Továbbá az újbóli újrafelhasználásoknál az újrafelhasználás költsége eltűnik, ahogy a megértés és a felmérés egyre több kódsorra kiterjed.

Posztarchitekturális szint

A posztarchitekturális modell a legrészletesebb a COCOMO II modellek között. Használható a rendszer egy kezdeti architektúráis tervezéséhez, ha már az al-rendszerszerkezet ismert.

A posztarchitekturális szint becslései ugyanazon az alapképleten ($PM = A \times Méret^B \times M$) alapulnak, mint a korai tervezésé. Ugyanakkor a becslésnek ebben a szakaszában a méret becslése pontosabb lehet. Ráadásul egy bővebb (7 helyett 17) termék, folyamat és szervezeti jellemzőegyüttes áll a rendelkezésre a kezdeti becslés finomítására. Azért lehet ebben a szakaszban több jellemzőt használni, mivel több információ áll rendelkezésre a szoftverfejlesztésről és a fejlesztési folyamatról.

A posztarchitekturális modellben a kódméret becsléséhez három komponens használatos:

1. A fejlesztett új kód összes sorának becsült száma.
2. Az újrafelhasználási modell segítségével számított ekvivalens forrássorok számának (ESLOC) becslése.
3. A követelmények változása miatt szükséges változtatások kódsorainak számára vonatkozó becslés.

Ezen három becslés összege adja a teljes kódméretet (KSLOC), amelyet a munka meghatározásának képletében használunk. A becslés utolsó komponense (a módosított kód sorainak száma) azt a tényt tükrözi, hogy a szoftverkövetelmények mindig változnak. A rendszer programjaiban ezek a követelményváltozások újabb kódok fejlesztésével jelennek meg. Természetesen a megváltozott kódsorok számának becslése nem könnyű, sőt gyakran több bizonytalanságot viszünk be vele a fejlesztés becslésébe.

A munka kiszámításának képletében lévő kitevőnek (B) a COCOMO 1-ben háromféle értéke lehet, ezek a projekt bonyolultságának különböző szintjeihez kapcsolódnak. A projekt egyre bonyolulttá válásával a növekvő rendszerméretnek egyre jelentősebb a hatása. Ugyanakkor a jó vállalati gyakorlatok és eljárások közben tudják tartani a „nem gazdaságos skálázást”. Ezt felismerve, a COCOMO II-ben a kitevő becslésénél már öt skálázó tényezőt vesznek figyelembe (26.9. ábra), ezeket egy hatfokú skálán helyezik el, a *nagyon alacsonytól* (5) a *rendkívül magasig* (0). A tényezők skálaértékét összeadják, osztják százszal, majd ezt 1,01-dal meg növelve előáll a kitevő.

Skálázó tényező	Magyarázat
Korábbi hasonló esetek hiánya	A tényező azt tükrözi, mekkora tapasztalata van a szervezetnek az ilyen típusú projektek területén. Értéke <i>nagyon alacsony</i> , ha nincsenek korábbi tapasztalatok, és <i>rendkívül magas</i> , ha a szervezet jól ismeri az alkalmazási területet.
Fejlesztés rugalmassága	A fejlesztési folyamat rugalmasságának fokát tükrözi. <i>Nagyon alacsony</i> , ha előírt folyamatokat használnak, <i>rendkívül magas</i> , ha az ügyfél csak általános célokat jelöl meg.
Architektúra/kockázatfeloldás	Az elvégzett kockázatelemzés mértékét tükrözi. <i>Nagyon alacsony</i> , ha az elemzés felszínes, <i>rendkívül magas</i> , ha az elemzés teljes és részletes.
Csapat összetartása	Tükrözi, hogy a fejlesztőcsapatban mennyire ismerik egymást, és milyen jól tudnak együtt dolgozni. <i>Nagyon alacsony</i> , ha a kölcsönhatások nagyon bonyolultak, <i>rendkívül magas</i> , ha a csapat összeforrt, jó teljesítményt nyújt, és nincsenek kommunikációs problémák.
Folyamat fejlettsége	A szervezet folyamatainak érettségét tükrözi. Értékének kiszámítása függ a CMM Maturity Questionnaire-től, de becsülhető úgy, hogy 5-ből kivonják a CMM szerinti folyamatfejlettségi szintet.

