РАЗРАБОТВАНЕ НА ПРОГРАМНИ ПРИЛОЖЕНИЯ И ПРОБЛЕМИ ПРИ ОЦЕНЯВАНЕТО ИМ

Петър Халачев
Химикотехнологичен и металургичен университет – София, България

DEVELOPMENT OF SOFTWARE APPLICATIONS AND PROBLEMS IN THEIR EVALUATION

Petar Halachev
University of Chemical Technology and Metallurgy – Sofia, Bulgaria

Abstract

The assessment of the costs for design and development of software products is one of the most important stages in the process of their creation. This report explores the methods for evaluation of the software application and the amount of work to assess the labor and time required for its creation. The advantages and disadvantages of the presented models have been outlined and guidelines to improve the accuracy of the evaluation are given.

Keywords: evaluation, cost of labor, software applications, models

ВЪВЕДЕНИЕ

Оценяването на разходите при проектиране и разработване на програмни продукти е един от най-важните етапи в процеса на създаването им. От една страна определянето на по-ниска прогнозна стойност на разходите за трудови и материални ресурси, води до увеличаване на времето за разработване на програмния продукт и влошаване на качеството му. Определянето на по-висока предварителна стойност води до оскъпяване на програмния продукт, намалява ефективността и прави нецелесъобразно разработването му. В стандарта ISO12207, оценката на разходите е посочена като една от най-важните дейности при създаване на програмни продукти.

1.МЕТОДИ ЗА ОЦЕНЯВАНЕ НА ОБЕМА НА ПРОГРАМНОТО ПРИЛОЖЕНИЕ

Оценката на стойността на програмните продукти се съставя с цел определяне на ресурсите, необходими за проектирането, разработването, внедряването и по-нататъшното поддържане на програмното приложение. Общата стойност на проекта включва стойността на разработване на програмния продукт, стойността на съпътстващите и други видове разходи. Най-голям дял в общата стойност на продукта има стойността на разходите за труд, което изисква максимално точното им оценяване. При оценяване на разходите за труд при разработване на програмни приложения възникват проблеми, свързани с:

- избор на метрики за оценяване на обема на програмния продукт;
- избор на модел и метод за оценяване на разходите на труд;
- определяне на критерии за точността на оценката [1].

Преодоляването на посочените проблеми зависи от етапа на процеса на разработване на програмното приложение и изисква прилагането на разнообразни методи за оценяване на стойността. Разработването на програмни продукти преминава през следните етапи [2]:

1. Определяне и анализ на функционалните и програмни изисквания;
2. Определяне на етапите на разработката и отговорностите;

3. Оценяване на размера на програмния код;
4. Планиране на трудоемкостта на програмата;
5. Определяне на сроковете за изпълнение на проекта;
6. Изчисляване на стойността на проекта;
7. Оценка на рисковете при изпълнението на проекта;
8. Утвърждаване на сроковете, оценките и бюджета на проекта;
9. Сравнение на оценките с реалните показатели и определяне на степента им на точност.

Оценяването на обема на работите, необходими за изпълнение на проекта има важно значение при определяне на трудоемкостта. В практиката са известни различни методи за оценяване. Според Kan размера на изходния код на софтуера, обикновено се изразява в „реда код“ (Lines Of Code - LOC) [3].

При метода за оценяване *Lines Of Code,* обемът на работа се определя по количеството изходни редове код (с изключение на празните редове, коментари и специфични оператори), които съдържа програмния продукт. От гледна точка на мерната единица за обем съществуват:

Физически редове код, които са общият брой редове код в програмата;

Логически редове код, които са броя изпълними инструкции, от които се състои програмата.

Предимство на използването на *Line of Code*, като единица за обема на програмния продукт е бързото и лесно приложение, а недостатък е, че размера на проекта може да бъде определен само след завършването му. При използване на готови софтуерни компоненти (шаблони) приложението на *Line of Code* не е удачно, тъй като кода се създава автоматично. Оценяването при метода *Line of Code* зависи от програмния език и при него не се отчита качеството на кода (т.е. големия обем на кода не е показател за качеството му). В проекти, в които се използват повече от 1 програмен език оценяването на обема на кода също е затруднено. Така например при създаване на примерна програма за извеждане на съобщение „Hello world“ при код, написан на програмен език Assembler обемът е 11 реда, а при програмен език C++ броят на редовете е 3.

