16. Икономика на софтуерното производство според Боем

1. Необходимост и цели

Всеки ръководител на софтуерен проект би желал във възможно най-ранен момент от ЖЦ на разработвания ПП да знае колкото се може по-точно какви разходи ще трябва да се направят и колко ще продължи целият процес.

С този проблем теоретиците започват да се занимават още през 60-те години.

В резултат се появява първият метод за оценяване разходите по произвеждането на даден ПП – SDC, 1965 г.

Последват го още няколко метода, основани на съответни модели.

През 1981 година се появява COCOMO, предложен от Boehm (COCOMO II - 2000).

Тази разработка се счита за фундаментална защото:

– намира реално приложение в практиката,

- поставя на здрави основи изследванията по цената и по-общо – по икономиката на разработването на софтуер.

Оказва се обаче, че процесът на намиране цената на разработване води до резултати, полезни и в други насоки:

- **избор на проект за реализация – един от решаващите фактори при такъв избор е цената на бъдещата разработка;**

- **определяне състава на колектива разработчик - също е в пряка зависимост от установената цена и обратно – изменяйки състава (по брой, квалификация, опит и др.) можем пряко да влияем върху цената на разработката;**
- **определяне на маркетинговата политика по отношение разработвания софтуер – както е известно цената е един от основните компоненти на т.н. маркетингов микс;**
- **оценяване работата на членовете на колектива - това може да става на основата на сравнение на предварително получените оценки с крайните резултати от работата.**

2. Критерии

Един от важните приноси на Boehm е, че формулира множество от критерии, които следва да удовлетворява даден модел, респ. метод, за установяване цената на разработване на софтуерни продукти.

Критериите, формулирани от Boehm, са следните.

2.1. Определеност

Означава точно и обективно определяне на началните данни и понятия, както и на крайните резултати и количествените характеристики.

Доколкото всеки метод на оценка се основава на някакъв модел на ЖЦ, то е много важно да се знае:

- кои са точно фазите му,
- какво е тяхното съдържание,
- какви функции се изпълняват по време на всяка фаза и
- в какво точно се състои всяка от тях.

Смята се, че една от причините моделът COCOMO да е получил широка известност и приложение е тъкмо отличното спазване на този критерий.

2.2. Точност

Има се предвид съответствието между предсказаната от модела цена на разработване и реално получилата се накрая.

Наред със същността на метода от голямо значение е определеността и точността на входните данни, както и квалификацията на експертите, които правят оценката.

Има различни начини за повишаване на точността.

Колкото и авторите да се опитват да направят модела и метода си универсален, по правило той работи най-добре за определен клас програмни продукти.

При прилагането му извън този клас точността на резултатите рязко се влошава.

Тогава се прави опит за настройка – промяна на базовите уравнения, добавяне на нови фактори, премахване на нерелевантни такива.

Има изследвания, които показват недобра точност на много методи дори при прилагането им към "подходящи" за метода продукти.

Едно от тях е на основата на 15 големи проекта с прилагането на 4 различни модела.

При него се оказва, че при определянето продължителността на няколко (вече завършени) проекта грешката се е движила между 85% и 772%.

Макар и не така разочароващи, грешки са показани и в точността на предсказването на цената на разработвания софтуер.

2.3. Обективност

Важен критерий е постигането на възможно най-голяма обективност, т.е. максимално избягване на субективния фактор.

Навсякъде, където се очаква лична оценка от страна на експерти:

- **изготвят се предварително указания,**
- **въвеждат се тегла,**
- **тяхното определяне се формализира в рамките на възможното,**
- **предоставят се еталони.**

2.4. Детайлност

Колкото един модел е по-подробен, толкова той е по-адекватен на реалните обекти и процеси.

Тогава основаният на него метод да даде по-точни резултати.

Обаче по-дълбоката детайлност изисква:

- повече ресурси (време, хора, пари) в етапа на разработване на модела и

- при прилагането му в конкретни случаи.

При това положение се търси компромис между цената на извършваната оценка и желаната степен на точност.

В модела COCOMO този проблем е решен с помощта на 3 версии на модела – базова, междинна и детайлна.

2.5. Устойчивост

Този критерий е въведен от Boehm с цел отделяне на неподлежащи на оценка разработки поради съществуващи граници на приложение на метода.

Така например, неустойчив се оказва моделът DOTY.

При него започват да се наблюдават силни отклонения в оценките при продукти с над 10 000 реда първичен код.