26.9. ábra. A COCOMO II kitevőjének számításánál használt skálázó tényezők

Szemléltetésképpen tekintsünk egy szervezetet, amelynek nincsenek korábbi tapasztalatai arról a területről, amellyel a projektben foglalkoznak. A projekt megrendelője nem határozta meg az alkalmazandó folyamatot, és a projektütemezésben nem engedélyezett időt az alaposabb kockázatelemzésre. A rendszer implementálására új fejlesztőcsapatot kell összeállítani. A szervezet nemrég vezetett be egy folyamatjavító programot, és a CMM-modell szerint a 2. szinten álló szervezetnek felelnek meg (28. fejezet). A kitevő kiszámításánál a tényezők lehetséges értékei a következők:

- *Korábbi hasonló esetek hiánya.* A projekt a szervezet egy új projektje, ezért az osztályzat: Alacsony (4).
- *A fejlesztés rugalmassága.* Az ügyfél nem szól bele a fejlesztésbe, ezért az osztályzat: Nagyon magas (1).
- *Architektúra/kockázatfeloldás.* Nem végeznek kockázatelemzést, ezért az osztályzat: Nagyon alacsony (5).
- *A csapat összetartása.* Új a csapat, így nincsenek információk, ezért az osztályzat: Közepes (3).
- *A folyamat érettsége.* Alkalmaznak valamilyen folyamatszabályozást, ezért az osztályzat: Közepes (3).

Az osztályzatok összege 16, így a kitevő $0,16 + 1,01 = 1,17$.

A posztarchitekturális modell kezdeti becsléseinek korrigálására és az M szorzó megadására használt jellemzők (26.10. ábra) négy osztályba sorolhatók:

1. A termékjellemzők a fejlesztés alatt álló szoftverterméktől elvárt jellemzőkre vonatkoznak.
2. A számítógép jellemzői a hardverplatform által a szoftverre tett megszorítások.
3. A személyzeti jellemzők a projektben dolgozó emberek tapasztalatait és képességeit kifejező szorzók.
4. A projektjellemzők a szoftverfejlesztési projekt egyes jellemzőit érintik.

Jellemző	Típus	Leírás
RELY	Termék	A rendszer elvárt megbízhatósága
CPLX	Termék	A rendszermodulok bonyolultsága
DOCU	Termék	A szükséges dokumentummennyiség
DATA	Termék	A használt adatbázis mérete
RUSE	Termék	Hány százaléknyi komponens szükséges újrafelhasználni
TIME	Számítógép	A végrehajtási idő korlátai
PVOL	Számítógép	A fejlesztői platform változékonysága
STOR	Számítógép	A memóriakorlátok
ACAP	Személyzet	A projektelemzők képességei
PCON	Személyzet	A személyzet folytonossága
PCAP	Személyzet	A programozók képességei
PEXP	Személyzet	A programozók gyakorlata a projekt szakterületén
AEXP	Személyzet	Az elemzők gyakorlata a projekt szakterületén
LTEX	Személyzet	Gyakorlat a nyelvek és az eszközök terén
TOOL	Projekt	A szoftvereszközök használata
SCED	Projekt	A fejlesztési ütemezés összesűritése
SITE	Projekt	Mekkora kiterjedésű a több helyszínen folyó munka, és a helyszínek közötti kommunikáció minősége

26.10. ábra. A projektköltség összetevői

A 26.11. ábrán láthatunk egy példát arra, hogyan befolyásolhatják a projekt-költségek tényezői a munka becslését. A kitevő, mint a fenti példában, itt is 1,17, és feltételezem, hogy a projekt legfőbb költségtényezői a RELY, a CPLX, a STOR, a TOOL és a SCED. Az összes többi költségtényező névleges értéke 1, tehát nem befolyásolja a munka kiszámítását.

A kitevő értéke	1,17
A rendszer mérete (az újrafelhasználási tényezőt és a követelményváltozékonysági tényezőt is tartalmazza)	128 000 DSI
A COCOMO kezdeti becslése, a költségekre ható tényezők nélkül	730 emberhónap
Megbízhatóság	Nagyon magas, szorzó = 1,39
Bonyolultság	Nagyon magas, szorzó = 1,3
Memóriakorlátok	Magas, szorzó = 1,21
Eszközhazsnálat	Alacsony, szorzó = 1,12
Ütemezés	Gyorsuló, szorzó = 1,29
A korrigált COCOMO-becslés	2306 emberhónap
Megbízhatóság	Nagyon alacsony, szorzó = 0,75
Bonyolultság	Nagyon alacsony, szorzó = 0,75
Memóriakorlátok	Nincs, szorzó = 1
Eszközhazsnálat	Nagyon magas, szorzó = 0,72
Ütemezés	Normális, szorzó = 1
A korrigált COCOMO-becslés	295 emberhónap