Код на Assembler:

```
.model tiny
.code
org 100h

main  proc

      mov    ah,9
      mov    dx,offset hello_message
      int    21h

      retn

hello_message db 'Hello, world!$'

main  endp
end   main
```

Код на C++:

```
#include <iostream.h>

void main() {
    cout<<"Hello, World!"<<endl;
}
```

Според Barry Boehm's [4] времето за разработване на даден програмен продукт може да бъде определено по следната формула:

$$T = k * (SLOC)^{(1+x)} \qquad (1)$$

където:

T – човеко-години за разработване на програмата;

k – може да бъде изведен при повече от 1 KLOC;

x – дроб от вида на 0.1 или 0.25

Според закона на Brooks (Brooks's law) „добавянето на повече хора към проекта, може да забави изпълнението му“ [5]. Увеличаването на времето за разработване на проекта се обяснява с 2 основни фактора:

Времето за обучение на новия служител и за приспособяването му към колектива (т.нар. „рампа на времето“) е загуба на време не само за новите, но и за работещите вече по проекта разработчици;

С увеличаване на броя на членовете в екипа на проекта се увеличава и времето за комуникация между тях. Според J. Taylor [6] броят на комуникационните канали се увеличава в по-голяма степен, отколкото

броя на разработчиците в проекта по следната формула:

$$N = \frac{n(n-1)}{2} \quad (2)$$

където: N – брой комуникационни канали;
n – брой разработчици

Производителността на разработчиците може да бъде оценена чрез следната формула:

$$ПП = \frac{LOC}{T} \quad (3)$$

където:
ПП – производителност на програмиста;
LOC – брой редове код;
T – време на работа на програмиста.

Единицата за определяне на обема на кода при метода *Line of Code* не отразява функционалните и качествени характеристики на кода. Ако разработчикът се стреми да оптимизира кода на приложението, като намали броя на редовете код (с цел по-ефективното поддържане на кода по време на експлоатацията му), производителността му като показател би намаляла. Времето изразходвано за оптимизиране на кода би намалило производителността още по-вече.

Function Point е въведен като алтернатива на *Line of Code* и е разработен от A. Albrecht [7] за компанията IBM през 70-те години на миналия век. От 1986 г. разработката продължава като International Function Point User Group (IFPUG), която представява практическо ръководство по приложението на *Function Point Counting Practices Manual* (FPCPM), което по-късно е официално признато като стандарт за оценяване на размера на кода на програмния продукт [8]. При метода *Function Points* обемът на работа по създаване на програмния продукт се определя по броя на функционалните елементи. Функционални елементи на програмния продукт са:

- екрани за въвеждане на информация;
- брой изходни справки (отчети, документи, екранни форми);
- заявки - двойка „въпрос/отговор“;
- логически файл (съвкупност от записи данни, използвани в приложенията);
- интерфейс приложения (записи данни, предавани към друго приложение или получавани от него).

При метода *Function Point* програмата се подразделя на класове компоненти според техния формат и тип на логическите операции, като в основата на това делене е предположението, че областта на взаимодействие на програмата се подразделя на вътрешно (взаимодействие на приложенията) и външно (взаимодействие с други приложения).

Резултатът от метода *Function Point* може да бъде коригиран с помощта на фактор за корекция на стойността Value Adjustment Factor (VAF) [9]. Коефициентът на корекция на стойността (VAF) се основава на 14 общи характеристики на системата (General system characteristics - GSC's), които оценяват общата функционалност на разработваните приложения и показват производителността, възможностите за разпределена обработка и повторно използване на кода. Степените на влияние обхващат скала от 0 до 5. Общите характеристики на модела включват:

1.Обмен на данни. Брой комуникационни средства за обмен на данни.

2. Разпределена обработка на данните. Възможност на програмата за работа в разпределена среда или с разпределени данни.

3. Изпълнение. Време за реакция или пропускателна способност, изисквана от потребителя.

4. Хардуерни ресурси. Степен на ефективно използване на хардуерните ресурси.

5. Транзакции. Честота на изпълнение на транзакциите (ежедневно, ежемесечно

6. Въведена информация On-Line. Обем информация, която се въвежда On-Line от оператора.

7. Ефективност на потребителя. Възможност за ефективна работа чрез потребителския интерфейс.

8. On-Line актуализация. Брой вътрешни логически файлове, актуализирани чрез On-Line транзакция.

9. Процес на обработка на данните. Вграден в приложението голям математически или логически апарат.

10. Многократно използване. Възможност за работа на приложението като софтуерен продукт за определена фирма или е предназначен за софтуерния пазар.

