МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДВНЗ “ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАСИЛЯ СТЕФАНИКА”

Кафедра інформаційних технологій

Хрущ Л. З.

Практикум з економіки програмного забезпечення

Навчально-методичний посібник
з курсу
“Економіка програмного забезпечення”

м. Івано-Франківськ
Прикарпатський національний університет імені Василя Стефаника
2018

УДК 004.41

Рекомендовано до друку вченою радою факультету математики та інформатики ДВНЗ "Прикарпатський національний університет імені Василя Стефаника (протокол №_ від 08 червня 2018 р.)

Рецензенти:

Мельничук С.І. – доктор технічних наук, професор, завідувач кафедри інформаційних технологій факультету математики та інформатики ДВНЗ "Прикарпатський національний університет імені Василя Стефаника";

Никифорчин І.В. – кандидат економічних наук, доцент, доцент кафедри статистики і вищої математики факультету математики та інформатики ДВНЗ "Прикарпатський національний університет імені Василя Стефаника";

Бандура А.І. – кандидат фізико-математичних наук, доцент, доцент кафедри вищої математики Івано-Франківського національного технічного університет нафти і газу".

Хрущ Л.З.

Практикум з економіки програмного забезпечення : навчально-методичний посібник / Л. З. Хрущ. – Івано-Франківськ : Видавництво Прикарпатського національного університету, 2018. – 103 с.

Навчально-методичний посібник містить виклад основного матеріалу курсу "Економіка програмного забезпечення". Тут викладені вказівки щодо проведення практичних робіт з даної дисципліни, завдання по варіантах для індивідуального виконання. У кожному розділі наводиться теоретичний матеріал з теми для опрацювання й контрольні запитання для закріплення вивченого матеріалу. Вкінці пропонуються запитання із можливими кількома варіантами відповідей, а також запитання на встановлення відповідності для узагальнення матеріалу.

"Практикум з економіки програмного забезпечення" розроблено для студентів спеціальності "Інженерія програмного забезпечення".

УДК 004.41

ЗМІСТ

ВСТУП

Оцінка трудомісткості створення програмного забезпечення (ПЗ) є важливою складовою у визначенні термінів реалізації програмного проекту і його здійсненності.

Моделі і методи оцінки трудомісткості використовуються для розробки бюджету проекту, аналізу ступеня ризику і вибір компромісного рішення, планування і управління проектом.

Без адекватної і достовірної оцінки трудовитрат неможливо забезпечити чітке планування і управління проектом.

Дисципліна “Економіка програмного забезпечення” має спрямованість на практичне оволодіння основними прийомами, методами та навичками створення оцінки програмних систем та проектів та їх подальшого використання в різних сферах життя.

У результаті вивчення дисципліни студенти повинні знати основні показники ресурсного потенціалу підприємства та ефективність його використання; теоретичні і методологічні основи економіки програмного забезпечення, метрики складності програм, методи оцінки вартості ПЗ.

Студенти повинні вміти розраховувати трудомісткість, застосовувати різні методи, розраховувати вартість програмного продукту, використовувати засоби оцінки вартості програмного продукту.

Навчально-методичний посібник “Практикум з економіки програмного забезпечення” призначений для формування у студентів практичних навичок в області розрахунку трудовитрат на розробку програмних продуктів на етапі планування робіт.

Практикум з економіки програмного забезпечення складається з вступу, розділів з описанням теоретичного матеріалу і практичних робіт, додатків, списку використаних джерел та інтернет-ресурсів.

Дане видання містять виклад основного матеріалу курсу “Економіка програмного забезпечення”. Тут викладені завдання до виконання практичних робіт з даної дисципліни, варіанти для індивідуального виконання. У кожному розділі наводиться теоретичний матеріал з теми для опрацювання й контрольні запитання для закріплення вивченого матеріалу. Для самоперевірки та закріплення знань студентів вкінці навчально-методичного посібника запропоновані запитання, у яких можливий один або кілька варіантів відповідей, а також пропонуються запитання на встановлення відповідності.

Навчально-методичний посібник спряє основним завданнями вивчення дисципліни “Економіка програмного забезпечення”, а саме отримання

студентом компетенцій для того, щоб розпізнавати різні методології розробки і оцінки вартості програмного продукту та використовувати їх на практиці.

Навчально-методичний посібник орієнтований на студентів комп'ютерних спеціальностей та має професійно-орієнтовану спрямованість. "Практикум з економіки програмного забезпечення" розроблено для студентів спеціальності "Інженерія програмного забезпечення" та близьких до даної спеціальності напрямів підготовки.

РОЗДІЛ 1
ОЦІНКА ВАРТОСТІ РОЗРОБКИ ПРОГРАМНОЇ СИСТЕМИ ЗА МОДЕЛЛЮ СОСОМО

1.1.Теоретичний матеріал з теми розділу

COCOMO (Constructive Cost Model) – це конструктивна модель вартості, розроблена на початку 80-х років Баррі Боемом [1] для оцінки трудомісткості розробки програмних продуктів.

Модель складається з ієрархії трьох послідовно деталізованих та уточнюючих рівнів (режимів). На кожному рівні всі проекти розбиваються на три групи за рівнем складності:

1) розповсюджений або органічний тип (organic projects);

2) напівнезалежний або напіврозподілений тип (semidetached projects);

3) вбудований тип (embedded projects).

В моделі COCOMO використовуються три режими (рівні), за допомогою яких класифікується складність системи, а також середовище розробки.

Розповсюджений або органічний (organic) режим. Розповсюджений режим звичайно класифікується як платіжна відомість, опис або наукове обчислення. Інші характеристики режиму: невелика команда по розробці проекту, необхідні невеликі нововведення, є несуворі обмеження і кінцевий термін, а середовище розробки є стабільним.

Напівнезалежний або напіврозподілений (semidetach) режим. Напівнезалежний режим типізується прикладними системами, наприклад, компіляторами, системами баз даних або редакторами. Інші характеристики: невелика команда по розробці проекту середнього розміру, необхідні деякі інновації, помірні обмеження і кінцевий термін, а середовище розробки дещо нестабільне.

Вбудований (embedded) режим. Вбудований режим характеризується режимами реального часу, наприклад, системами контролю повітряного руху, мережами АТМ або воєнними системами. Інші характеристики: велика команда розробників проекту, великий об’єм необхідних інновацій, жорсткі обмеження і терміни здачі. Середовище розробки в цьому випадку складається з багатьох складних інтерфейсів, включаючи ті з них, які поставляються замовникам разом з апаратним забезпеченням.

Тип тієї чи іншої групи можна розглядати як один з параметрів моделі COCOMO.

Модель COCOMO поділяється на рівні: базовий (basic), проміжний

(intermediate), деталізований (advanced).

Модель базового рівня моделі (Basic COCOMO) – двохпараметрична. Як параметри виступають тип проекту і обсяг програми (число рядків коду).

Рівняння цієї моделі мають вигляд:

$$PM = a_i \times (SIZE)^{b_i},$$
$$TM = c_i \times (PM)^{d_i},$$
$$SS = PM / TM,$$
$$P = SIZE / PM,$$

де PM (People × Month) – трудомісткість (люд. × міс.);

TM (Time at Month) – час розробки в календарних місяцях; $SIZE$ – обсяг програмного продукту в тисячах рядків вихідного тексту (Kilo of Source Line of Code – KSLOC);

SS – середня чисельність персоналу;

P – продуктивність.

Коефіцієнти a_i, b_i, c_i, d_i вибираються з таблиці (див. табл. 1.1).

Таблиця 1.1

Значення коефіцієнтів базової рівня моделі COCOMO залежно від типу проекту

Тип проекту	a_i	b_i	c_i	d_i
Розповсюджений (Органічний)	2,4	1,05	2,5	0,38
Напівнезалежний (Напівроздільний)	3,0	1,12	2,5	0,35
Вбудований	3,6	1,20	2,5	0,32

Модель цього рівня підходить для ранньої швидкої приблизної оцінки витрат, але точність її дуже низька.

Модель проміжного рівня моделі (Intermediate COCOMO) уточнена за рахунок введення додаткових 15 “атрибутів вартості” (або факторів витрат) Cost Drivers (CD_k), які згруповані за чотирма категоріями:

1) Характеристики продукту (Product Attributes):
- RELY – Необхідна надійність ПЗ (Required Software Reliability);
- DATA – Розмір БД додатку (Size of Application Database);
- CPLX – Складність продукту (Complexity of the Product);

2) Характеристики апаратного забезпечення (Hardware Attributes):
- TIME – Обмеження швидкодії при виконанні програми (Run-Time

Performance Constraints);

– STOR – Обмеження пам’яті (Memory Constraints);

– VIRT (PVOL) – Нестійкість оточення віртуальної машини (Volatility of the Virtual Machine Environment);

– TURN (STIME) – Необхідний час відновлення (Required Turnabout Time);

3) Характеристики персоналу (Personnel Attributes):

– ACAP (ASAP) – Аналітичні здібності (Analyst Capability);

– AEXP – Досвід розробки (Applications Experience);

– PCAP (PERS) – Здібності до розробки ПЗ (Software Engineer Capability);

– VEXP (PEXP) – Досвід використання віртуальних машин (Virtual Machine Experience);

– LEXP (LTEX) – Досвід розробки на мовах програмування (Programming Language Experience);

4) Характеристики проекту (Project Attributes):

– MODP (FCIL) – Застосування методів розробки ПЗ (Application of Software Engineering Methods);

– TOOL – Використання інструментарію розробки ПО (Use of Software Tools);

– SCED – Вимоги дотримання графіка розробки (Required Development Schedule).

Значення кожного атрибута вибирається з таблиці (див. табл. 1.2) відповідно до його ступеня значущості (рейтингу) в конкретному проекті.

Таблиця 1.2

Значення атрибутів вартості залежно від їх рівня

Атрибути вартості, CD_k	**Рейтинг**					
	Дуже низь-кий	**Низь-кий**	**Серед-ній**	**Висо-кий**	**Дуже висо-кий**	**Критич-ний**
Характеристики продукту						
1. Необхідна надійність ПЗ	0,75	0,88	1,00	1,15	1,40	n/a
2. Розмір БД додатка	n/a	0,94	1,00	1,08	1,16	n/a
3. Складність продукту	0,70	0,85	1,00	1,15	1,30	1,65

Характеристики апаратного забезпечення						
4. Обмеження швидкодії при виконанні програми	n/a	n/a	1,00	1,11	1,30	1,66
5. Обмеження пам'яті	n/a	n/a	1,00	1,06	1,21	1,56
6. Нестійкість оточення віртуальної машини	n/a	0,87	1,00	1,15	1,30	n/a
7. Необхідний час відновлення	n/a	0,87	1,00	1,07	1,15	n/a
Характеристики персоналу						
8. Аналітичні здібності	1,46	1,19	1,00	0,86	0,71	n/a
9. Досвід розробки	1,29	1,13	1,00	0,91	0,82	n/a
10. Здібності до розробки ПЗ	1,42	1,17	1,00	0,86	0,70	n/a
11. Досвід використання віртуальних машин	1,21	1,10	1,00	0,90	n/a	n/a
12. Досвід розробки на мовах програмування	1,14	1,07	1,00	0,95	n/a	n/a
Характеристики проекту						
13. Застосування методів розробки ПЗ	1,24	1,10	1,00	0,91	0,82	n/a
14. Використання інструментарію розробки	1,24	1,10	1,00	0,91	0,83	n/a
15. Вимоги дотримання графіка розробки	1,23	1,08	1,00	1,04	1,10	n/a

Примітка: n/a (not available) - дані відсутні, тобто відповідний рівень не оцінюється

Формула проміжного рівня моделі має вигляд

$$PM = EAF \times a_i \times (SIZE)^{b_i},$$

де PM – трудомісткість (люд. × міс.);

$SIZE$ – обсяг програмного продукту в тисячах рядків вихідного тексту (Kilo of Source Line of Code – KSLOC).

EAF (Effort Adjustment Factor) – добуток обраних атрибутів вартості з таблиці (див. табл. 1.2):

$$EAF = \prod_{k=1}^{15} CD_k$$

Коефіцієнти моделі a_i, b_i вибираються з таблиці (див. табл. 1.3).

Таблиця 1.3

Значення коефіцієнтів проміжного рівня моделі COCOMO залежно від типу проекту

Тип проекту, *i*	a_i	b_i
1. Розповсюджений	3,2	1,05
2. Напівнезалежний	3,0	1,12
3. Вбудований	2,8	1,20

Час розробки розраховується за тією ж формулою, що і для базової моделі.

1.2. Практична робота № 1

Тема роботи: Оцінка вартості розробки програмної системи (ПС) за моделлю COCOMO.

Мета роботи: Розрахунок вартості ПС за моделлю COCOMO, в залежності від рівней моделі, складності системи.

Завдання.

1. Розрахувати за базовим рівнем моделі COCOMO трудовитрати (PM) і визначити час розробки (TM). Визначити середню чисельність (SS) і рівень продуктивності (P), якщо:

Варіант 1. розмір проекту, який розробляється, оцінюється в 10 KLOC.
Варіант 2. розмір проекту, який розробляється, оцінюється в 300 KLOC.
Варіант 3. розмір проекту, який розробляється, оцінюється в 50 KLOC.
Варіант 4. розмір проекту, який розробляється, оцінюється в 55 KLOC.
Варіант 5. розмір проекту, який розробляється, оцінюється в 320 KLOC.
Варіант 6. розмір проекту, який розробляється, оцінюється в 25 KLOC.
Варіант 7. розмір проекту, який розробляється, оцінюється в 72 KLOC.
Варіант 8. розмір проекту, який розробляється, оцінюється в 85 KLOC.
Варіант 9. розмір проекту, який розробляється, оцінюється в 400 KLOC.