26.11. ábra. A költségtényezők hatása a munka becslésére

A 26.11. ábrán a fő költségtényezőkhöz minimális vagy maximális értéket rendeltem, hogy látható legyen a munka becslésére gyakorolt hatásuk. Az értékek a COCOMO II referenciakönyvből valók (Boehm, 1997). Megfigyelhető, hogy ha a költségtényezők értékei magasak, akkor a munka becslése több mint háromszorosra lesz a kezdeti becslésnek, míg alacsony értékekkel az eredeti becslést a harmadára csökkentik. Ez rávilágít arra, mekkora különbségek vannak a különböző típusú projektek között, és hogy milyen nehéz az egyik alkalmazási területen megszerzett tapasztalatokat egy másikban hasznosítani.

A képletet a COCOMO II modell fejlesztői javasolták, az ő tapasztalataikat és adataikat tükrözi, de véleményem szerint túl bonyolult ahhoz, hogy a gyakorlatban is alkalmazható legyen: túl sok jellemzőt használ, és az attribútumok értékeinek becslésénél túl nagy teret enged a bizonytalanságnak. Elméletileg a modell minden felhasználójának a saját régebbi projektjeinek az adatai szerint kellene beállítania a modellt és a jellemzők értékeit, hogy nagyobb szerephez jussanak a modellre ható helyi körülmények.

Azonban a gyakorlatban kevés szervezetnek van elég összegyűjtött információja a korábbi projektjeiről olyan formában, amely segítené a modell beállítását. A COCOMO II gyakorlati alkalmazását a modell paramétereinek ismert értékei-

vel kellene elkezdni, de lehetetlen, hogy a felhasználó tudja ezek pontos viszonyát a saját esetéhez. Ez viszont azt jelenti, hogy a COCOMO II modell gyakorlati alkalmazhatósága korlátozott. A nagyon nagy szervezetek rendelkeznek csak annyi erőforrással, hogy alkalmazni tudjanak egy költségszakértőt a COCOMO II modell adaptálására és használatára. Így a szervezetek többsége számára az olyan algoritmikus modelleknek, mint a COCOMO, a beállítása és megtanulása drága, ezért valószínűleg nem vezetik be.

26.3.2. Algoritmikus költségmodellek a projekttervezésben

Az algoritmikus költségmodellezés egyik leghatékonyabb módja a projektköltségek csökkentése céljából tett pénzbefektetések különböző módjainak összehasonlítása. Ez különösen fontos ott, ahol kompromisszumot kell kötni a hardver és a szoftverkölségek között, és ahol szükség lehet speciális szaktudású emberek toborzására a projekthez. Az algoritmikus modell segít mindegyik opció kockázatainak felmérésében. A költségmodell kimutatja a különböző vezetői döntésekhez kapcsolódó pénzügyi hátteret.

Vegyünk egy olyan beágyazott rendszert, amely egy úrkísérletet irányít. Az úrben végzett kísérleteknek nagyon megbízhatónak kell lenniük, és szigorú súlykorlátokat kell betartani. A COCOMO-modell alapján szemlélve, a számítógép által tett megszorításokon és a megbízhatóságán alapuló szorzók magasabbak, mint 1.

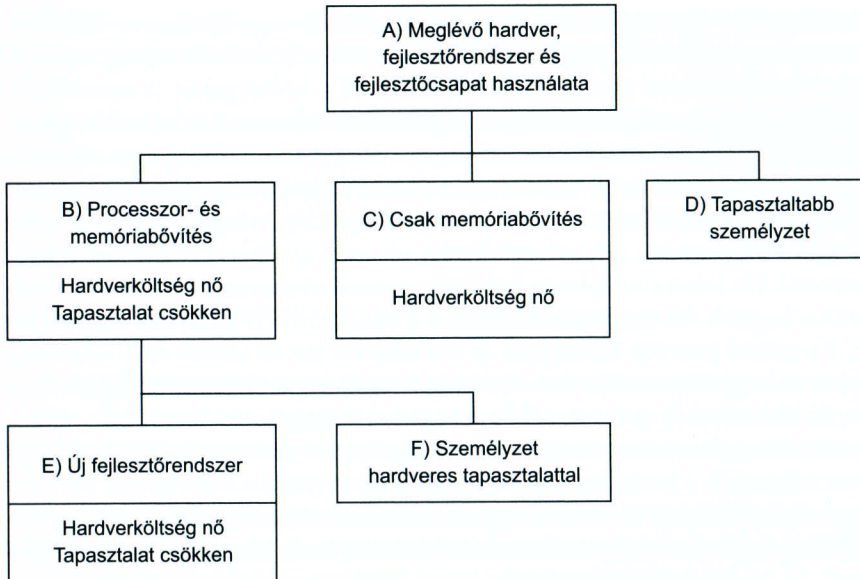
A projekt költségbecslésénél három összetevőt kell figyelembe venni:

1. a rendszer futtatását végző célhardver költsége;
2. annak a platformnak (számítógép és szoftver) a költsége, amelyen a rendszert fejlesztik;
3. a szoftver kifejlesztéséhez szükséges munka költsége.

A 26.13. ábra azokat a választási lehetőségeket mutatja, amelyeket fontolóra kell venni. A választási lehetőségek közé tartozik a szoftverkölség csökkentése érdekében kiadott nagyobb összeg a célhardverre vagy a jobb fejlesztőeszközökhöz való befektetés.

A példa esetében elfogadható lehet a magasabb hardverkölség, mert a rendszer egy specializált rendszer, amely nem kerül tömegtermelésre. Azonban ha a vásárló a hardvert is megveszi a termékkel együtt, akkor ritkán szerencsés, ha a szoftverkölség csökkentése érdekében a célhardverbe ruháznak be, mert ez növeli a termék egységárát.

A 26.13. ábrán bemutatott A–F opciók hardver-, szoftver- és teljes költsége a 26.12. ábrán látható. A költségtényezők nélkül alkalmazott COCOMO II modell szerint 45 emberhónapnyi munka kell arra, hogy a fenti alkalmazáshoz beágyazott szoftverrendszert fejlesszenek ki. Egy emberhónapnyi munka átlagköltsége 15 000 dollár.



26.12. ábra. A menedzsment lehetőségei

Opció	RELY	STOR	TIME	TOOL	LTEX	Teljes munka	Szoftver-költség	Hardver-költség	Teljes költség
A	1,39	1,06	1,11	0,86	1	63	949 393	100 000	1 049 393
B	1,39	1	1	1,12	1,22	88	1 313 550	120 000	1 402 025
C	1,39	1	1,11	0,86	1	60	895 653	105 000	1 000 653
D	1,39	1,06	1,11	0,86	0,84	51	769 008	100 000	897 490
E	1,39	1	1	0,72	1,22	56	844 425	220 000	1 044 159
F	1,39	1	1	1,12	0,84	57	851 180	120 000	1 002 706

26.13. ábra. A menedzsmentlehetőségek költsége

A megfelelő szorzók a tárra és végrehajtási időre tett korlátokon (TIME és STOR), a fejlesztői rendszerhez (TOOL) és a fejlesztőcsapat platformon szerzett tapasztalataihoz (LTEX) rendelkezésre álló eszköztámogatáson (keresztfordítók stb.) alapulnak. A megbízhatósági szorzó (RELY) minden opciónál 1,39, jelezve, hogy megbízható rendszerek fejlesztéséhez jelentős pluszmunka szükséges.

A szoftverköltséget (SC) a következőképpen számítják:

$$SC = \text{Becsült munka} \times \text{RELY} \times \text{TIME} \times \text{STOR} \times \text{TOOL} \times \text{EXP} \times 15\,000 \$$$

Az A lehetőség a meglévő támogatással és résztvevőkkel létrehozott rendszerek költségét jelenti, a többi lehetőség költségét ehhez hasonlítjuk. Az összes

többi lehetőségnél vagy nagyobbak a hardverkiadások, vagy új résztvevőket kell toborozni (az ezzel járó költségekkel és kockázatokkal). A B lehetőség azt mutatja, hogy a jobb hardver nem feltétlenül csökkenti a költségeket. A személyzet gyakorlatának hiánya az új hardveren megnöveli a tapasztalati szorzót, így elvész a STOR és TIME szorzónál szerzett előny. Valójában a költségekre inkább a tár növelése van jobb hatással, mint a teljes számítógép-konfiguráció fejlesztése.