11. Лекота на инсталиране. Затруднено ли е инсталирането и прехода към нова версия.

12. Лекота на поддръжка. Степен на автоматизиране на процедурите по стартиране, спиране и архивиране.

13. Скалируемост на приложението. Възможност за приспособяване на приложението към размера и дейността на организацията.

14. Преходност на приложението. Възможност за лесна промяна на архитектурата и свойствата на приложението.

След отчитане на 14-те основни характеристики на системата се изчислява фактора за регулиране на стойността по формулата [9]:

$$VAF = 0.65 + \left[\left(\sum_{i=1}^{14} C_i \right) / 100 \right] \quad (4)$$

Където Ci – степен на влияние на i-тата характеристика от 14 GSC.

Приложението на *Function Points* притежава следните предимства:

-Точно определяне на размера на кода, като важен показател за производителността от гледна точка на вход/изход;

-Може да се използва както от технически специалисти, така и от неспециалисти, от потребители и клиенти.

-Чрез *Function Points* може да определи дали даден инструмент, език за разработка или среда за програмиране е по-производителна от друга.

Методът *Function Points* е в сътветствие с ISO 14143 - Functional size measurement.

Съществуват обаче програмни продукти, при които оценяването на обема на кода прилагането на метода *Function Points* е неефективно [11], например: управление на процесите в реално време; математически изчисления; симулации; системни приложения; инженерни приложения, тъй като те са характеризират с алгоритми с повишена сложност. За решаването на този проблем организацията Software Productivity Research (SPR) разработва методика за анализ на характерните точки (feature points - FP) [11]. При изчисляване на FP се оценява броя на алгоритмите в програмата и се модифицира степента на значимост (weighting values).

При оценяване на обема на програмния код се използва и метода PERT, който включва експертна оценка на 3 възможни размера на кода: възможно най-голям; възможно най-малък и най-вероятен. Методът PERT се използва успешно и за оценяване на стойността на отделни компоненти на програмния продукт.

2. ПРОБЛЕМИ ПРИ ОЦЕНЯВАНЕ НА СТОЙНОСТТА НА ПРОГРАМНИ ПРИЛОЖЕНИЯ

Методите за оценка на стойността на програмното осигуряване в зависимост от вида на оценката се подразделят на неалгоритмични и алгоритмични.

При неалгоритмичните методи се прилагат определени принципи, схеми и модели за оценяване и те включват: Price-to-win, оценка по Паркинсон, експертна оценка, оценка по аналогия. Алгоритмични методи са SLIM, COCOMO и др. и основа за определяне на стойността при тях е математически апарат.

Методът *Price-to-win* [4] се основава на принципа „клиентът винаги има право“ и независимо от прогнозираните разходи за разработване на даден проект в организацията, оценката се коригира в зависимост от желанието на потребителя на базата на договаряне. Недостатъци на метода са: планиране на недостатъчно количество ресурси за изпълнението на проекта, неспазване на сроковете, отказ от поръчката.

Според първия закон на *Паркинсон* „Работата се разраства до запълване на наличното време.“ При разработване на програмни проекти законът на Паркинсон може да бъде перифразиран „за да се повиши производителността на разработчиците е необходимо да бъде намалено времето за работа“ [4]. Недостатък на метода е, че при намаляване на времето за разработване би се намалило качеството на програмния продукт.

При *метода експертна оценка* се привличат специалисти, които на базата на опита си дават своите оценки за обема на работа и на основата на консенсус се взема решение. Прилага се в проекти с нови технологии, нови процеси или решаващи иновационни задачи [12].

Оценката по аналогия [13] е разновидност на експертната оценка, при която се използват данни за вече завършени програмни продукти.

Моделът *SLIM (Software Lifecycle Management)* се основава на анализа на Putman за продължителността на жизнения цикъл на програмния продукт, в който зависимостта между броя редове код, производителността, времето за разработване на проекта се определя по следната формула:

$$\frac{B^{1/3}.S}{P} = E^{1/3}.T^{4/3} \qquad (5)$$

където:

S - брой редове код - Putnam използва ESLOC (Effective Source Lines of Code).