Варіант 10. розмір проекту, який розробляється, оцінюється в 7,5 KLOC.

2. Визначити режим складності системи за проміжним рівнем моделі COCOMO, якщо розмір проекту за першим завданням, а інші показники беруться відповідно до варіанту:

Варіант 1.

а) значення множників (драйверів) витрат ACAP, PCAP, TIME, DATA, LEXP змінюються до високих, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY, DATA, CPLX – низькі; TIME, STOR, VIRT (PVOL), TURN (STIME) – високі; ACAP (ASAP), AEXP, PCAP (PERS), VEXP (PEXP), LEXP (LTEX) – високі; MODP (FCIL), TOOL, SCED – високі.

Варіант 2.

а) значення множників (драйверів) витрат RELY, DATA, PVOL, PCAP, змінюються до низьких, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY, DATA, CPLX – високі; TIME, STOR, VIRT (PVOL), TURN (STIME) – дуже високі; ACAP (ASAP), AEXP, PCAP (PERS), VEXP (PEXP), LEXP (LTEX) – дуже високі; MODP (FCIL), TOOL, SCED –низькі.

Варіант 3.

а) значення множників (драйверів) витрат ACAP, CPLX змінюються до високих TIME, DATA, LEXP змінюються до низьких, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY, DATA, CPLX – високі; TIME, STOR, VIRT (PVOL), TURN (STIME) – високі; ACAP (ASAP), AEXP, PCAP (PERS), VEXP (PEXP), LEXP (LTEX) – дуже високі; MODP (FCIL), TOOL, SCED – дуже високі.

Варіант 4.

а) значення множників (драйверів) витрат TIME, LEXP, CPLX, змінюються до дуже високих, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY – низький, DATA – дуже високий, CPLX – високий; TIME – високий, STOR – дуже високий, VIRT (PVOL) – низький, TURN (STIME) – високий; ACAP (ASAP) – високий, AEXP – дуже високий, PCAP (PERS) – високий, VEXP (PEXP) – високий, LEXP

(LTEX) – дуже високий; MODP (FCIL), TOOL, SCED – високі.

Варіант 5.

а) значення множників (драйверів) витрат TOOL, SCED змінюються до низьких, LEXP, STOR змінюються до дуже високих, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY – високий, DATA – дуже високий, CPLX – високий; TIME – дуже високий, STOR –високий, VIRT (PVOL), – високий, TURN (STIME) – дуже високий; ACAP (ASAP) – дуже високий, AEXP – низький, PCAP (PERS) – низький, VEXP (PEXP) – високий, LEXP (LTEX) – високий; MODP (FCIL), TOOL, SCED – низький.

Варіант 6.

а) значення множників (драйверів) витрат CPLX, STOR, DOCU, PCAP змінюються до дуже високих, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY – низький, DATA – дуже низький, CPLX – низький; TIME – дуже низький, STOR – високий, VIRT (PVOL) – дуже високий, TURN (STIME) – високий; ACAP (ASAP) – високий, AEXP – дуже низький, PCAP (PERS) – дуже низький, VEXP (PEXP) – високий, LEXP (LTEX) –високий; MODP (FCIL), TOOL, SCED – високий.

Варіант 7.

а) значення множників (драйверів) витрат ACAP, APEX, PCAP, LEXP змінюються до низьких, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY – високий, DATA – дуже високий, CPLX – низький; TIME – дуже низький, STOR – високий, VIRT (PVOL) – дуже високий, TURN (STIME) – дуже високий; ACAP (ASAP) – високий, AEXP – низький, PCAP (PERS) – низький, VEXP (PEXP) – високий, LEXP (LTEX) –високий; MODP (FCIL) – високий, TOOL – дуже високий, SCED – високий.

Варіант 8.

а) значення множників (драйверів) витрат CPLX, SCED змінюються до дуже низьких, ACAP змінюються до низьких, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY – низький, DATA – дуже високий, CPLX – низький; TIME – дуже високий, STOR – високий, VIRT (PVOL) – дуже високий, TURN (STIME) – дуже високий; ACAP (ASAP) – високий, AEXP – високий, PCAP (PERS) – дуже низький, VEXP (PEXP) –

високий, LEXP (LTEX) –високий; MODP (FCIL) – високий, TOOL – дуже високий, SCED – високий.

Варіант 9.

а) значення множників (драйверів) витрат RELY, DATA, ACAP, PCAP, STOR змінюються до низьких, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY – дуже низький, DATA – дуже низький, CPLX – низький; TIME – дуже високий, STOR – низький, VIRT (PVOL) – дуже низький, TURN (STIME) – дуже високий; ACAP (ASAP) – високий, AEXP – високий, PCAP (PERS) – дуже низький, VEXP (PEXP) – високий, LEXP (LTEX) – низький; MODP (FCIL) – низький, TOOL – дуже низький, SCED – високий.

Варіант 10.

а) значення множників (драйверів) витрат SITE, TOOL змінюються до дуже низькі, SCED змінюються до низьких, всі інші значення номінальні.

б) значення множників (драйверів) витрат: RELY – низький, DATA – високий, CPLX – низький; TIME –високий, STOR – низький, VIRT (PVOL) – дуже низький, TURN (STIME) – високий; ACAP (ASAP) – високий, AEXP – високий, PCAP (PERS) – дуже низький, VEXP (PEXP) – високий, LEXP (LTEX) – низький; MODP (FCIL) – низький, TOOL –низький, SCED – високий.

3. Написати COCOMO калькулятор для розрахунку трудоємності, терміну розробки програмного продукту на основі базового рівня вартості COCOMO із зручним користувацьким інтерфейсом (див. рис.1.1).

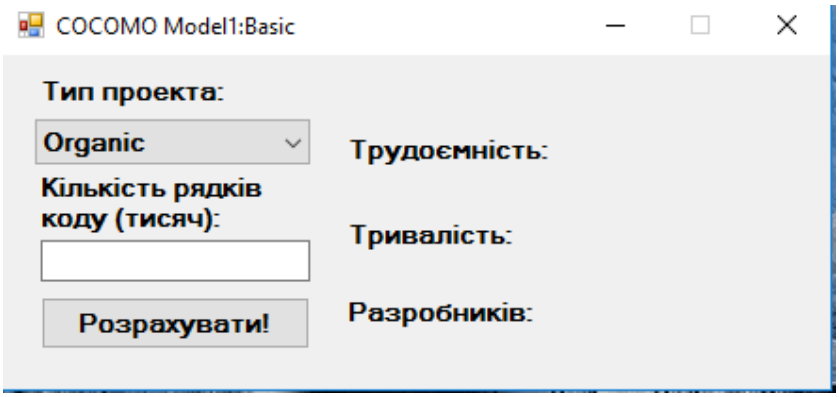


Рис. 1.1. Приклад інтерфейсу COCOMO-калькулятора

Приклад програмної реалізації COCOMO-калькулятора:

```
      using System;
using System.Collections.Generic; using System.Linq;
                                    using System.Text; using System.Data;

namespace COCOMO_Model1_Basic { public static class BasicModel {
      //Таблиця коефіцієнтів
      static double[][] modelTable = new double[3][]; static BasicModel()
      {
         modelTable[0] = new[] { 2.4, 1.05, 2.5, 0.38 }; modelTable[1] =
         new[] { 3.0, 1.12, 2.5, 0.35 }; modelTable[2] = new[] { 3.6,
         1.20, 2.5, 0.32 };
      } //Трудоємність
      public static double GetEfforts(int codeSize, ProjectType type) {
         return modelTable[(int)type][0] * (Math.Pow(codeSize,
modelTable[(int)type][1]));
      }
      //Термін розробки
      public static double GetTimeToDevelop(int codeSize, ProjectType
type) {
         return modelTable[(int)type][2] * (Math.Pow(GetEfforts(codeSize,
type), modelTable[(int)type][3]));
      }
      //Число необхідних розробників
      public static double GetPersonsToDevelop(int codeSize, ProjectType
type) {
         return GetEfforts(codeSize, type) / GetTimeToDevelop(codeSize,
type);
      } }
   //Типи проекту
   public enum ProjectType { Organic = 0, Semi_detached = 1, Embedded = 2
   }
     }


using System;
using System.Collections.Generic; using System.ComponentModel;  using
System.Data;
using System.Drawing; using System.Linq; using System.Text;
using System.Windows.Forms;

namespace COCOMO_Model1_Basic {
   public partial class MainForm : Form { bool isDigit = true;
      public MainForm() { InitializeComponent();
         this.Load += new EventHandler(MainForm_Load);
         txtCodeSize.KeyDown += new
KeyEventHandler(txtCodeSize_KeyDown); txtCodeSize.KeyPress += new
KeyPressEventHandler(txtCodeSize_KeyPress);
         btnGetData.Click += new EventHandler(btnGetData_Click); }

      void btnGetData_Click(object sender, EventArgs e) {  if
```

```
            (string.IsNullOrWhiteSpace(txtCodeSize.Text)) {
               return; }
            ProjectType type = (ProjectType)Enum.Parse(typeof(ProjectType),
cmbProjectType.Text);
            int codeSize = int.Parse(txtCodeSize.Text);
            lblEfforts.Text = string.Format("{0} {1}",
               BasicModel.GetEfforts(codeSize,
type).ToString("F2"),"чел/мес.");
                                    lblTimeTo.Text = string.Format("{0} {1}",
                                        BasicModel.GetTimeToDevelop(codeSize,
type).ToString("F2"), "мес.");
            lblDevelopers.Text = string.Format("{0} {1}",
               BasicModel.GetPersonsToDevelop(codeSize,
type).ToString("F1"), "чел."); }

         void txtCodeSize_KeyPress(object sender, KeyPressEventArgs e) {
            e.Handled = isDigit; }

         void txtCodeSize_KeyDown(object sender, KeyEventArgs e) { isDigit =
            ((e.KeyCode < Keys.D0 || e.KeyCode > Keys.D9)
&& (e.KeyCode!= Keys.Back)); }

         void MainForm_Load(object sender, EventArgs e) { string[]
            projectTypes =
Enum.GetNames(typeof(ProjectType));
            cmbProjectType.Items.AddRange(projectTypes); cmbProjectType.Text
            =
cmbProjectType.Items[0].ToString(); lblEfforts.Text = ""; lblTimeTo.Text
            = ""; lblDevelopers.Text = "";
         }

      }
        }
```

1.3. Контрольні запитання

1. Які рівні (режими) моделі COCOMO ви знаєте?
2. Чим характеризується organic projects моделі COCOMO?
3. Чим характеризується semidetached projects моделі COCOMO?
4. Чим характеризується embedded projects моделі COCOMO?
5. Який вигляд мають рівняння базового рівня моделі COCOMO?
6. Що таке KSLOC?
7. Які характеристики продукту проміжного рівня моделі COCOMO ви знаєте?
8. Які характеристики апаратного забезпечення проміжного рівня моделі COCOMO ви знаєте?
9. Які характеристики персоналу проміжного рівня моделі COCOMO ви знаєте?

10. Які характеристики проекту проміжного рівня моделі COCOMO ви знаєте?
11. Який вигляд мають рівняння проміжного рівня моделі COCOMO?
12. Що таке Effort Adjustment Factor?

РОЗДІЛ 2
ОЦІНКА ВАРТОСТІ РОЗРОБКИ ПРОГРАМНОЇ СИСТЕМИ ЗА МОДЕЛЛЮ COCOMO II

2.1. Теоретичний матеріал з теми розділу

У 1997 методика була вдосконалена і отримала назву COCOMO II.

Розрізняють дві стадії оцінки проекту:

1) попередня оцінка на початковій фазі (Early Design)
2) детальна оцінка після опрацювання архітектури (Post Architecture).

Для розрахунку трудомісткості необхідно спочатку оцінити фактори (чинники) маштабу (Scale Drivers) та множники трудомісткості (Cost Drivers або Effort Multipliers). Фактори масштабу застосовуються на двох стадіях оцінки проекту. Множники трудомісткості відрізняються для різних стадій оцінки проекту. На різних стадіях відрізняється їх кількість і значення. Для стадії Early Design необхідно оцінити сім множників трудомісткості, а для стадії Post Architecture – сімнадцять.

Формула оцінки трудомісткості проекту в люд. × міс. має вигляд

$$PM = EAF \times A \times (SIZE)^{E}$$

де $E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$ **;**

$B = 0{,}91;\ A = 2{,}94$ для попередньої оцінки;

$A = 2{,}45$ для детальної оцінки;

SF_j – фактори (чинники) масштабу (Scale Factors) (див. табл. 2.1-2.2);

$SIZE$ – обсяг програмного продукту в тисячах рядків вихідного тексту (KSLOC – Kilo of Source Line of Code);

EM_j – множники трудомісткості (Effort Multipliers). n=7 – для попередньої оцінки (табл. 2.3), n=17 – для детальної оцінки (табл. 2.4);

EAF (Effort Adjustment Factor) – добуток обраних множників трудоємкості:

$$EAF = \prod_{k=1}^{n} EM_k .$$

В методиці використовуються п’ять факторів масштабу SF, які

визначаються наступними характеристиками проекту:

PREC – прецедентність, наявність досвіду аналогічних розробок,

FLEX – гнучкість процесу розробки,

RESL – архітектура і дозвіл ризиків,

TEAM – спрацьованість команди,

PMAT – зрілість процесів.

Всі фактори масштабу мають певну оцінку: Very Low – дуже низька оцінка фактора, Low – низька оцінка, Nominal – середня оцінка, High – висока оцінка, Very High – дуже висока оцінка, Extra High – критично висока оцінка.