Az összes alapbecslés közül a D lehetőség szolgáltatja a legalacsonyabb költséget: itt nincs megemelve a hardverköltség, viszont új résztvevőket kell a projektbe bevonni. Ha lehet a vállalatban belülről embereket szerezni, akkor ez a lehetőség tűnik a legjobb választásnak, különben a vállalatban kívülről kell embereket szerezni. Ez utóbbi jelentős költséggel és kockázattal jár, és emiatt ez a lehetőség nem is biztos, hogy olyan előnyös a költség szempontjából, mint ahogyan az a 26.13. ábrán látható volt. A C lehetőséggel gyakorlatilag kockázatvállalás nélkül majdnem 50 000 dollár takarítható meg, ezért a konzervatív vezetők lehet, hogy inkább ezt választják a kockázatosabb D lehetőség helyett.

Az összehasonlításból látható, hogy szorzóként milyen fontos a résztvevők gyakorlata: a projektköltség jelentősen csökkenhet, ha a megfelelő gyakorlattal rendelkező, jó képességű embereket válogatják össze, és ez megegyezik azzal, amit a 26.1. alfejezetben a termelékenységi tényezők kapcsán írtam le. Az is látható, hogy ha új hardverbe és eszközökbe ruháznak be, akkor ez negatívan is hathat a költségekre. Az ilyen stratégiákat azok a fejlesztők támogatják, akik szeretnek új rendszerekkel dolgozni, viszont a rendszerköltséget sokkal érzékenyebben érinti a tapasztalatlanság, mint az új hardverrendszer miatti megtakarítás.

26.4. A PROJEKT IDŐTARTAMA ÉS RÉSZTVEVŐI

A projektvezetőknek a szoftverrendszer kifejlesztéséhez szükséges munka és összköltség becslése mellett azt is fel kell mérniük, mennyi idő kell a szoftver kifejlesztésére, és mikor lesz szükség a résztvevők munkájára a projekthez. A projekt fejlesztési idejét projektütemezésnek hívják. A szervezetek egyre rövidebb fejlesztési ütemezést követelnek meg, így termékeikkel versenytársaikat megelőzve tudnak megjelenni a piacon.

A projekten dolgozó résztvevők száma, a szükséges összes munka és a fejlesztési idő közötti kapcsolat nemlineáris. A résztvevők számának növekedésével több munkára lehet szükség, az embereknek több idő kell a kommunikációra, és több idő kell a rendszer részei közötti interfészek definíálásához. A résztvevők számának megkétszerezése például nem jelenti azt, hogy a projekt időtartama a felére csökken.

A COCOMO-modell tartalmaz egy képletet egy projekt befejezéséhez szükséges naptári idő (TDEV) becslésére, amely a modell minden szintjén azonos:

$$TDEV = 3 \times (PM)^{(0,33 + 0,2 * (B - 1,01))}$$

A PM a munka kiszámítása, a B a fenti módon kiszámított kitevő (értéke 1 a korai prototípus-készítési szinten). A számítás a projekt névleges ütemezését jelzi előre.

Az előre jelzett projektütemezés és a projekttervben megkívánt ütemezés azonban nem feltétlenül egyezik meg, mert a tervezett ütemezés hosszabb vagy rövidebb ideig is eltarthat, mint a névleges előre jelzett ütemezés. Nyilvánvaló, hogy az ütemezés ezen változásának mértéke behatárolható, ami a COCOMO II modell szerint:

$$TDEV = 3 \times (PM)^{(0,33 + 0,2 \cdot (B - 1,01))} \times SCEDPercentage/100$$

A SCEDPercentage a névleges ütemezés százalékos csökkenése vagy növekedése. Ha az előre jelzett érték nagyon eltér a tervezett ütemezéstől, akkor ez arra hívja fel a figyelmet, hogy nagy a kockázat, és problémák merülhetnek fel a szoftver leszállításának tervezett idejével kapcsolatban.

A COCOMO fejlesztésütemezési számításának szemléltetésére tegyük fel, hogy egy szoftverprojekthez 60 hónap fejlesztői munkát becsültek (a C lehetőség a 26.12. ábrán), továbbá tegyük fel, hogy a B kitevő értéke 1,17.

Ha ezeket behelyettesítjük az ütemezési egyenlőségbe, akkor azt kapjuk, hogy a projekt befejezéséhez szükséges idő

$$TDEV = 3 \times (60)^{0,36} = 13 \text{ hónap}$$

Ebben az esetben az ütemezés nincs se összesűrítve, se szétnyújtva, így a kép-let utolsó kifejezése egyáltalán nem befolyásolja a számítást.