Самият Boehm признава, че неговият модел COCOMO също може да се покаже като неустойчив при оценяване на малки проекти – до 2 000 реда първичен код.

Една от причините за това явление е, че всички модели са създавани на основата на изследване на големи и много големи проекти.

Уравненията, чрез които се пресмятат оценките се базират на тях.

Впрочем има правен опит – моделът P3 (Programmer's Personal Planner), предназначен за относително малки проекти – до 18 000 първични реда и до колектив от максимум 3 души.

Подобен модел представлява особен интерес днес, при непрекъснато растящите нужди от малки програмни продукти за персоналните компютри.

2.6. Област на приложение

Няма универсален модел за определяне цената и продължителността на разработване на софтуерен продукт.

Следователно, за всеки модел трябва да може ясно да се определи областта на приложение.

Така например PRICE S е предназначен за оценка на аерокосмически софтуер.

Знае се, че методът на функционалните точки работи най-добре за информационни системи и бизнес приложения.

Областта на приложение на някои модели е по-широка поради възможността някои коефициенти в уравненията за оценка да се променят предварително – пример COCOMO.

2.7. Конструктивност

Всеки модел трябва да дава възможност да бъдат анализирани и разбрани получените резултати.

В известни случаи даден модел води до по-ниски или по-високи резултати от реално получените накрая.

Винаги обаче трябва да бъде възможно тези отклонения да бъдат обяснени.

Целта е конструктивна – подобряване на модела.

Към тази характеристика спада и още една особеност.

Моделът трябва да бъде конструктивен и в смисъл, че допуска получаването на различни резултати в зависимост от това на какъв фактор потребителят придава по-голямо значение.

С други думи моделът може да предлага различни алтернативи според това дали се държи на скъсяване на сроковете (разбира се в допустимите граници) за сметка на ползването на повече или по-скъпа работна сила или пък точно обратното.

2.8. Простота на прилагане

Този критерий определя:

- **степента на трудност на разбирането и получаването на входните данни и**
- **степента на трудност на изпълнение на процедурите по оценяване.**

По принцип по-добра точност се получава при по-голяма детайлност, която обикновено се реализира йерархично.

В този случай обаче нараства значително трудоемкостта, а и множеството фактори също усложняват прилагането на метода.

Ясно е, че става дума за критерий, който е в пряка зависимост с други критерии – точност, определеност.

Тази бележка впрочем може да се обобщи – десетте разглеждани критерии са в голяма степен зависими и често подобрение в един от тях води до влошаване в други.

2.9. Предсказуемост

Този критерий засяга проблема за практическото използване на моделите.

Ясно е, че стремежът на всеки потребител е да получи възможно най-точна оценка колкото се може по-рано в процеса на разработване на софтуер.

По-долу, при разглеждането на модела COCOMO ще видим, че той се основава на броя редове първичен код.

За съжаление това не е толкова лесно предсказуем параметър, т.е. дори и много голям опит не може да гарантира сравнително точно определяне на този брой в началната фаза на жизнения цикъл на софтуерния продукт.

Значителна част от изследователските усилия са били и продължават да бъдат съсредоточени тъкмо на този проблем: като входни данни - основа на съответния метод да се избере точно и ранно предсказуем фактор.

2.10. Икономичност

Това изискване е очевидно, но и то като много от вече изброените има компромисен характер.

Желателно е:

- входните данни за оценката да бъдат по-малко на брой и лесно измерими, но и
- да не са прекалено малко, така че да доведат до фатална неточност на крайните резултати.

С други думи – желателна е икономичност, но не и на всяка цена.

При отделни модели е забелязано, че когато се преминава от една област на приложение към друга някои от факторите се оказват в новата област толкова взаимосвързани, че отпадането на някои от тях става възможно.

По този начин се повишава икономичността на модела.

3. Моделът на Boehm COCOMO

3.1. Цели и основни идеи

COCOMO произлиза от Constructive Cost Model.

Основната му цел е за всеки планиран софтуерен проект да се оцени цената и срока на разработване.

Основополагащите му идея е използването броя редове първичен код.

Първоначално предложеният модел е усъвършенстван в различни направления – чрез въвеждане на 3 нива на подробност, чрез отличаване на 3 типа на разработване, чрез въвеждане на коригиращи коефициенти за определящите параметри.