Детальний опис факторів масштабу наведено в таблиці 2.1 (див. табл.2.1).

Таблиця 2.1

Опис рівнів значущості чинників масштабу

SFj	Опис	Рівень значущості чинника					
		Дуже низький	Низький	Середній	Високий	Дуже високий	Критичний
1. PREC. Precedentedness	Прецедентність, наявність досвіду аналогічних розробок	досвід у продукті і платформі відсутній	продукт і платформа не дуже знайомі	деякий досвід в продукті і платформі присутній	продукт і платформа в основному відомі	продукт і платформа великою мірою знайомі	продукт і платформа повністю знайомі
2. FLEX. Development Flexibility	Гнучкість процесу розробки	процес строго детермінований	допускаються деякі компроміси	значна жорсткість процесу	відносна жорсткість процесу	незначна жорсткість процесу	визначені тільки загальні цілі
3. RESL. Architecture / Risk Resolution	Архітектура і дозвіл ризиків	ризики відомі / / проаналізовані на 20%	ризики відомі / / проаналізовані на 40%	ризики відомі / / проаналізовані на 60%	ризики відомі / / проаналізовані на 75%	ризики відомі / / проаналізовані на 90%	ризики дозволені на 100%
4. TEAM. Team Cohesion	Спрацьованість команди	формальна взаємодія	важка взаємодія до деякої міри	частіше колективна робота	у основному колективна робота	висока міра взаємодії	повна довіра, взаємозаміна і взаємодопомога

5. PMAT. Process Maturity	Зрілість процесів	CMM Рівень 1 (нижче середнього)	CMM Рівень 1 (вище середнього)	CMM Рівень 2	CMM Рівень 3	CMM Рівень 4	CMM Level 5

Ці фактори застосовуються на обох стадіях оцінки проекту.

Числові значення фактора масштабу в залежності від оцінки його рівня, наведені в таблиці 2.2 (див. табл. 2.2).

Таблиця 2.2

Значення чинника масштабу залежно від оцінки його рівня

Чинник масштабу, *SFj*	**Оцінка рівня чинника (фактора)**					
	Very Low	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
1. PREC	6,20	4,96	3,72	2,48	1,24	0,00
2. FLEX	5,07	4,05	3,04	2,03	1,01	0,00
3. RESL	7,07	5,65	4,24	2,83	1,41	0,00
4. TEAM	5,48	4,38	3,29	2,19	1,10	0,00
5. PMAT	7,80	6,24	4,68	3,12	1,56	0,00

Кількість і значення множників трудомісткості EM відрізняються для різних стадій оцінки проекту.

1) Стадія попередньої оцінки трудомісткості програмного проекту (Early Design). Для цієї оцінки необхідно оцінити для проекту рівень семи множників трудомісткості EM_j:

– параметри персоналу:

1. PERS (Personnel Capability) – кваліфікація персоналу (Extra Low – аналітики і програмісти мають нижчу кваліфікацію, плинність більше 45%; Extra High – аналітики і програмісти мають вищу кваліфікацію, плинність менше 4%);

2. PREX (Personnel Experience) – досвід персоналу (Extra Low – новий додаток, інструменти і платформа; Extra High – додаток, інструменти і платформа добре відомі);

– параметри продукту:

3. RCPX (Product Reliability and Complexity) – складність і надійність продукту (Extra Low – продукт простий, спеціальних вимог по надійності немає, БД маленька, документація не потрібна; Extra High – продукт дуже складний, вимоги по надійності жорсткі, БД надвелика, документація потрібно

в повному обсязі);

4. RUSE (Developed for Reusability) – розробка для повторного використання (Low – не вимагається; Extra High – передбачається повторне використання в інших продуктах);

– параметри платформи:

5. PDIF (Platform Difficulty) – складність платформи розробки (Extra Low – спеціальні обмеження по пам'яті і швидкодії відсутні, платформа стабільна; Extra High – жорсткі обмеження по пам'яті і швидкодії, платформа нестабільна);

– параметри проекта:

6. FCIL (Facilities) – обладнання (Extra Low – інструменти найпростіші, комунікації ускладнені; Extra High – інтегровані засоби підтримки життєвого циклу, інтерактивні мультимедіа комунікації);

7. SCED (Required Development Schedule) – необхідний виконання графіка робіт (Very Low – 75% від номінальної тривалості; Very High – 160% від номінальної тривалості).

Значення множників трудомісткості в залежності від їх рівня наведені в табл. 2.3 (див. табл. 2.3).

Таблиця 2.3

Значення множників трудомісткості залежно від оцінки їх рівня (Early Design)

№	Множник трудомісткості, EMi	Оцінка рівня множника трудомісткості						
		Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
1	PERS	2,12	1,62	1,26	1,00	0,83	0,63	0,50
2	PREX	1,59	1,33	1,22	1,00	0,87	0,74	0,62
3	RCPX	0,49	0,60	0,83	1,00	1,33	1,91	2,72
4	RUSE	n/a	n/a	0,95	1,00	1,07	1,15	1,24
5	PDIF	n/a	n/a	0,87	1,00	1,29	1,81	2,61
6	FCIL	1,43	1,30	1,10	1,00	0,87	0,73	0,62
7	SCED	n/a	1,43	1,14	1,00	1,00	n/a	n/a

Примітка: n/a (not available) - дані відсутні, тобто відповідний рівень не оцінюється

2) Стадія детальної оцінки після опрацювання архітектури (Post Architecture). Для цієї оцінки необхідно оцінити для проекту рівень сімнадцяти множників трудомісткості EM_j:

– параметри персоналу:

1) Analyst Capability (ACAP) – можливості аналітика;

2) Applications Experience (AEXP) – досвід розробки додатків;

3) Programmer Capability (PCAP) – можливості програміста;

4) Personnel Continuity (PCON) – тривалість роботи персоналу;

5) Platform Experience (PEXP) – досвід роботи з платформою;

6) Language and Tool Experience (LTEX) – досвід використання мови програмування і інструментальних засобів.

– параметри продукту:

7) Required Software Reliability (RELY) – необхідна надійність програми;

8) Database Size (DATA) – розмір бази даних;

9) Software Product Complexity (CPLX) – складність програми;

10) Required Reusability (RUSE) – необхідна можливість багаторазового використання;

11) Documentation Match to Life-Cycle Needs (DOCU) – відповідність документації потребам життєвого циклу.

– параметри платформи:

12) Execution Time Constraint (TIME) – обмеження часу виконання;

13) Main Storage Constraint (STOR) – обмеження пам'яті;

14) Platform Volatility (PVOL) – змінність платформи.

– параметри проекту:

15) Use of Software Tools (TOOL) – використання інструментальних програмних засобів;

16) Multisite Development (SITE) – багатоабонентська (віддалена) розробка;

17) Required Development Schedule (SCED) – необхідний виконання графіка робіт.

Значення множників трудомісткості в залежності від їх рівня наведені в табл. 2.4 (див. табл. 2.4).

Таблиця 2.4

Значення множників трудомісткості залежно від оцінки їх рівня (Post Architecture)

№	Effort Multiplier, EMJ		Very Low	Low	Nominal	High	Very High	Extra High
	Personnel Factors							
1	ACAP	Analyst Capability	1,42	1,29	1,00	0,85	0,71	n/a
2	AEXP	Applications Experience	1,22	1,10	1,00	0,88	0,81	n/a
3	PCAP	Programmer Capability	1,34	1,15	1,00	0,88	0,76	n/a
4	PCON	Personnel Continuity	1,29	1,12	1,00	0,90	0,81	n/a
5	PEXP	Platform Experience	1,19	1,09	1,00	0,91	0,85	n/a

6	LTEX	Language and Tool Experience	1,20	1,09	1,00	0,91	0,84	n/a
	Product Factors							
7	RELY	Required Software Reliability	0,84	0,92	1,00	1,10	1,26	n/a
8	DATA	Database Size	n/a	0,23	1,00	1,14	1,28	n/a
9	CPLX	Software Product Complexity	0,73	0,87	1,00	1,17	1,34	1,74
10	RUSE	Required Reusability	n/a	0,95	1,00	1,07	1,15	1,24
11	DOCU	Documentation Match to Life -	0,81	0,91	1,00	1,11	1,23	n/a
		Cycle Needs						
	Platform Factors							
12	TIME	Execution Time Constraint	n/a	n/a	1,00	1,11	1,29	1,63
13	STOR	Main Storage Constraint	n/a	n/a	1,00	1,05	1,17	1,46
14	PVOL	Platform Volatility	n/a	0,87	1,00	1,15	1,30	n/a
	Project Factors							
15	TOOL	Use of Software Tools	1,17	1,09	1,00	0,90	0,78	n/a
17	SITE	Multisite Development	1,22	1,09	1,00	0,93	0,86	0,80
16	SCED	Required Development Schedule	1,43	1,14	1,00	1,00	1,00	n/a

Примітка: n/a (not available) - дані відсутні, тобто відповідний рівень не оцінюється

Тривалість проекту або час розробки проекту TM в методиці COCOMO II для обох рівнів розраховується за формулою:

$$TM = SCED \times C \times (PM_{NS})^{B+0,2\times(E-B)}$$

де $C = 3{,}67; D = 0.28;$ PM_{NS} – розрахована трудомісткість проекту без урахування множника SCED, що визначає стиснення розкладу.

2.2. Практична робота № 2. Оцінка вартості розробки програмної системи за моделлю COCOMO II

Мета роботи: Розрахунок вартості ПС за моделлю COCOMO II за попередньою та детальною оцінками.

Завдання

1. Оцінити трудовитрати за моделлю COCOMO II (для попередньої оцінки). Значення $SIZE$ згідно свого варіанту. Показник SF – середній рівень, EM – високий рівень.

Варіант 1. розмір проекту, який розробляється, оцінюється в 10 KLOC.
Варіант 2. розмір проекту, який розробляється, оцінюється в 300 KLOC.
Варіант 3. розмір проекту, який розробляється, оцінюється в 50 KLOC.
Варіант 4. розмір проекту, який розробляється, оцінюється в 55 KLOC.
Варіант 5. розмір проекту, який розробляється, оцінюється в 320 KLOC.
Варіант 6. розмір проекту, який розробляється, оцінюється в 25 KLOC.
Варіант 7. розмір проекту, який розробляється, оцінюється в 72 KLOC.
Варіант 8. розмір проекту, який розробляється, оцінюється в 85 KLOC.
Варіант 9. розмір проекту, який розробляється, оцінюється в 400 KLOC.
Варіант 10. розмір проекту, який розробляється, оцінюється в 7,5 KLOC.

2. Оцінити трудовитрати за моделлю COCOMO II (для попередньої оцінки).

Варіант 1. Розмір проекту, який розробляється, оцінюється в 10 KLOC.
Досвід у продукті і платформі відсутній, присутня значна жорсткість процесу розробки, ризики проаналізовані на 75 %, у команді в основному колективна робота, зрілість процесів визначається за рівнем 4 моделі CMM.
Множники трудомісткості оцінюються з наступної інформації:
Досвід персоналу – низький рівень;
Кваліфікація персоналу – дуже низький рівень;
Складність і надійність продукту – середній рівень;
Складність платформи розробки – середній рівень;
Розробка для повторного використання – дуже низький рівень;
Необхідне виконання графіка робіт – низький рівень;
Обладнання – середній рівень.

Варіант 2. Розмір проекту, який розробляється, оцінюється в 300 KLOC.
Досвід у продукті і платформі присутній, присутня значна жорсткість процесу розробки, ризики проаналізовані на 40 %, у команді формальна взаємодія, зрілість процесів визначається за рівнем 1 (вище середнього) моделі CMM.
Множники трудомісткості оцінюються з наступної інформації:
Кваліфікація персоналу – низький рівень;
Складність і надійність продукту – високий рівень;
Розробка для повторного використання – високий рівень
Досвід персоналу – середній рівень;

Обладнання – середній рівень;

Складність платформи розробки – низький рівень;

Необхідне виконання графіка робіт – низький рівень.

Варіант 3. Розмір проекту, який розробляється, оцінюється в 50 KLOC.

Досвід у продукті і платформі в основному присутній, присутня відносна жорсткість процесу розробки, ризики проаналізовані на 90 %, у команді в частіше колективна робота,, зрілість процесів визначається за рівнем 2 моделі CMM.

Множники трудомісткості оцінюються з наступної інформації:

Складність і надійнсть продукту – дуже високий рівень;

Кваліфікація персоналу – високий рівень;

Досвід персоналу – дуже високий рівень;

Обладнання – низький рівень;

Розробка для повторного використання – середній рівень;

Необхідне виконання графіка робіт – високий рівень;

Складність платформи розробки – низький рівень.

Варіант 4. Розмір проекту, який розробляється, оцінюється в 55 KLOC.

Досвід у продукті і платформі не дуже присутній, процес розробки строго детермінований, ризики відомі і проаналізовані на 40 %, у команді до деякої міри важка взаємодія, зрілість процесів визначається за рівнем 2 моделі CMM.

Множники трудомісткості оцінюються з наступної інформації:

Кваліфікація персоналу – низький рівень;

Обладнання – дуже низький рівень;

Складність і надійність продукту – низький рівень;

Розробка для повторного використання – низький рівень;

Досвід персоналу – середній рівень;

Складність платформи розробки – дуже низький рівень;

Необхідне виконання графіка робіт – низький рівень.

Варіант 5. Розмір проекту, який розробляється, оцінюється в 320 KLOC.

Досвід у продукті і платформі повнісю присутній, у процесі розробки присутня незначна жорсткість, ризики дозволені на 90 %, у команді присутня повна довіра, взаємозаміна і взаємодопомога, зрілість процесів визначається за рівнем 4 моделі CMM.