A COCOMO-modellből azt az érdekes következtetést lehet levonni, hogy a projekt befejezéséhez szükséges idő a projekthez szükséges teljes munka függvénye, és nem függ a projekten dolgozó szoftvermérnökök számától. Ez alátámasztja azt az elképzelést, hogy az elhúzódo projekteknel a résztvevők számának növelése nem biztosíték arra, hogy behozzák a lemaradást az ütemtervhez képest. Az ütemezés gyorsításának kérdésével Myers (Myers, 1989) foglalkozott. Szerinte a projektek jelentős problémákba ütközhetnek, ha úgy próbálnak szoftvert fejleszteni, hogy nem áll elegendő idő a rendelkezésükre.

A projekthez szükséges munka és a fejlesztési ütemezés hanyadosa nem alkalmas a projektcsapat szükséges létszámának jelzésére. A szoftverprojekteken alkalmazott emberek létszáma kezdetben rendszerint viszonylag alacsony, később növekedve elér egy pontig, majd csökken. A projekt kezdetén a tervek és a specifikáció elkészítéséhez nincs szükség sok emberre. Ahogyan a projekt halad előre és egyre aprólékosabb munkát várnak el, a résztvevők száma növekszik, míg el nem ér egy csúcst. Az implementáció és a tesztelés befejeztével a résztvevők száma csökkenni kezd, egészen addig, hogy a termék szállításakor már csak egy-két személy alkotja a csapatot. A projektlétszám nagyon gyors felduzzasztása gyakran a projektütemezés csúszásával kapcsolható össze, ezért a projektvezetőknek tartózkodniuk kell attól, hogy a projekt élettartamának korai szakaszában túl sok résztvevővel bővítsék a csapatot.

A munka felfutása az úgynevezett Rayleigh-görbével (Londeix, 1987) modellezhető, és Putnam becslési modellje (Putnam, 1978) is tartalmaz egy ezekre a görbékre alapozott projektrésztvevői modellt. Putnam modellje a fejlesztési időt mint kulcstényezőt is tartalmazza: csökkentésével exponenciálisan nő a rendszer kifejlesztéséhez szükséges munka.

Kulcsfogalmak

- Nincs szükségszerű kapcsolat a rendszer ára és a fejlesztési költsége között. Egy szervezeti tényező jelentheti azt, hogy a kért magasabb ár ellensúlyozza a magasabb kockázatot, vagy az alacsonyabb viszont versenyelőnyt jelenthet.
- A termelékenységre ható tényezők közé tartoznak: az egyéni adottság (mint domináns tényező), a szakterület ismerete, a fejlesztési folyamat, a projekt mérete, az eszköztámogatás és a munkakörnyezet.
- A szoftverek árát gyakran úgy állapítják meg, hogy ezzel szerződéshez jussanak, majd ezután ehhez a becsült árhoz igazítják a rendszer funkcionalitását.
- A szoftver költségbecslésének különböző technikái vannak. Egy becslés létrehozásához eltérő technikák használhatók. Ha a becslések nagyon eltérnek, akkor az azt jelenti, hogy a becsléshez rendelkezésre álló információk nem megfelelőek.
- A COCOMO II költségelemzési modell egy jól kifejlesztett algoritmikus modell, amely a költségbecslés elkészítésénél figyelembe veszi a projekt, a termék, a hardver és a résztvevők jellemzőit, és a fejlesztések ütemezésének becslésére is tartalmaz eszközt.
- Az algoritmikus költségmodellek értékesek a vezetés számára, mivel támogatják a mennyiségi opcióelemzést. Segítségükkel több különböző lehetőség költsége kiszámítható, és ha ezek a számítások esetleg hibásak is, a lehetőségeket tárgyilagosan össze lehet hasonlítani.
- A projekt befejezéséhez szükséges idő nem egyszerűen a projekten dolgozó emberek számával arányos. Egy projekt elhúzódó befejezését tovább késleltetheti, ha még több embert állítanak rá.