3.2. Същност на модела

За оценяване на трудоемкостта на даден софтуерен проект се прилага формулата:

$$ЧМ = 2.4 \times ХРПК^{1.05}$$

където ЧМ означава брой човекомесеци

ХРПК означава хиляди реда първичен код

За оценяване продължителността на разработване на софтуерния проект формулата е:

$$В = 2.5 \times ЧМ^{0.38}$$

където В е срокът на разработване в месеци

Формулите се прилагат при следните предположения:

а) редовете първичен код (т.е. тези, които се пишат на някакъв език за програмиране):

- се броят без коментарните редове

- принадлежат на крайния продукт (а не на негови междинни версии)

- не включват използваните стандартни програми;

б) включват се само фазите проектиране, програмиране и оценка, включително усилията по управлението и документирането по време на тези фази;

в) не се включват обучението, планирането и инсталирането на софтуера при потребителя;

г) включва се трудът на проектантите и програмистите, но не и този на компютърните оператори, висшите ръководители и секретарките;

д) счита се, че един човекомесец е от 19 дни или 152 часа;

е) предполага се, че никакви сериозни промени не се правят в продукта след одобряването на документа, който съдържа изискванията към него;

ж) двете страни – потребителят и разработчикът – се предполага, че са добросъвестни през цялото време.

3.3. Пример и следствия

За илюстрация да дадем един прост пример.

Да предположим, че в резултат на предварителна експертиза бъдещият софтуерен продукт се оценява на 32 000 реда първичен код.

Като приложим двете формули, ще получим следните числови резултати:

ЧМ = 2.4 x $32^{1.05}$ = 91 (човекомесеца)

В = 2.5 x $91^{0.38}$ = 14 (месеца)

От тези базови резултати могат да се получат още две важни характеристики:

Производителност: 32 000 / 91 = 352 реда първичен код за човекомесец

Екип: 91 / 14 = 6.5 човека.

Тълкуването на последната бройка е ясно – един или повече специалисти от екипа ще бъдат заети с проекта не през всичките 14 месеца на разработката.

Правени са изследвания за това **как се изменя производителността с промяната на размера на проекта**.

Boehm предлага **условно разделяне на проектите** на **малки, междинни, средни и големи**.

За тези 4 категории са получени следните данни.

Тип проект	ХРПК	ЧМ	В – месеци	Екип - брой	Производителност
Малък	2	5.0	4.6	1.1	400
Междинен	8	21.3	8.0	2.7	376
Среден	32	91.0	14.0	6.5	352
Голям	128	392.0	24.0	16.0	325

3.4. Усъвършенстване на модела

Първото усъвършенстване на дотук изложения базов модел е въвеждането на 3 типа софтуерни проекти

- **разпространен (organic),**
- **полунезависим (semidetached) и**
- **вграден (embedded).**

За всеки от тях формулата е различна.

В следващата таблица всеки тип се характеризира кратко, илюстрира се с примери и му се съпоставя предложената от Boehm формула.

Тип	Характеристика	Примери	Формула
Разпространен	Разработва се от малка група в познатите условия на собствената фирма	Прости системи за управление на запаси или прости производствени процеси	ЧМ = 2.4 x ХРПК$^{1.05}$
Полунезависим	Има междинно положение между разпространен и вграден тип	Системи за обработка на съобщения	ЧМ = 3.0 x ХРПК$^{1.12}$
Вграден	Софтуерът работи свързан с апаратура, друг софтуер и изчислителни процеси	Авиационни или радиоелектронни системи	ЧМ = 3.6 x ХРПК$^{1.20}$

Следващото усъвършенстване е свързано с установяването на факта, че все пак редовете първичен код не биха могли да са единственият параметър на установената формула.

Преминава се към по-сложни модели, в които се отчитат допълнителни фактори.

Тези модели са два вида - междинен (intermediate) и детайлен (detailed).

Част от факторите (те са общо 15), заедно с възможните им рейтинги, са дадени в следващата таблица.

Оценявани атрибути	Много нисък	Нисък	Нормален	Висок	Много висок	Свръх висок
Сложност на продукта	0.70	0.85	1.00	1.15	1.30	1.65
Квалификация на програмистите	1.42	1.17	1.00	0.86	0.70	
Прилагане на съвременни методи	1.24	1.10	1.00	0.91	0.82	

При извършването на оценката експертите следва да се ръководят от тази таблица и за конкретния проект трябва да определят за всеки атрибут съответния рейтинг.

Получените стойности се заместват в формулата, която вече е придобила по-сложен вид:

ЧМ = к х ХРПК^{p} х A_1 х A_2 . . . х A_{15}

където ЧМ и ХРПК са вече познатите величини,

к и p са коефициенти, които са различни за различните типове софтуер, които видяхме

(k=2.4, 3.0, 3.6; p=1.05, 1.12, 1.20).