Множники трудомісткості оцінюються з наступної інформації:

Досвід персоналу – дуже високий рівень;

Складність платформи розробки – дуже високий рівень;

Кваліфікація персоналу – критично високий рівень;

Складність і надійність продукту – високий рівень;

Розробка для повторного використання – критично високий рівень;

Необхідне виконання графіка робіт – високий рівень;

Обладнання – високий рівень.

Варіант 6. Розмір проекту, який розробляється, оцінюється в 25 KLOC.

Продукт і платформа в основному відомі, у процесі розробки присутня відносна жорсткість, ризики дозволені на 60 %, у команді присутня висока міра взаємодії, зрілість процесів визначається за рівнем 3 моделі CMM.

Множники трудомісткості оцінюються з наступної інформації:

Складність платформи розробки – дуже високий рівень;

Складність і надійність продукту – середній рівень;

Кваліфікація персоналу – низький рівень;

Досвід персоналу – високий рівень;

Розробка для повторного використання – високий рівень;

Обладнання – високий рівень;

Необхідне виконання графіка робіт – середній рівень.

Варіант 7. Розмір проекту, який розробляється, оцінюється в 72 KLOC.

Присутній деякий досвід у продукті і платформі, у процесі розробки незначна жорсткість, ризики дозволені на 100 %, у команді в основному колективна робота, зрілість процесів визначається за рівнем 3 моделі CMM.

Множники трудомісткості оцінюються з наступної інформації:

Кваліфікація персоналу – дуже низький рівень;

Необхідне виконання графіка робіт – середній рівень;

Досвід персоналу – низький рівень;

Складність і надійність продукту – середній рівень;

Складність платформи розробки – високий рівень;

Обладнання – низький рівень;

Розробка для повторного використання – високий рівень.

Варіант 8. Розмір проекту, який розробляється, оцінюється в 85 KLOC.

Відсутній досвід у продукті і платформі, у процесі розробки допускаються деякі компроміси, ризики проаналізовані на 20 %, у команді присутня частіше колективна робота, зрілість процесів визначається за рівнем 1 (вище середнього) моделі CMM.

Множники трудомісткості оцінюються з наступної інформації:

Кваліфікація персоналу – критично низький рівень;

Розробка для повторного використання – низький рівень;

Досвід персоналу – дуже низький рівень;

Обладнання – критично низький рівень;

Складність платформи розробки – середній рівень;

Необхідне виконання графіка робіт – дуже низький рівень;

Складність і надійність продукту – низький рівень.

Варіант 9. Розмір проекту, який розробляється, оцінюється в 400 KLOC.

Продукт і платформа велико мірою знайомі, у процесі розробки визначені тільки загальні цілі, ризики дозволені на 100 %, у команді присутня висока міра взаємодії, зрілість процесів визначається за рівнем 5 моделі CMM Level.

Множники трудомісткості оцінюються з наступної інформації:

Складність платформи розробки – високий рівень;

Досвід персоналу – критично високий рівень;

Складність і надійнсть продукту – критично високий рівень;

Кваліфікація персоналу – високий рівень;

Розробка для повторного використання – дуже високий рівень;

Необхідне виконання графіка робіт – високий рівень;

Обладнання – критично високий рівень.

Варіант 10. Розмір проекту, який розробляється, оцінюється в 7,5 KLOC.

Продукт і платформа не дуже знайомі, процес розробки строго детермінований, ризики проаналізовані на 60 %, у команді присутня важка до деякої міри взаємодія, зрілість процесів визначається за рівнем 2 моделі CMM.

Множники трудомісткості оцінюються з наступної інформації:

Необхідне виконання графіка робіт – низький рівень;

Досвід персоналу – дуже низький рівень;

Кваліфікація персоналу – критично низький рівень;

Складність і надійнсть продукту – низький рівень;

Складність платформи розробки – низький рівень;

Розробка для повторного використання – середній рівень;

Обладнання – високий рівень.

3. Провести детальну оцінку у попередньому завданні, вважаючи решта множників трудомісткості високого рівня.

4. Визначити час розробки за попередньою і детальною оцінкою.

5. Оцінити трудовитрати за попередньою і детальною оцінкою моделі COCOMO II, якщо розмір проекту та фактори масштабу відповідають розміру та факторам масштабу першого завдання, а множники трудомісткості подано нижче:

Варіант 1.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Можливості аналітика – низький рівень; можливості програміста – високий рівень; досвід використання мови програмування і інструментальних засобів – високий рівень, інші параметри персоналу середнього рівня. Розмір бази даних – середній рівень, інші параметри продукту низького рівня. Змінність платформи – низький рівень, інші параметри платформи середнього рівня. Використання інструментальних програмних засобів – високий рівень; необхідне виконання графіка робіт – середній рівень; багатоабонентська (видалена) розробка – низький рівень .

Варіант 2.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Досвід використання мови програмування і інструментальних засобів – високий рівень; досвід розробки додатків – дуже високий рівень;тривалість роботи персоналу– середній рівень; досвід роботи з платформою – низький рівень; інші параметри персоналу високого рівня. Складність програми – високий рівень; необхідна надійність програми – високий рівень; інші параметри продукту середнього рівня. Обмеження часу виконання – середній рівень; інші параметри платформи високого рівня. Параметри проекту середнього рівня, але використання інструментальних програмних засобів – дуже високий рівень.

Варіант 3.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Можливості програміста – середній рівень; досвід використання мови програмування і інструментальних засобів – високий рівень; тривалість роботи персоналу– високий рівень; інші параметри персоналу дуже високого рівня.

Параметри продукту високого рівня, але необхідна надійність програми – дуже високий рівень; складність програми – дуже високий рівень; розмір бази даних – середній рівень. Обмеження пам'яті – критично високий рівень; інші параметри платформи дуже високого рівня. Багатоабонентська (видалена) розробка – дуже високий рівень; інші параметри проекту високого рівня.

Варіант 4.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Тривалість роботи персоналу– дуже низький; можливості аналітика – середній рівень; досвід роботи з платформою – середній рівень; інші параметри персоналу низького рівня. Параметри продукту низького рівня, але необхідна надійність програми – середній рівень; необхідна можливість багатократного використання – середній рівень; розмір бази даних – дуже низький рівень. Параметри платформи середнього рівня, тільки змінність платформи – низький рівень. Використання інструментальних програмних засобів – низький; необхідне виконання графіка робіт – низький рівень; багатоабонентська (видалена) розробка – середній рівень.

Варіант 5.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Досвід розробки додатків – високий рівень; досвід використання мови програмування і інструментальних засобів – високий рівень; тривалість роботи персоналу – середній рівень; інші параметри персоналу дуже високого рівня Складність програми – дуже високий рівень; розмір бази даних – середній рівень; відповідність документації потребам життєвого циклу – дуже високий; необхідна можливість багатократного використання – середній рівень; інші параметри продукту високого рівня. Параметри платформи: обмеження часу виконання – дуже високий рівень; змінність платформи – високий рівень; обмеження пам'яті – критично високий рівень. Параметри проекту дуже високого рівня, проте необхідне виконання графіка робіт – високий рівень.

Варіант 6.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Параметри персоналу високого рівня, проте можливості аналітика – дуже високий рівень; досвід роботи з платформою – середній рівень; тривалість

роботи персоналу– низький рівень; можливості програміста – дуже високий рівень.Для параметрів продукту характерним є розмір бази даних – дуже високий рівень; необхідна надійність програми – високий рівень; необхідна можливість багатократного використання – дуже високий рівень, а інші параметри середнього рівня. Обмеження пам'яті платформи – високий рівень; обмеження часу виконання – критично високий рівень; інші параметри платформи дуже високого рівня. Для проекту характерним є необхідне виконання графіка робіт – дуже високий рівень; використання інструментальних програмних засобів – дуже високий рівень; багатоабонентська (видалена) розробка – високий рівень.

Варіант 7.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Досвід роботи з платформою – низький рівень; можливості програміста – високий рівень; досвід розробки додатків – дуже високий рівень; інші параметри персоналу середнього рівня. Параметри продукту високого рівня, проте кладність програми – дуже високий рівень; необхідна надійність програми – середній рівень; відповідність документації потребам життєвого циклу – дуже високий рівень. Для платформи характерним є обмеження пам'яті – середній рівень; обмеження часу виконання – високий рівень; інші параметри низького рівня. багатоабонентська (видалена) розробка – дуже високий рівень; необхідне виконання графіка робіт – високий рівень, інші параметри проектусереднього рівня.

Варіант 8.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Параметри персоналу низького рівня, проте досвід роботи з платформою – дуже низький рівень; можливості програміста – дуже низький рівень; досвід використання мови програмування і інструментальних засобів – середній рівень. Параметри продукту дуже низького рівня, проте складність програми – низький рівень; розмір бази даних – середній рівень; необхідна можливість багатократного використання – низький рівень. Обмеження пам'яті – середній рівень, інші параметри платформи низького рівня. Багатоабонентська (видалена) розробка – високий рівень; використання інструментальних програмних засобів – високий рівень; інші параметри проекту середнього рівня.

Варіант 9.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Можливості аналітика – високий рівень; тривалість роботи персоналу – середній;можливості програміста – високиий рівень; досвід використання мови програмування і інструментальних засобів – високий рівень; інші параметри персоналу дуже високого рівня. Параметри продукту критично високого рівня, проте відповідність документації потребам життєвого циклу – дуже високий рівень; розмір бази даних – високий рівень; складність програми – середній рівень. Обмеження часу виконання – дуже високий рівень; змінність платформи – дуже високий рівень; інші параметри платформи високого рівня. Необхідне виконання графіка робіт – дуже високий рівень; використання інструментальних програмних засобів – середній рівень; інші параметри проекту дуже високого рівня.

Варіант 10.

Множники трудомісткості для детальної оцінки визначаються з наступної інформації про рівні параметрів:

Можливості аналітика – середній рівень; тривалість роботи персоналу – середній рівень; досвід розробки додатків – дуже низький; досвід роботи з платформою – дуже низький рівень; інші параметри персоналу низького рівня. Ппараметри продукту середнього рівня, проте розмір бази даних– низький рівень; відповідність документації потребам життєвого циклу – низький рівень; складність програми – дуже низький рівень. Ппараметри платформи низького рівня, але обмеження часу виконання – середній рівень Параметри проекту середнього рівня, але багатоабонентська (видалена) розробка – високий рівень.

2.3. Контрольні запитання

1. Які існують стадії оцінки проекту у моделі COCOMO II.
2. Що таке Scale Drivers? Які фактори фаштабу ви знаєте?
3. Що таке Effort Multipliers? Які множники трудомісткості ви знаєте?
4. Який вигляд має формула оцінки трудомісткості проекту для попередньої оцінки проекту?
5. Який вигляд має формула оцінки трудомісткості проекту для детальної оцінки проекту?
6. Чи відрізняються фактори масштабу для різних стадій оцінки проекту?
7. Які ви знаєте фактори масштабу?
8. Чи відрізняються множники трудомісткості для різних стадій оцінки

проекту?

9. Виберіть серед вказаних множників трудомісткості PREX, PREX, RUSE, SCED, ACAP, AEXP, AEXP, AEXP, PERS, PCAP, PCON, PEXP, RUSE, LTEX, RELY, DATA, CPLX, FCIL, CPLX, RUSE, DOCU, TIME, STOR, PVOL, TOOL, SITE ті, які необхідно оцінити для стадії попередньої оцінки проекту?
10. Виберіть серед вказаних множників трудомісткості PREX, PREX, RUSE, SCED, ACAP, AEXP, AEXP, AEXP, PERS, PCAP, PCON, PEXP, RUSE, LTEX, RELY, DATA, CPLX, FCIL, CPLX, RUSE, DOCU, TIME, STOR, PVOL, TOOL, SITE ті, які необхідно оцінити для стадії детальної оцінки проекту?
11. За якою формулою розраховується час розробки проекту в методиці COCOMO II для Early Design?
12. За якою формулою розраховується час розробки проекту в методиці COCOMO II для Post Architecture?

РОЗДІЛ 3
ЗАСОБИ ОЦІНКИ ВАРТОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Теоретичний матеріал з теми розділу

Програма-калькулятор Costar 7.0 розроблена компанією Softstar (http://www.softstarsystems.com/) на основі моделі COCOMO II для автоматизації оцінки вартості разробки програмних продуктів.

В методиці використовуються п’ять факторів масштабу SF, які визначаються наступними характеристиками проекту:

PREC – прецедентність, наявність досвіду аналогічних розробок (Very Low – досвід в продуктіі платформі відсутній; Extra High – продукт і платформа повністю знайомі)

FLEX – гнучкість процесу розробки (Very Low – процес строго детермінований; Extra High – визначені тільки загальні цілі).

RESL – архітектура і дозвіл ризиків (Very Low – ризики невідомі / не проаналізовані; Extra High – ризики дозволені на 100%)

TEAM – спрацьованість команди (Very Low – формальна взаємодія; Extra High – повна довіра, взаємозаміна і взаємодопомога).

PMAT – зрілість процесів (Very Low – CMM Level 1; Extra High – CMM Level 5)

Параметри вартості. Параметр вартості (cost drivers або effort multipliers) – це суб’єктивна величина, яка оцінює різні часові, якісні і ресурсні аспекти розробки ПЗ. Кожний з параметрів може бути відкаліброваним. Калібрування параметрів вартості – це корегування значень параметрів, яка впливає на значення трудовитрат, і, відповідно, на вартість при оцінці програмного проекту. При калібруванні вказаних сімнадцяти параметрів вибирається оціночний рівень (дуже високий, високий, вище номінального, номінальний, нижче номінального, низький, дуже низький) параметру. В формулах цей рівень відображається у вигляді коефіцієнту трудовитрат і, таким чином, на кожній стадії розробки проекту впливає на вартість і тривалість тої або іншої стадії. Виділяють наступні групи параметрів: продукту (product factors), платформи (platform factors), персоналу (personnel factors) і проекту (project factors) (табл. 3.1). В табл. 3.2 наданий короткий опис кожного параметру.