További irodalom

- „The unmyths of project estimation”. Egy pragmatikus cikk a projektbecslés gyakorlati nehézségeiről és a terület néhány alapvető hipotéziséről. (Armour, P., *Comm. ACM*, 45(11), November 2002.)
- Software Cost Estimation with COCOMO II*. Alapkönyv, a modell komplett leírása, sok példával és szoftverrel az implementációhoz. Rendkívül részletes, nem könnyű olvasmány. Boehm másik könyve (lásd alább) véleményem szerint, könnyebb bevezetés a modellbe. (Boehm, B. és mások, 2000, Prentice Hall.)
- Software Project Management: Readings and Cases*. Válogatott cikkek és esettanulmányok a szoftverprojekt-menedzsment területéről, amely különösen mélyen

foglalkozik az algoritmikus költségmodellezés témakörével. (Kemerer, C. F. (ed.), 1997, Irwin.)

„Cost models for future software life cycle processes: COCOMO II”. Átfogó bevezetés a COCOMO II költségbecslési modellbe, amely tartalmazza a formulák magyarázatát is. Könnyebben olvasható, mint az alapkönyv. (Boehm, B. és mások, *Annals of Software Engineering*, 1, Balzer Science Publishers, 1995.)

Feladatok

- 26.1. Milyen körülmények között kérhet egy vállalat magasabb árat egy szoftverrendszerért, mint ami egy költségbecslés és egy normál profit összegéből következik?
- 26.2. Írjon le két olyan metrikát, amellyel mérhető a programozók termelékenysége. Írja le röviden előnyeiket és hátrányaikat.
- 26.3. Nevezzen meg öt olyan tényezőt nagy, beágyazott, valós idejű rendszerekhez, amelyek jelentős hatással lehetnek a szoftverfejlesztő csapat termelékenységére.
- 26.4. A költségbecslések velejárója a kockázat, bármilyen becslési technikát alkalmazzanak is. Javasoljon négy módot arra, hogyan lehetne csökkenteni a költségbecslési kockázatot.
- 26.5. Miért használhatók egyes becslési technikák a nagy, komplex szoftverrendszerek költségbecslésére?
- 26.6. Egy szoftvermenedzsert olyan nagy biztonságú szoftverrendszer fejlesztésével bíznak meg, amelyet rákos betegek kezelésére szolgáló sugárterápiás gép irányítására terveztek. A rendszer a gép részét képezi, és speciális célú, rögzített méretű memóriával (8 MB) rendelkező processzoron kell futnia. A gép kapcsolatban áll a betegek adatait tároló adatbázisrendszerrel: innen szerzi az információkat a betegekről, és a kezelés után automatikusan ide jegyzi be a besugárzás mennyiségét és a kezelés további részleteit.

A rendszer kifejlesztéséhez szükséges munka becslésére a COCOMO-modellt használták. A becslésnél minden költség tényező értékét 1-nek vették, és így a szükséges munkára 26 emberhónapot számítottak ki.

Fejtse ki, miért kellene a becslést úgy igazítani, hogy az számításba vegye a projekt tényezőket, a terméktényezőket, a személyzeti és szervezeti tényezőket. Nevezzen meg négy olyan tényezőt, amelyeknek fontos hatása lehet a kezdeti COCOMO-becslésre, és javasoljon lehetséges értéket ezeknek a tényezőknek. Indokolja meg, miért ezeket a tényezőket választotta.

- 26.7. Soroljon fel három okot, hogy miért nem hasonlíthatók össze közvetlenül a különböző szervezeteknél készített algoritmikus költségbecslések.
- 26.8. Magyarázza el, hogy a projektvezetők hogyan alkalmazhatják a költségbecslés algoritmikus megközelítését opcióelemzésre. Ismertessen egy olyan helyzetet, ahol a vezetőknek nem a legalacsonyabb projektköltségen alapuló megközelítést kellene választaniuk.

- 26.9. Néhány nagyon nagy szoftverprojektben több millió soros kódok készülnek. Fejtse ki, hogyan lehet hasznosítani a költségbecslési modelleket ezeknél a rendszereknél. Nagyon nagy szoftverrendszerekre miért nem lehetnek érvényesek ezek alapfeltevései?
- 26.10. Etikus-e, ha egy vállalat, tudván, hogy a követelmények nincsenek egyértelműen megállapítva, alacsony árat ad meg a szoftverszerződésben, majd magas árat kér a vásárló utólagos kívánságára készített változtatásokért?
- 26.11. Fel kellene-e a használni a vezetőknek a mért termelékenységet a résztvevők értékelésének folyamatában? Milyen óvintézkedésekkel biztosítható, hogy ez ne befolyásolja a minőséget?