A_I са рейтингите на атрибутите

3.5. Cocomo II - 2000

$$PM = \prod_{i=1}^{17} (EM_i) \cdot A \cdot \left[\left(1 + \frac{REVL}{100}\right) \cdot Size\right]^{\left(0.91 + 0.01 \sum_{j=1}^{5} SF_j\right)} + \left(\frac{ASLOC \cdot \left(\frac{AT}{100}\right)}{ATPROD}\right)$$

where

$$Size = KNSLOC + \left[KASLOC \cdot \left(\frac{100 - AT}{100}\right) \cdot \frac{AA + SU + 0.4 \cdot DM + 0.3 \cdot CM + 0.3 \cdot IM}{100}\right]$$

$$B = 0.91 + 0.01 \sum_{j=1}^{5} SF$$

Estimate effort with:

Symbol	Description
A	Constant, currently calibrated as 2.45
AA	Assessment and assimilation
ADAPT	Percentage of components adapted (represents the effort required in understanding software)
AT	Percentage of components that are automatically translated
ATPROD	Automatic translation productivity
REVL	Breakage: Percentage of code thrown away due to requirements volatility
CM	Percentage of code modified
DM	Percentage of design modified
EM	Effort Multipliers: RELY, DATA, CPLX, RUSE, DOCU, TIME, STOR, PVOL, ACAP, PCAP, PCON, APEX, PLEX, LTEX, TOOL, SITE
IM	Percentage of integration and test modified
KASLOC	Size of the adapted component expressed in thousands of adapted source lines of code
KNSLOC	Size of component expressed in thousands of new source lines of code
PM	Person Months of estimated effort
SF	Scale Factors: PREC, FLEX, RESL, TEAM, PMAT
SU	Software understanding (zero if DM = 0 and CM = 0)

3.6. Критика на “редове първичен код”

Основните критики към COCOMO се свеждат до следните две:

- няма нито стандарт, нито единно виждане за това какво е “ред първичен код”.

- много е трудно, дори за експерти с голям опит да предскажат достатъчно точно в ранен етап на разработването броя на редовете първичен код.

По въпроса за редовете първичен код се изтъкват следните проблеми.

1. Какво всъщност да се разглежда – физически или логически редове.

Физическият край на даден ред се получава с натискането на клавиша Enter, което води до генериране на съответни служебни символи за край на реда.

Логическият край е някакъв ограничител, зависещ от конкретния език за програмиране, например двоеточие, запетая или друг точно определен знак.

Езици, които позволяват написването на няколко оператора на един ред, могат да дадат неколкократна разлика при двата вида броене.

Същото може да се случи в обратна посока и когато (за прегледност или по други причини) един логически оператор се разполага върху 2 или повече редове.

Показателно е едно изследване, което показва, че в САЩ:

- **35% от ръководителите на проекти броят физическите редове,**
- **15% броят логическите,**
- **останалите 50% въобще не считат за уместно да борят редовете първичен код.**

2. Друг елемент на несигурност внасят различните от семантична гледна точка типове редове.

Почти всеки процедурен език включва 4 типа първични редове:

- **изпълними оператори, чрез които се задават различни операции (логически, аритметични, вход/изход и др.),**
- **определения на данни, чрез които се дефинират различните типове данни,**
- **коментари, които дават разясняваща информация,**
- **празни редове, които се ползват за повишаване прегледността на програмата.**

Установено е, че в дадено типично бизнес приложение:

- **40% от общия брой оператори са изпълними,**
- **35% са определения на данни,**
- **15% са коментари и**
- **10% - празни редове.**

При системния софтуер (например операционни системи):

- **45% са изпълними оператори,**
- **30% са определения на данни и отново**
- **15% са коментари и**
- **10% празни редове.**

Правени са изследвания сред създателите на софтуер и отново е установено различие в разбиранията:

- 10% броят само изпълнимите редове,
- 20 % броят изпълнимите редове и определенията на данни,
- 15% добавят коментарите,
- 5% броят дори и празните редове.

Както вече беше казано, 50% въобще не броят редовете първичен код.

3. Голям проблем е повторно използваемият код.

По съвсем общи оценки един програмист на традиционните процедурни езици за програмиране Фортран, КОБОЛ, С средно копира от други програми между 20 и 30% код в своите програми.

За обектно ориентирани езици като Smalltalk, C++ този процент достига до 50.

Има отделни софтуерни фирми, които са организирали специални библиотеки от модули за повторно използване и там въпросният процент достига дори 75.