Таблиця 3.1

Параметри	Опис
Продукту	Враховують характеристики ПЗ, яке розробляється. (RELY, DATA, CPLX, RUSE, DOCU)
Платформи	Враховують характеристики програмно-апаратного комплексу, який необхідний для функціонування ПЗ. (TIME, STOR, PVOL)
Персоналу	Враховують рівень знань і злагодженості роботи колективу програмістів. (ACAP, PCAP, PCON, APEX (AEXP), PLEX (PEXP), LTEX)
Проекту	Враховують вплив сучасних підходів і технологій, територіальної віддаленості членів колективу розробників і терміни виконання проекту. (TOOL, SITE, SCED)

Таблиця 3.2

Параметри	Опис
RELY (Required Software Reliability)	Враховує міру виконання програмою задуманої дії протягом певного часу
DATA (Database Size)	Враховує вплив об`єму тестових даних на розробку продукту. Рівень цього параметру розраховується як співвідношення байт в тестованій базі даних до SLOC в програмі
CPLX (Product Complexity)	Включає п`ять типів операцій: управління, рахункові, пристрій-залежні, управління даними, управління користувацьким інтерфейсом. Рівень складності це суб`єктивне середньо-зважене значення рівнів типів операцій
RUSE (Developed for Reusability)	Враховує трудовитрати, необхідні додатково для написання компонентів, призначених для повторного використання в даному або наступних проектах. Використовує наступні оціночні рівні: в проекті, в програмі, в лінійці продуктів, в різних

	лінійках продуктів. Значення параметру накладає обмеження на наступні параметри: RELY і DOCU
DOCU (Documentation Match To Life-Cycle Needs)	Враховує степінь відповідності документації проекту його життєвому циклу
TIME (Execution Time Constraint)	Враховує часові ресурси, які використовуються ПЗ, при виконанні поставленої задачі
STOR (Main Storage Constraint)	Враховує процент використання сховищ даних
PVOL (Platform Volatility)	Враховує термін життя платформи (комплекс апаратного і програмного забезпечення, який необхідний для функціонування ПЗ, який розробляється)
ACAP (Analyst Capability)	Враховує аналіз, здатність проектувати, ефективність і комунікативніздібності групи спеціалістів, якірозроблюютьвимогиі специфікації проекту. Параметр не повинен оцінювати рівень кваліфікаціїокремо взятого спеціаліста
PCAP (Programmer Capability)	Враховує рівень програмістів в колективі. При виборі значення для цього параметру слід особливо звернути увагу на комунікативні і професійні здібності програмістів і на командну роботу в цілому
PCON (Personnel Continuity)	Враховує плинність кадрів в колективі
APEX (ApplicationsExperience)	Враховує досвід колективу при роботі над додатками певного типу
PLEX (Platform Experience)	Враховує вміння використовувати особливості платформ, такі як графічний інтерфейс, бази даних, сітковий інтерфейс, розподілені системи
LTEX (Language and Tool	Враховує досвід програмістів (мови, середовище і інструменти)

Experience)	
TOOL (Use Of Software Tools)	Враховує рівень використання інструментів розробки
SITE (Multisite Development)	Враховує територіальну віддаленість (від офісу до міжнародних офісів) членів команди розробників засоби комунікації, які ними використовуються (від телефону до відео конференц-зв`язку)
SCED (Required Development Schedule)	Враховує вплив часових обмежень, накладених на проект і на значення трудовитрат

3.2. Практична робота № 3

Тема: Засоби оцінки вартості програмного забезпечення

Мета роботи: За допомогою програм-калькуляторів розрахувати вартість ПЗ на основі власних параметрів.

Завдання 1.

За допомогою програми SoftStar Systems Costar розрахувати вартість ПЗ на основі власних параметрів.

Запустити програму Costar 7.0 Demo.

Ввести в програму свої параметри:

1 крок – вибрати модель COCOMO II – ранню розробку проекту чи постархітектурну.

2 крок – ввести кількість рядків вихідного коду (SLOC).

3 крок – вибрати фактори масштабу (5 характеристик) (див. рис. 3.1-3.6)

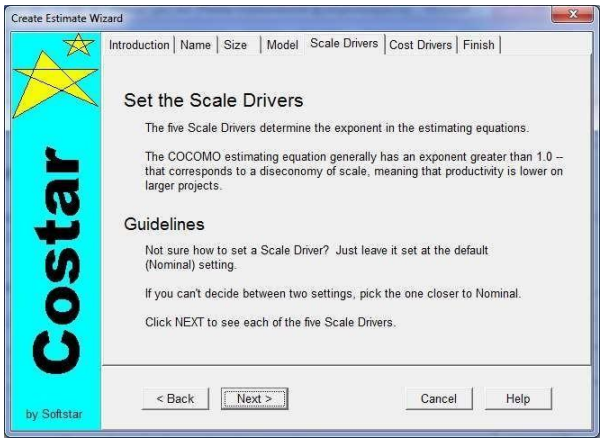


Рис. 3.1. Установка 5-ти факторів масштабу (Scale Drivers)

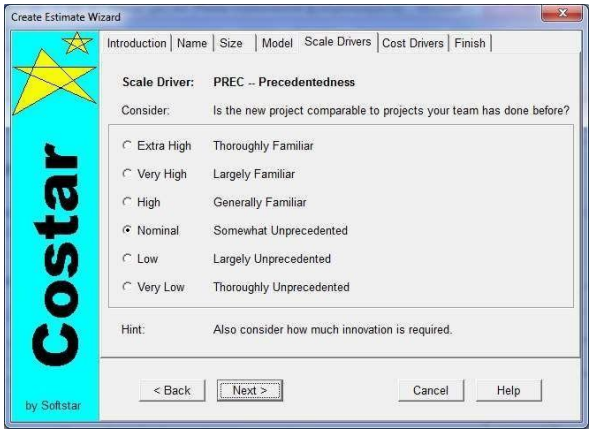


Рис. 3.2. Вибір рівня фактора масштабу «Прецедентність» (Precedentedness, PREC)

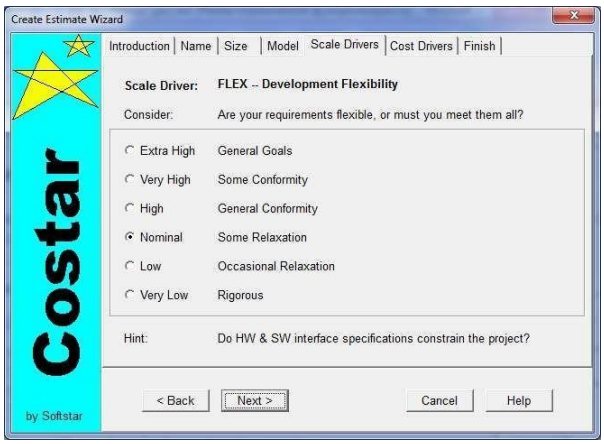


Рис. 3.3. Вибір рівня фактора масштабу «Гнучкість розробки» (Development Flexibility, FLEX.)

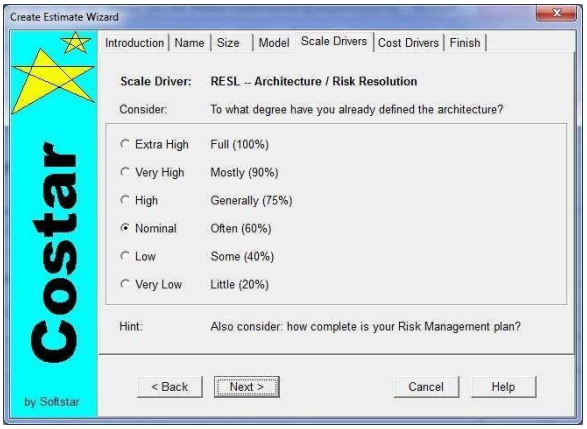


Рис. 3.4. Вибір рівня фактора масштабу «Архітектура/Дозвіл ризиків» (Architecture / Risk Resolution, RESL)

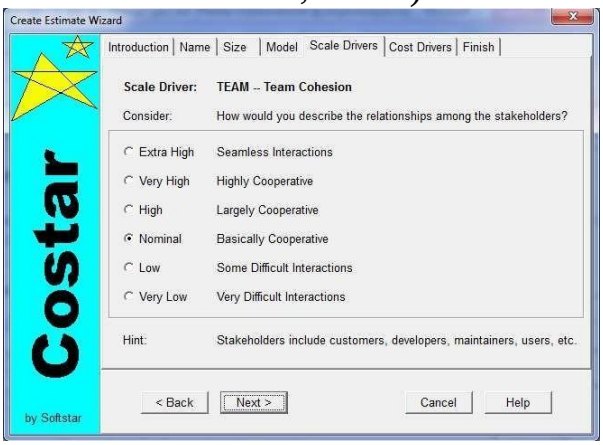


Рис. 3.5. Вибір рівня фактора масштабу «Спрацьованість команди» (Team Cohesion, TEAM)

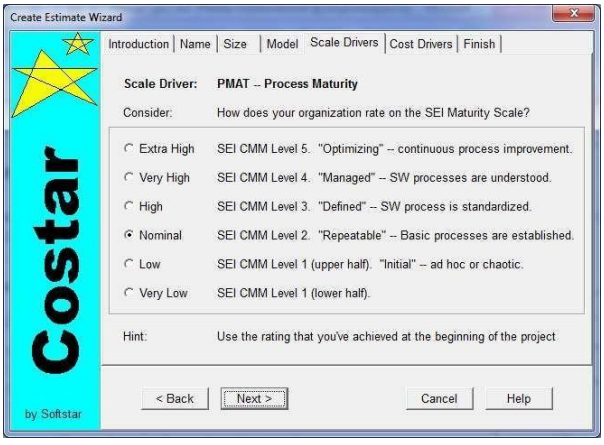


Рис. 3.6. Вибір рівня фактора масштабу «Зрілість процесів» (Process Maturity, PMAT)

4 крок – вибрати Параметри вартості (cost driver) – (17 характеристик) (див. рис. 3.7-3.9).

Рис. 3.7. Установка 17-ти факторів затрат (Cost Drivers або Effort Multipliers)

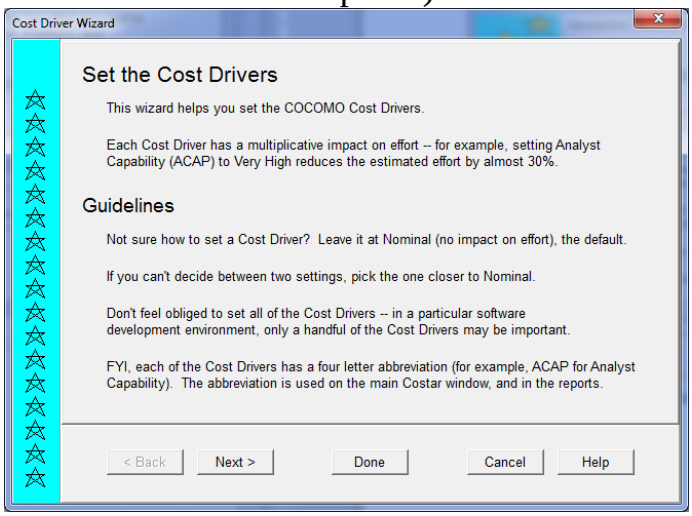


Рис.3.8. Методичні рекомендації для выбору факторів затрат. Наприклад, «якщо ви не впевнені у виборі фактору, то залиште його значення за замовчуванням (Nominal)»

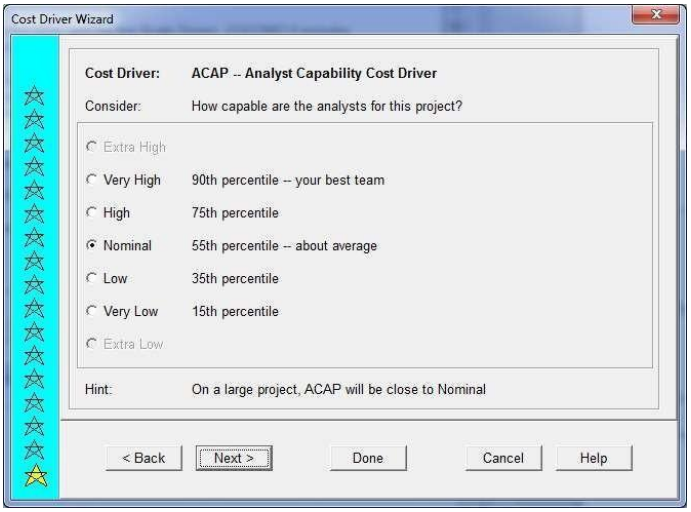


Рис. 3.9. Вибір рівня фактора затрат ACAP

5 крок – Отримати результат.