B - скалиращ фактор и е функция на размера на проекта;

P - продуктивността на процеса на разработка в организацията;

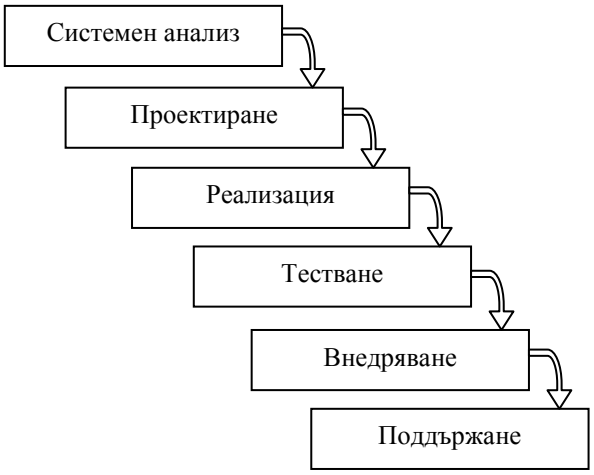
E - усилието вложено в проекта, и измерено в човекогодини;

T - времето за разработване на проекта (в години);

В изследване, проведено от PENGELLY се посочва, че SLIM не е приложим за оценяване на малки проекти. Според [14], [15], [16] SLIM е подходящ за оценяване на софтуерни проекти, които отговарят на следните критерии:

- S по-голям от 5000 реда;
- T по-дълъг от 1,5 години;

Моделът COCOMO (Constructive Cost Model) е статистически модел, разработен от Barry Boehm. Използвани са фактически данни за 63 програмни проекта, като за всеки проект са определени стойностите на 42 параметъра, характеризиращи програмния продукт и условията на работа. Чрез методите на статистическия анализ е получена аналитична зависимост на влиянието на 17 основни параметъра на показателите за разходи. Моделът COCOMO се основава на каскадния модел (Waterfall) за жизнен цикъл на продукта (Фигура 1), при който всички оценки на стойността и разходите за ресурси на етапите на програмния проект се отнасят към етапите на жизнения цикъл. Видовете и съдържанието на работите също се определят от каскадния модел, при който не се предвижда обратна връзка при изпълнението на етапите на разработване на приложението.



Фигура 1. *Каскаден модел на жизнения цикъл при разработване на програмно приложение*

Моделът COCOMO е йерархия от 3 модела в ред на нарастване на детайлността и точността им.

В COCOMO Model 1 (Basic) се изчисляват трудоемкостта и стойността на разработката като функция на размера на програмния продукт, който се изразява в оценъчни хиляди реда код (KLOC - *kilo lines of code*). При COCOMO Model 1, в зависимост от обема на работа и разработващия екип се разграничават 3 типа разработки:

- Organic mode – малки екипи разработчици с опит в работата със средно големи изисквания към разработката;
- Intermediate/Semi-detached mode – средни по големина екипи разработчици с различен опит в работата при разработки със средни изисквания;
- Intered/Embedded mode – разработки с високи изисквания към програмното, операционното и апаратното осигуряване.

Оценяването на трудоемкостта, срокът за изпълнение и необходимия брой разработчици в COCOMO Model 1 могат да бъдат изразени по следния начин [4]:

$$E = a_b(KLOC)^{Bb} \qquad (6)$$

$$T = c_b E^{Db} \qquad (7)$$

$$N = \frac{E}{T} \qquad (8)$$

където: E е трудоемкост на разработваното приложение в човекомесеци;

T - срок на изпълнението в месеци

N - брой разработчици.

Стойностите на коефициентите a_b, b_b, c_b и d_b са в зависимост от типа на проекта и са дадени в Таблица 1.

Таблица 1. Коефициенти на модела COCOMO 1 (Basic)

Тип на проекта	a_b	b_b	c_b	d_b
Organic mode	2.4	1.05	2.5	0.38
Intermediate/Semi-detached mode	3.0	1.12	2.5	0.35
Intered/Embedded mode	3.6	1.20	2.5	0.32

COCOMO Model 1 е подходящ за бърза оценка на стойността на разработката. При него обаче не се отчитат различията в програмните и апаратните ограничения, в качеството и опита на разработчиците, използването на съвременната техника и средства за разработка.

В COCOMO Model 2 (Intermediate) трудоемкостта на разработката се изчислява като функция от размера й и от 4 основни фактора, влияещи върху стойността, включващи субективна оценка на:

-Характеристики на продукта – изискана надежност, размер на базата данни и приложенията, сложност на продукта.

-Характеристики на проекта - използване на инструменти за разработване, прилагани методи за разработване на програмния продукт, спазване на графика за разработване.