При това положение спорът се върти около това как да се брои даден повторно използван модул:

- да се брои при всяко негово появяване;
- да се брои само веднъж, независимо от броя появявания
- въобще да не се брои, доколкото този модул не е разработван за оценявания проект.

Изследванията показват, че в САЩ:

първата алтернатива се прилага от 25% от професионалистите,

втората – от 20%,

третата – от 5%.

За останалите 50% вече се разбра, че не броят.

4. Друга област на несигурност е как да се ползват редовете първичен код при приложения, писани на повече от един език.

Има данни, че около една трета от софтуера в САЩ е написан на повече от един език за програмиране.

Колкото и да изглежда невероятно, дори около 1996 година най-срещаните комбинации от езици все още са били следните:

- КОБОЛ заедно с някакъв език за обработка на заявки за търсене от типа на SQL;
- КОБОЛ и език за дефиниция на данни от типа на DL/1;
- КОБОЛ и някакъв специализиран език;
- С и Асемблер;
- АДА и Асемблер.

Доколкото, както стана вече ясно, няма стандарти и единство дори при броенето в приложения с един език, още по-малко може да се очаква такова единство при софтуер, написан на повече от език.

5. Има и някои допълнителни по-малки проблеми от рода на това да се включва или изключва:

- временно поставен в програмата код,
- код от типа макрос,
- код, свързан с управлението на заданието в рамките на операционната система.

Трудности предизвиква и разликата в стила на програмистите.

Преди време в IBM направили експеримент, като възложили на 8 програмиста да напишат първичен код по зададена спецификация.

Разликата в броя редове първичен код (преброени естествено по една и съща фиксирана методика) между най-големия и най-малкия била петкратна.

Независимо от критиката по горните 5 пункта, моделът COCOMO продължава да има своите привърженици и да се прилага.

Освен това са разработени уточняващи схеми.

Пример:

Definition Checklist for Source Statement Counts

Definition name: *Physical Source Lines of Code (basic definition)* Date: *8/7/92*

Originator: *SEI*

Measurement unit: **Physical source lines** ✔ / **Logical source statements**

				Includes	Excludes
Statement type	**Definition** ✔	**Data array**			
When a line or statement contains more than one type, classify it as the type with the highest precedence.					
1 Executable		**Order of precedence ->**	1	✔	
2 Nonexecutable					
3 Declarations			2	✔	
4 Compiler directives			3	✔	
5 Comments					
6 On their own lines			4		✔
7 On lines with source code			5		✔
8 Banners and nonblank spacers			6		✔
9 Blank (empty) comments			7		✔
10 Blank lines			8		✔
11					
12					
How produced	**Definition** ✔	**Data array**		**Includes**	**Excludes**
1 Programmed				✔	
2 Generated with source code generators				✔	
3 Converted with automated translators				✔	
4 Copied or reused without change				✔	

5 Modified				✔	
6 Removed					✔
7					
8					
Origin	**Definition** ✔	**Data array**		**Includes**	**Excludes**
1 New work: no prior existence				✔	
2 Prior work: taken or adapted from					
3 A previous version, build, or release				✔	
4 Commercial, off-the-shelf software (COTS), other than libraries				✔	
5 Government furnished software (GFS), other than reuse libraries				✔	
6 Another product				✔	
7 A vendor-supplied language support library (unmodified)					✔
8 A vendor-supplied operating system or utility (unmodified)					✔
9 A local or modified language support library or operating system				✔	
10 Other commercial library				✔	
11 A reuse library (software designed for reuse)				✔	
12 Other software component or library				✔	
13					
14					
Usage	**Definition** ✔	**Data array**		**Includes**	**Excludes**
1 In or as part of the primary product				✔	
2 External to or in support of the primary product					✔
3					

4. Литература

1. **Boehm B.W.**, **Software Engineering Economics**, Prentice Hall, Englewood Cliffs, N.J., 1981.

2. **Lederer A.L.**, **J. Prasad**, **Software Management and cost estimating error**, The Journal of Systems and Software, 50 (2000), p.33-42.

3. **Kemerer C.**, **An empirical validation of software cost estimation models**, Communication of the ACM 30(5), 416-429.

4. **Jones C., Applied Software Measurement**, McGraw-Hill, New York, 1997.

5. **Park R.E., Software Size Measurement: A Framework for Counting Source Statements**, Technical ReportCMU/SEI-92-TR-020 ESC-TR-92-020, 1992, 1996 SEI, Pittsburgh.