Після появи вікна з результатами (див. рис. 3.10) – перенести значення, які у червоному обрамлені, в закладку Costs (див. рис. 3.11)

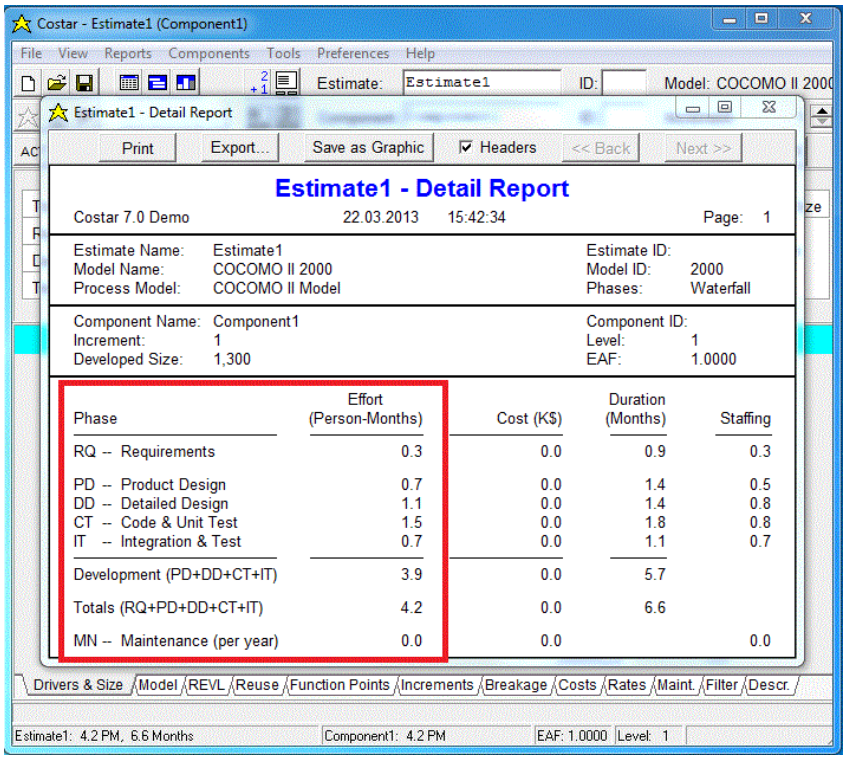


Рис. 3.10. Детальний звіт

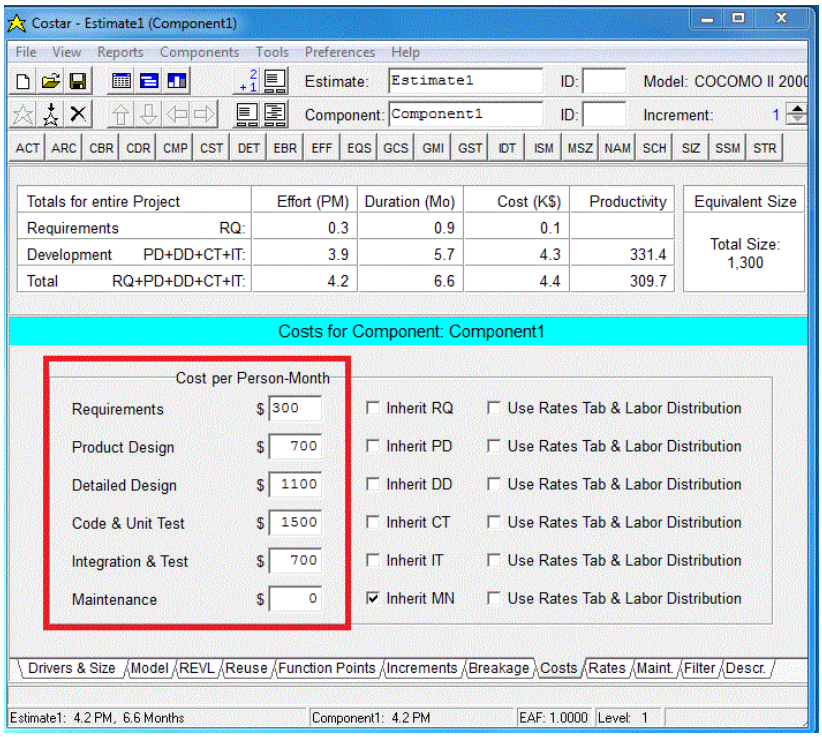


Рис. 3.11. Введення витрат

Вивести вікно результату, де будуть виведені зусилля 1 людино-місяця, строк розробки, вартість (див. рис. 3.12).

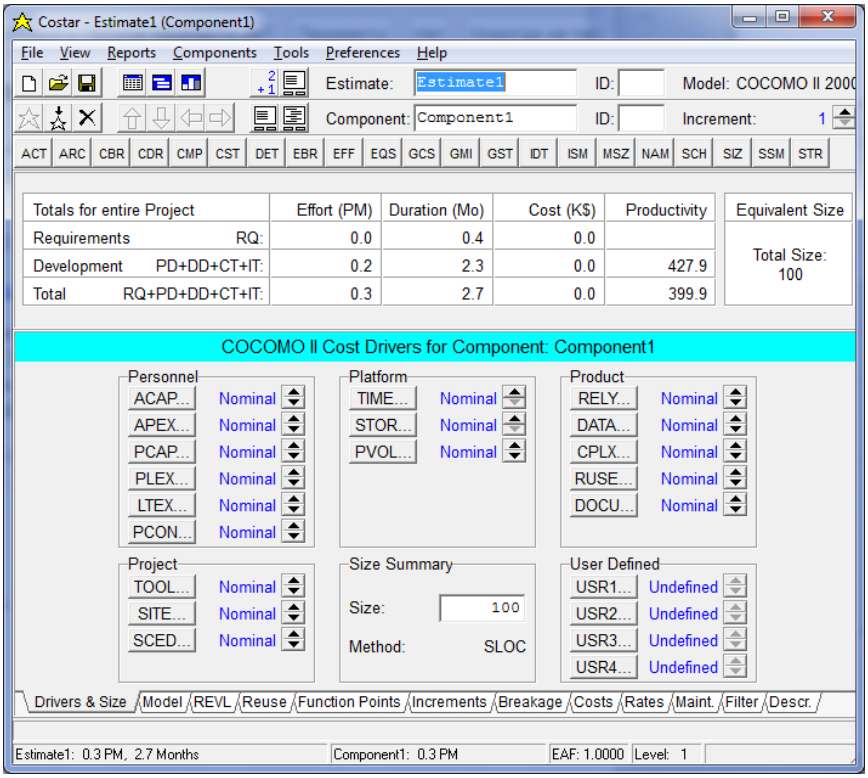


Рис. 3.12. Оцінка трудомісткості і часу виконання проектуа обсягом 100 SLOC

Завдання 2. За допомогою інтерфейс онлайн-калькулятора розрахувати вартість ПЗ на основі власних параметрів.

Розглянути інтерфейс онлайн-калькулятора COCOMO Suite of Constructive Cost Models на сайті Центру системного і програмного інжинірингу Університету Південної Кароліни (США) (USC Center for Systems and Software Engineering) (http://csse.usc.edu/tools/COCOMOSuite.php) (див. рис. 3.13).

Рис. 3.13. Інтерфейс інтернет-калькулятора «COCOMO Suite of Constructive Cost Models»

В правому верхньому куті розміщено вікно вибору моделі (див. рис. 3.14).

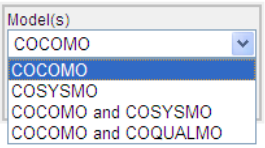


Рис. 3.14. Вікно вибору моделі

Тут:
COSYSMO – Constructive Systems Engineering Cost Model,
COQUALMO – Constructive Quality Model.

Ввести в програму свої параметри.

Завдання 3. Порівняти результати виконані у завданні 2 та завданні 3.

Зауваження. На сайті http://csse.usc.edu/csse/tools/ знаходяться посилання на інші програмні продукти разрахунків за моделями COCOMO, розроблені в USC CSSE.

3.3. Контрольні запитання

1. Для чого призначена програма Costar?
2. Ким розроблена програма Costar ?
3. Який фактор масштабу визначає прецедентність, наявність досвіду аналогічних розробок ?
4. Який фактор масштабу визначає гнучкість процесу розробки ?
5. Який фактор масштабу визначає архітектуру і дозвіл ризиків ?
6. Який фактор масштабу визначає спрацьованість команди ?
7. Який фактор масштабу визначає зрілість процесів ?
8. Які оціночні рівні є для факторів масштабу?
9. Які оціночні рівні є для параметрів вартості?
10. Що характеризують параметри RELY, DATA, CPLX, RUSE, DOCU?
11. Що характеризують параметри TIME, STOR, PVOL?
12. Що характеризують параметри ACAP, PCAP, PCON, APEX, PLEX, LTEX?
13. Що характеризують параметри TOOL, SITE, SCED?

РОЗДІЛ 4
МЕТРИКИ ОЦІНКИ СКЛАДНОСТІ ПРОГРАМНОЇ СИСТЕМИ. ЦИКЛОМАТИЧНА СЛАДНІСТЬ ЗА МАК-КЕЙБОМ

4.1. Теоретичний матеріал з теми розділу

Однією з груп оцінок складності програм є метрики складності потоку керування програм. Як правило, за допомогою цих оцінок оперують або щільністю керуючих переходів усередині програм, або взаємозв'язками цих переходів.

І в тому чи в іншому випадку стало традиційним уявлення програм у вигляді керуючого орієнтованого графа G (V, E), де V – вершини, які відповідають операторам, а E – дуги, що відповіднають переходам. У дузі (U, V) – вершина V є вихідною, а U – кінцевою. При цьому U безпосередньо слідує за V, а V безпосередньо передує U. Якщо шлях від V до U складається більш ніж з однієї дуги, тоді U слідує за V, а V передує U.

Для представлення програми використовується потоковий граф. Його особливостями є:

- Граф будується відображенням керуючої структури програми.
- У ході відображення закривають дужки умовних операторів і операторів циклів (end if; end loop) розглядаються як окремі (фіктивні) оператори.
- Вузли (вершини) потокового графа відповідають лінійним ділянкам програми, включають один або кілька операторів програми.
- Дуги потокового графа відображають потік управління у програмі (передачі управління між операторами). Дуга – це орієнтоване ребро.
- Розрізняють операторні і предикатні вузли. З операторного вузла виходить одна дуга, а з предикатного – дві дуги.
- Предикатні вузли відповідають простим умовам у програмі. Складна умова програми відображається в кілька предикатних вузлів. Складовим називають умова, в якому використовується одна чи кілька булевих операцій (OR, AND).
- Замкнуті області, утворені дугами і вузлами, називають регіонами.
- Середовище, що оточує граф розглядається як додатковий регіон.

Вперше графічне представлення програм було запропоновано Мак-Кейбом. Основною метрикою складності він пропонує вважати цикломатичну складність графа програми, або, ще називають, цикломатичне число Мак-Кейба, що характеризує трудомісткість тестування програми.

Цикломатичне складність – метрика програмного забезпечення, яка забезпечує кількісну оцінку логічної складності програми. У способі тестування базового шляху Цикломатична складність визначає:

- Кількість незалежних шляхів у базовому безлічі програми.
- Верхню оцінку кількості тестів, яке гарантує одноразове виконання всіх операторів.

Цикломатичну складність можна обчислити одним з трьох способів:

- Цикломатична складність дорівнює кількості регіонів потокового графа.
- Цикломатична складність визначається за формулою $Z(G) = m - n + 2$, де m – кількість дуг, n – кількість вузлів потокового графа.
- Цикломатичне складність формується як $Z(G) = p+1$, де p – кількість предикатних вузлів в потоковому графі G.

Для обчислення цикломатичного числа Мак-Кейба Z(G) застосовується формула

$$Z(G) = m-n+2p,$$

де m – число дуг орієнтованого графа G; n число вершин; p – число компонентів зв'язності графа.

Число компонентів зв'язності графа можна розглядати як кількість дуг, які необхідно додати для перетворення графа у сильнозв’язний.

Сильнозв’язний називається граф, будь-які дві вершини якого взаємнодосяжні. Для графів коректних програм, тобто графів, які не мають недосяжних від точок входу ділянок і "висячих" входу і виходу, сильнозв’язний граф, як правило, отримують шляхом замикання однієї вершини, що позначає кінець програми, на вершину, що позначає точку входу в цю програму.

Незалежним називається будь-який шлях, що вводить новий оператор обробки або нову умову. У термінах потокового графа незалежний шлях повинен містити дугу, що не входить у раніше визначені шляхи. Шлях починається у початковому вузлі, а закінчується в кінцевому

вузлі графа. Незалежний шлях формуються в порядку від найкоротшого до найдовшого.

По суті Z (G) визначає число лінійно незалежних контурів в сильнозв'язному графі. Інакше кажучи, цикломатичне число Мак-Кейба показує необхідне число проходів для покриття всіх контурів сильнозв'язного графа або кількість тестових прогонів програми, необхідних для вичерпного тестування за критерієм "працює кожна гілка".

Для програми, граф якої можна подати у вигляді див. рис. 4.2, цикломатичне число при m = 10, n = 8, p = 1 визначиться як Z (G)=10-8+2= =4.

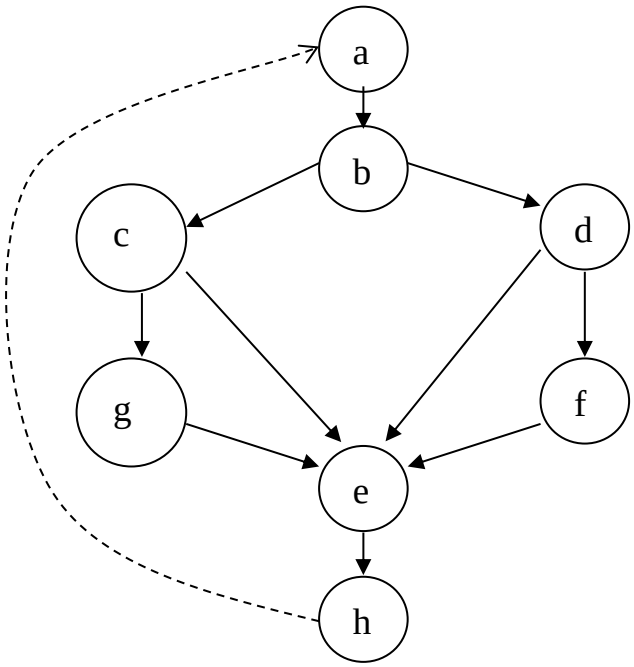


Рис. 4.1. Граф

Таким чином, маємо сильнозв'язний граф з чотирма лінійно незалежними контурами:

a-b-c-g-e-h-a;

a-b-c-e-h-a;

a-b-d-f-e-h-a;

a-b-d-e-h-a;

4.2. Практична робота №4

Тема: Метрики оцінки складності програмної системи.