-Характеристики на апаратното осигуряване – ограничение на бързодействието при изпълнение на програмата; ограничение на паметта; неустойчивост на обкръжението на виртуалната машина; изисквано време за възстановяване.

-Характеристики на персонала – аналитични способности; опит при разработки, при използване на виртуални машини и на езика за програмиране.

Общият брой на факторите е 15 с шестобален рейтинг, разграничени по степен на важност на фактора от „много нисък” до „много висок”

В COCOMO Model 3 (Advanced/Detailed) трудоемкостта на разработката се определя на всеки етап от процеса и включва всички характеристики от предишните модели.

Моделът COCOMO позволява да се оцени трудоемкостта, независимо от различните равнища на изисквания при създаване на програмни продукти. Лесен е за приложение. Основен недостатък е необходимостта от значителен обем информация за предишни разработки.

Повечето от представените модели се основават на обема на програмния код, което при съвременните технологии не напълно точно и адекватно отразява трудоемкостта при създаването на програмния продукт. Отсъстват показатели, необходими за отчитането на редица параметри, поради което направена на ранни етапи предполагаемата оценка може да е грешна.

ЗАКЛЮЧЕНИЕ

При оценяване на обема на програмния продукт и планиране на разходите за труд на разработчиците се наблюдават сложни взаимовръзки, както в първия етап на жизнения цикъл на разработването на проекта, така и на всеки следващ етап от създаването му. Процесът на оценката често е затруднен тъй като проекта трябва да удовлетворява взаимоизключващи се изисквания (конкурентна цена, високо качество, спазване на графика на работа, функционалност на приложението). Големият брой ограничения водят до усложняване на процеса на оценяване и намаляват точността на оценката. Тези проблеми биха могли да бъдат избегнати чрез спазването на някои изисквания като:

Оценяване на разходите на всеки етап от жизнения цикъл при разработване на програмното приложение;

Точно определяне на размера на работата, както от гледна точка на броя на редовете код, така и на функционалните точки чрез подходящи методи;

Използване на стандарта UML като унифициран език за описание на изискванията и структурата на програмното приложение;

Набиране и анализиране на максимално голям обем информация от разработване на предшестващи проекти;

Спазване на изискванията, както на разработчиците, така и на потребителите и др.

За точно и адекватно оценяване на трудоемкостта на процеса на проектиране и разработване на програмни продукти е необходимо разработване на такава методика, която да отчита комплексното влияние на всички взаимозависими фактори, ускоряващи или забавящи процеса на разработване.

ЛИТЕРАТУРА

[1]Rois W., Upravlenii preoktami po sozdamiiu programnova obespechenia, Moskva, LORI, 2002.

[2] Leung H., Zhang F., Software Cost Estimation :fttp://cs.pitt.edu/chang/handbook/42b.pdf, Retrieved 10.09.18.

[3] Kan St., Metrics and Models in Software Quality Engineering, 2nd Edition, Addison-Wesley Professional, 2003.

[4] Boehm B., Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[5] Frederick P., Brooks Jr., The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition 2nd Edition, Addison-Wesley, 1995.

[6] Taylor J., A Survival Guide for Project Managers, 2nd edition, AMACOM, 2006.

[7] Albrecht A., J. Gaffney, Software function, source lines of codes, and development effort prediction: a software science validation, IEEE Transactions on Software Engineering, 1983.

[8]ISO/IEC 20926:2003, Software engineering - IFPUG 4.1 Unadjusted functional size measurement method, Counting practices manual

[9] Longstreet D., Function Point Analysis Training Course, Longstreet Consulting Inc., 2004

[10] Rethinking and Restarting Software Development, Free Online book Introduction To Function Point Analysis, http://www.softwaremetrics.com/fpafund.htm, Retrieved 10.09.18.

[11] Fenton N., Pfleeger S., Software Metrics: A Rigorous and Practical Approach 2nd edition, PWS, 1998.

[12] Coates J., Technological Forecasting and Social Change, Elsevier Science Inc., 1999

[13] Shepperd M., C. Schofield, Estimating software project effort using analogy, IEEE Trans Software Engineering, 1997.

[14] Londeix B., Cost Estimation for Software Development. Addison-Wesley: Workingham, 1987.

[15] Londeix B., Aspects of estimation practice in software development, in Proc. Software Engineering 88, ed. Pyle, IC, Liverpool: IEE/BCS, 1988.

[16] http://www.ecfc.u-net.com/cost/slim.htm, Retrieved 10.09.18