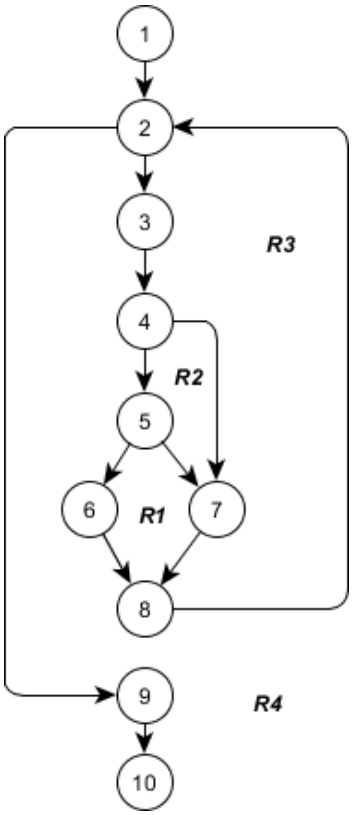
Цикломатична складність за Мак-Кейбом.

Мета роботи: навчитись визначати цикломатичну складность за Мак-Кейбом.

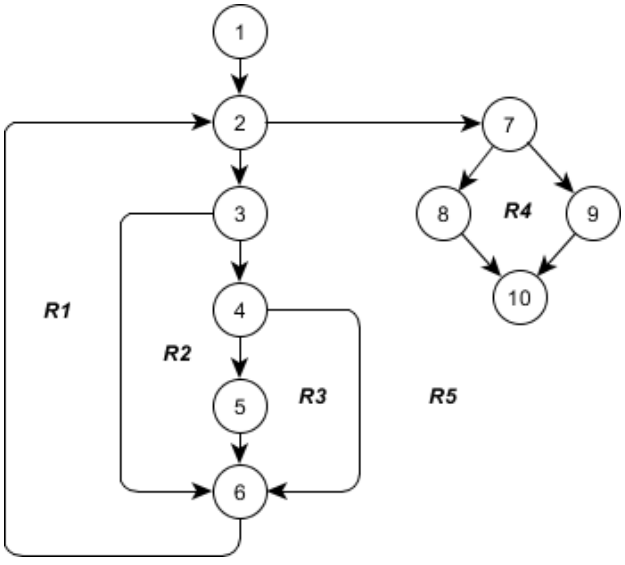
Завдання.

Знайти цикломатичну складність графа і цикломатичне число за варіантом.

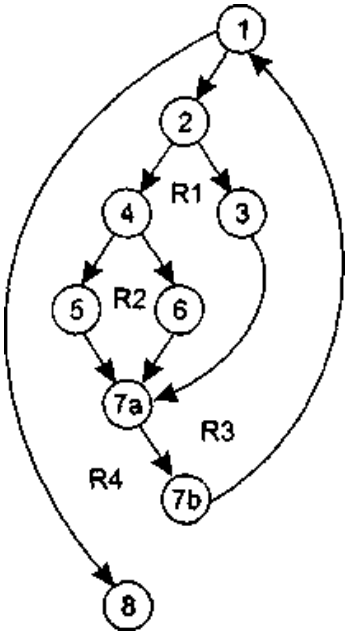
Варіант 1.



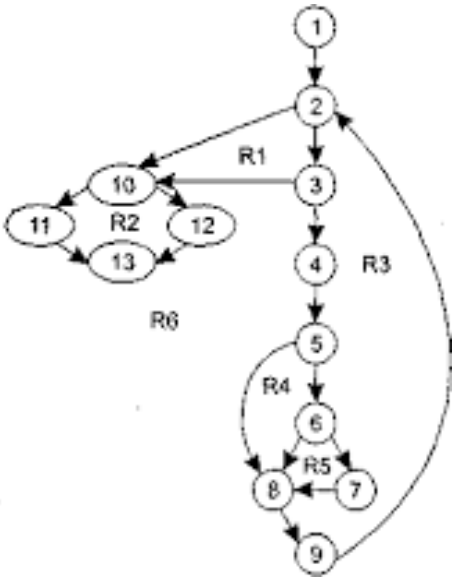
Варіант 2.



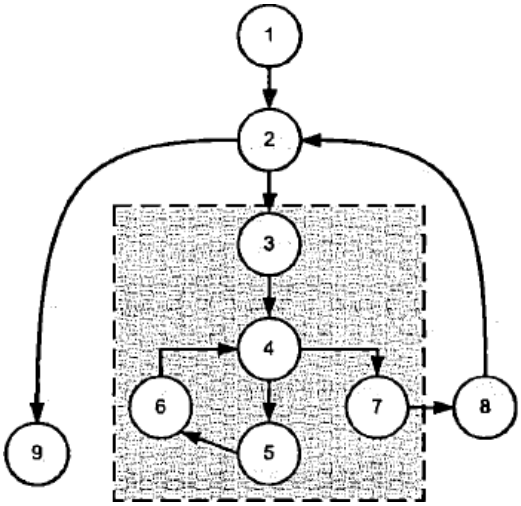
Варіант 3.



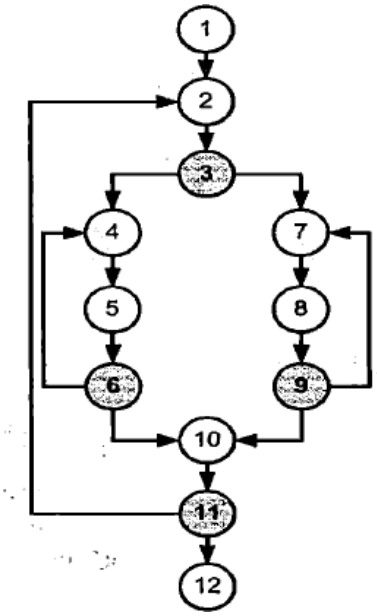
Варіант 4.



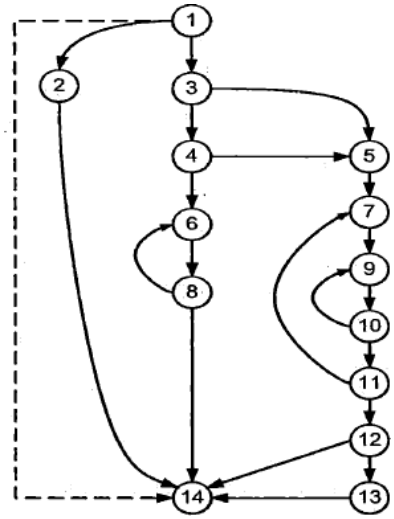
Варіант 5.



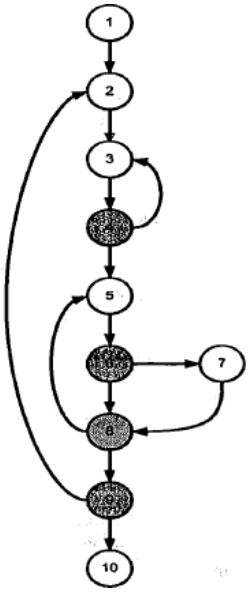
Варіант 6.



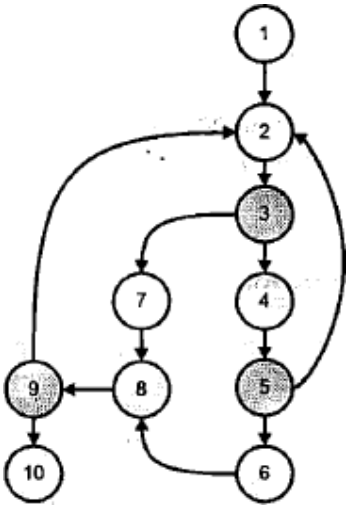
Варіант 7



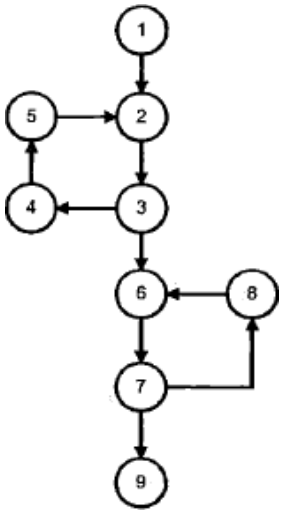
Варіант 8.



Варіант 9.



Варіант 10.



4.3. Контрольні запитання

1. Що таке керуючий орієнтований граф G (V, E)?
2. Які особливості потокового графа?
3. Що визначає цикломатична складність?
4. Що таке цикломатичне число Мак-Кейба?
5. Як обчислюється цикломатичне число Мак-Кейба?
6. Що таке сильнозв’язний граф?
7. Як зробити граф сильнозв’язним?
8. Який шлях обходу графа є незалежним?
9. Як формується незалежний шлях?
10. Що таке число компонент зв’язності?
11. Які способи знаходження цикломатичної складності ви знаєте?
12. Що визначає число лінійно незалежних контурів в сильнозв’язному графі.

РОЗДІЛ 5
МЕТРИКИ ОЦІНКИ СКЛАДНОСТІ ПРОГРАМНОЇ СИСТЕМИ (ХОЛСТЕДА, МАК-КЕЙБА, ДЖИЛБА, СПІНА, ЧЕПІНА)

5.1. Теоретичний матеріал з теми розділу

Метрика Холстеда. При застосуванні метрики Холстеда частково компенсуються недоліки, пов'язані з можливістю запису однієї і тієї ж функціональності різною кількістю рядків і операторів.

Використовуючи метрику Холстеда, визначають:

1) n1 – число різних операторів даної реалізації (унікальних операторів), включаючи символи-роздільники, імена процедур і знаки операцій;

2) n2 – число різних операндів даної реалізації (унікальних операндів);

3) N1 – загальне число всіх операторів;

4) N2 – загальне число всіх операндів.

На основі наведених вище характеристик знаходять:

а) n=n1+n2 – довідник (словник) програми;

б) N=N1+N2 – довжину програми;

в) $V = (N1+N2) * \log_2(n1+n2)$ – обсяг програми.

Кількість символів, що використовуються при реалізації деякого алгоритму, визначається в числі інших параметрів і словників програми n, що представляє собою мінімально необхідну кількість символів, що забезпечують реалізацію алгоритму.

Далі Холстедом введено n* – теоретичний словник програми, тобто словниковий запас, необхідний для написання програми з урахуванням того, що необхідна функція вже реалізована в даній мові і, отже, програма зводиться до виклику цієї функції. Наприклад, згідно з Холстедом, можливе здійснення процедури виділення простого числа могло б виглядати так:

CALL SIMPLE (X, Y),

де Y- масив чисельних значень, що містять шукане число X.

Теоретичний словник в цьому випадку буде складатися з

n1*: {CALL, SIMPLE (...)} n1* = 2;

n2*: {X, Y}, h2*= 2;

а його довжина, яка визначається як

n * = n1 * + n2 * дорівнюватиме 4.

Використовуючи n*, Холстед вводить оцінку V*: $V^* = n^* \log_2 n^*$, за

допомогою якої описується потенційний обсяг програми, що максимально компактно реалізує даний алгоритм.

–V*= n* * $\log_2$n* – теоретичний об'єм програми, де n* – теоретичний словник програми. Це потенціальний обсяг програми

– D – складність програми

– L – рівень програми (рівень якості програмування)

– λ – рівень мови

– N'= n1 * $\log_2$ (n1) + n2 * $\log_2$(n2) – теоретична довжина програми (для стилістично коректних програм відхилення N від N' не перевищує 10%)

– L = V*/ V - рівень якості програмування, для ідеальної програми L=1

– L'= (2 * n2) / (n1 * N2) – рівень якості програмування, заснований лише на параметрах реальної програми без урахування теоретичних параметрів,

EC = V / (L ') 2 – складність розуміння програми,

D = 1 / L ' – трудомісткість кодування програми,

y'= V / D2 – рівень мови вираження

I = V / D – інформаційний зміст програми, дана характеристика дозволяє визначити розумові витрати на створення програми

E = N '* log2 (n / L) - оцінка необхідних інтелектуальних зусиль при розробці програми, що характеризує число необхідних елементарних рішень при написанні програми.

Оцінка мови програмування λ = L2 * V. Вона характеризує рівень мови програмування.

Метрика Холстеда «інтелектуальності». За допомогою цієї метрики Голстед збирався вимірювати інтелектуальний зміст алгоритму, інваріантне по відношенню до мов реалізації. I = L * V.

Метрика Холстеда кількості помилок. Кількість помилок B = N / 3000, де N – кількість рядків у програмі.

Метрика Чепіна є метрикою складності потоку керування даних, суть методу полягає в оцінці інформаційної міцності окремо взятого програмного модуля за допомогою аналізу характеру використання змінних зі списку введення-виведення.

Всі безліч змінних, складових список введення-виведення,

розбивається на 4 функціональні групи:

1. P – вводяться змінні для розрахунків і для забезпечення виведення,

2. M – модифікуються, або створювані всередині програми змінні,

3. C – змінні, що беруть участь в управлінні роботою програмного модуля (керуючі змінні),

4. T – які не використовуються в програмі («паразитні») змінні.

Оскільки кожна змінна може виконувати одночасно кілька функцій, необхідно враховувати її в кожній відповідній функціональній групі.

Метрика Чепіна:

Q = a1*P + a2*M + a3* C + a4*T, де a1, a2, a3, a4 – вагові коефіцієнти.

Вагові коефіцієнти використані для відображення різного впливу на складність програми кожної функціональної групи. Найбільшу вагу, рівний 3, має функціональна група C, так як вона впливає на потік управління програми. Вагові коефіцієнти інших груп розподіляються наступним чином: a1 = 1, a2 = 2, a4 = 0.5. Ваговий коефіцієнт групи T НЕ дорівнює 0, оскільки «паразитні» змінні не збільшують складність потоку даних програми, але іноді ускладнюють її розуміння. З урахуванням вагових коефіцієнтів:

$$Q = P + 2M + 3C + 0.5T$$

Метрика спіна грунтується на локалізації звернень до даних усередині кожної програмної секції. Спін – це число тверджень, які містять даний ідентифікатор, між його першим і останнім появою в тексті програми. Отже, ідентифікатор, що з'явився n раз, має спін, рівний n-1. При великому значенні спіна ускладнюється тестування і налагодження.

Метрика Джилбі. Вона відноситься до кількісних метрик і показує складність програмного забезпечення на основі насиченості програми умовними операторами або операторами циклу. Дана метрика, не дивлячись на свою простоту, досить добре відображає складність написання і розуміння програми, а при додаванні такого показника, як максимальний рівень вкладеності умовних і циклічних операторів, ефективність даної метрики значно зростає.

Однією з найбільш простих, але досить ефективних оцінок складності програм є метрика Т. Джілбі, в якій логічна складність програми визначається як насиченість програми виразами IF_THEN_ELSE. При цьому вводяться дві характеристики:

1) CL – абсолютна складність програми, що характеризується кількістю операторів умови;

2) cl – відносна складність програми, що характеризується насиченістю програми операторами умови, тобто cl визначається як відношення CL до загальної кількості операторів.

Використовуючи метрику Джілбі, її доповнили ще однією складовою, а саме характеристикою максимального рівня вкладеності оператора CLI, що дозволило застосувати метрику Джілбі до аналізу циклічних конструкцій.

5.2. Практична робота № 5

Тема: Метрики оцінки складності програмної системи (Холстеда, Мак-Кейба, Джилба, Спіна, Чепіна).

Мета роботи: Навчитись виконувати розрахунок метрик Холстеда, Мак-Кейба, Джилба, Спіна, Чепіна.

Завдання.

Для заданих програм – лінійної програми, програми з вказівниками та програми з модулями (див. Додатки А,Б,В) виконати:

1. Розрахунок метрики Холстеда. Словники операторів та операндів подати у вигляді таблиці для кожної програм (див. Табл. 4.1-4.3):

Таблиця 4.1

Метрика Холстеда для лінійної програми

Словник операторів (оператори, кількість)		Словник операндів (операнди, кількість)	

Таблиця 4.2

Метрика Холстеда для програми з вказівниками

Словник операторів		Словник операндів	

Таблиця 4.3

Метрика Холстеда для програми з модулями

Словник операторів		Словник операндів	

На основі наведених вище характеристик вичислити словник програми; довжину програми; теоретичну довжину програми, об’єм програми, рівень програми з без оцінки теоретичого обсягу, інформаційний зміст програми.

2. Зобразити графи для програм та обчислити цикломатичну складність за Мак-Кейбом.

3. Здійснити розрахунок метрики Чепіна (див. табл. 4.4).

Таблиця 4.4

Метрика Чепіна для програм

	Лінійна програма	Програма з вказівниками	Програма з модулями
P			
M			
C			
Q			

4. Розрахунок метрики спіна (див. табл. 4.5).

Таблиця 4. 5

Метрика спіна для програм

	Линійна програма	Програма з вказівниками	Програма з модулями
Choice			
File			
Num			
Path			
St			

5. Розрахунок метрики Джилба.

Додаткове завдання

Написати програму, яка виконує наступні функції:

• Введення даних про товари магазину;
• Запис в файл на диску;
• Пошук запису;
• Висновок на екран знайдених записів.

Програму необхідно написати в трьох варіантах:

• Лінійний (весь текст в одному модулі);
• З використанням модулів для кожної функції;
• З використанням вказівників.

Провести розрахунок метрик створених програм.

5.3. Контрольні запитання

1. Як визначається словник програми у метриці Холстеда?
2. Як визначається довжина програми у метриці Холстеда?
3. Як визначається обсяг програми у метриці Холстеда?
4. Що таке теоретичний словник програми?
5. Як визначається рівень якості програмування?
6. На які функціональні групи розбиваються змінні у метриці Чепіна?
7. Які вагові коефіцієнти функціональних груп змінних у метриці Чепіна?
8. Що таке спін?

9. Чи ускладнюється тестування і налагодження програми при великому значенні спіна?
10. Чим характеризується метрика Джилбі?
11. Чим визначається логічна складність програми у метриці Джилбі?
12. Чим відрізняється абсолютна та відносна складності програмимерики Джилбі?

РОЗДІЛ 6
МЕТОД ФУНКЦІОНАЛЬНИХ ТОЧОК

6.1. Теоретичний матеріал з теми розділу

Покроковий алгоритм оціювання програми на основі методу функціональних точок.

Крок 1. Підрахунок функцій в кожній категорії

- Зовнішні вводи (введення даних користувачів).
- Зовнішні виводи (звіти, екрани, роздрукування, повідомлення).
- Зовнішні запити (діалогові вводи-виводи).
- Локальних внутрішні логічні файли (звернення до текстових файлів).
- Зовнішні інтерфейсні файли.

Крок 2. Застосування вагових множників складності (табл. 6.1-6.5).

Необхідно помножити кожну величину конкретного типу (простий, середній, складний) всередині кожної категорії (вивід, введення, запити, файли, інтерфейси) на відповідний ваговий множник.

Таблиця 6.1

Інструкції по визначенню рівня складності для вводу

Посилання на файли	Елементи даних		
	1-4	5-15	>15
0-1	Низький (3)	Низький (3)	Середній (4)
2	Низький (3)	Середній (4)	Високий (6)
>2	Середній (4)	Високий (6)	Високий (6)

Таблиця 6.2

Інструкції по визначенню рівня складності для виводу

Посилання на файли	Елементи даних		
	1-4	5-19	>19
0-1	Низький (4)	Низький (4)	Середній (5)
2-3	Низький (4)	Середній (5)	Високий (7)
>3	Середній (5)	Високий (7)	Високий (7)

Таблиця 6.3

Інструкції по визначенню рівня складності для запиту

Посилання на файли	Елементи даних		
	1-4	5-19	>19
0-1	Низький (3)	Низький (3)	Середній (4)
2-3	Низький (3)	Середній (4)	Високий (6)
>3	Середній (4)	Високий (6)	Високий (6)

Таблиця 6.4

Інструкції по визначенню рівня складності для внутрішніх логічних файлів

Типи елементів записів	Елементи даних		
	1-19	20-50	>50
1	Низький (7)	Низький (7)	Середній (10)
2-5	Низький (7)	Середній (10)	Високий (15)
>5	Середній (10)	Високий (15)	Високий (15)

Таблиця 6. 5

Інструкції по визначенню рівня складності для зовнішніх інтерфейсних файлів

Типи елементів записів	Елементи даних		
	1-19	20-50	>50
1	Низький (5)	Низький (5)	Середній (7)
2-5	Низький (5)	Середній (7)	Високий (10)
>5	Середній (7)	Високий (10)	Високий (10)

В кожну категорію додаються підсумкові результати, які виражені в кількості «фізичних функціональних точок».

Крок 3. Застосування факторів середовища

Необхідно оцінити кожний фактор середовища (див. Табл. 6.6) за шкалою від 0 до 5 (в даному випадку 0 означає неможливість застосування фактора). 1– вплив випадковий, 2 – незначний, 3 – вплив середній, 4 – значний, 5 – вплив основний. Фактори середовища (1. канали передачі даних, 2. Розподілена обробка даних, 3. Вимоги до продуктивності, 4. Поширеність використовуваної конфігурації, 5. Частота (швидкість) транзакцій, 6. Інтерактивний запит (запис), оперативний ввід даних, 7. Ефективність на рівні кінцевого користувача, 8. Інтерактивне обновлення,

9. Складна обробка, 10. Повторне використання, 11. Спрощене перетворення/встановлення, 12. Спрощення операції, 13. Використання на декількох вузлах, 14. Потенціал зміни функції).

Таблиця 6.6

Опис факторів середовища

№	Фактор середовища	Рейтинг(0, 1, 2, 3, 4, 5)
1	Обмін даними (Data Communi-cations)	Ступінь необхідності обміну даними для ПС, його комунікційний можливості: 0-ПС реалізована як єдиний пакет на автономному комп'ютері; 1-ПС реалізована як єдиний пакет, але має віддалений введення даних або віддалений висновок (друк); 2-ПС реалізована як єдиний пакет, але має віддалений введення даних і віддалений висновок (друк); 3-ПС включає в себе окремий інтерфейсний блок або клієнтську частину, призначену для збору або віддаленої обрбки даних в режимі реального часу (on line); 4-в ПС в повному обсязі використані клієнт-серверні технології, але підтримується тільки один тип телекоммунікаціційного протоколу; 5-в ПС в повному обсязі використані клієнт-серверні технології і підтримується більш як один тип телекоммунікаціційного протоколу.
2	Розподілені функції (Distributed Functions)	Наявність в ПС функцій підтримки розподіленої обробки даних: 0-дані між компонентами ПС і системи не передаються; 1-ПС готує дані для кінцевої обробки іншими засобами, наприклад, електронними таблицями або СУБД; 2- ПС готує дані, які потім передаються для обробки (але не для кінцевої) інший компоненті системи; 3- ПС підтримує розподілену обробку і передачу даних в режимі реального часу (on line), але тільки в одному напрямку; 4- ПС підтримує розподілену обробку і передачу даних в режимі реального часу (on line) в обох

		напрямках; 5-функції обробки даних динамічно виконуються на найбільш підходящій компоненті системи
3	Продуктивність (Performance)	Ступінь критичності вимог до продуктивності ПС: 0-- ніяких спеціальних вимог до продуктивності ПС користувачем не було встановлено 1-вимоги до продуктивності і проектування ПС були встановлені і розглянуті, але щоб їх задовольнити, ні- яких спеціальних заходів не треба було; 2-час відгуку або пропускна здатність ПС критичні в певні пікові періоди, проте проектування ПС з урахуванням завантаження центрального процесора не потрібно; 3-час відгуку або пропускна здатність ПС критичні протягом усього робочого періоду періоди, проте проектування ПС з урахуванням завантаження центрального процесора не потрібне; 4-на додаток, вимоги до продуктивності ПС, установлені користувачем, досить жорсткі, тому на етапі проектування потрібний аналіз продуктивності; 5- на додаток, для того, щоб досягти вимог користувача, на етапах проектування, реалізації і впровадження потрібним є використання спеціальних інструментів аналізу про- тивності.;
4	Інтенсивно використовувана конфігурація (Heavily Used Configuration)	Інтенсивність використання обладнання, на якому буде встановлено ПС: 0 - явних чи неявних обмежень на використання ресурсів обладнання не встановлено; 1-операційні обмеження існують, але вони менші, ніж у типовому додатку, і тому спеціальних зусиль для їх задоволення не буде потрібно; 2-деякі вимоги з безпеки і часу, визвані функціонуванням інших ПС, присутні; 3-присутні спеціальні вимоги до процесора, пов’язані з роботою окремих частин ПС; 4-встановлені операційні обмеження вимагають лімітного використання ПС центрального або віддаленого процесора; 5-на додаток, існують спеціальні обмеження, які

		накладаються на ПС, в розподілених компонентах системи
5	Інтенсивність транзакцій Transaction Rate	0 - пікові періоди транзакцій годі чекати; 1-пікові періоди транзакцій очікуються щорічно, щосезонно, щоквартально, щомісячно; 2-пікові періоди транзакцій очікуються щотижня; 3 - пікові періоди транзакцій очікуються щодня; 4-інтенсивність транзакцій, встановлена користувачем у вимогах до ПС або угодах про рівень обслуговування, досить висока, щоб вимагати аналізу продуктивності на етапі проектування; 5-інтенсивність транзакцій, встановлена користувачем у вимогах до ПС або угодах про рівень обслуговування, досить висока, щоб вимагати аналізу продуктивності з використанням спеціальних інструментів на етапах проектування, реалізації та впровадження
6	Діалоговий ввід даних (On-line Data Entry)	Складність діалогових транзакцій з урахуванням числа екранівфункцій: 0-всs транзакції обробляються в пакетному режимі; 1-від 1% до 7% транзакцій є інтерактивним введенням даних; 2-від 8% до 15% транзакцій є інтерактивним введенням даних$ 3-від 16% до 23% транзакцій є інтерактивним введенням даних; 4-від 24% до 30% транзакцій є інтерактивним введенням даних; 5-більше 30% транзакцій є інтерактивним введенням даних.
7	Ефективність для кінцевого користувача(End User Efficiency)	Ступінь підтримки діалоговими функціями ефективності для кінцевого роботи кінцевого користувача оцінюється за наявністю в ПС наступних елементів призначеного користувацького інтерфейсу: Допомога при навігації (функціональні клавіші, посилання, динамічне меню); меню; on-line допомога і документація; автоматична установка курсору; скролінг; віддалена друк (за допомогою on-line

		транзакцій); запрограмованих користувачем функціональні клавіші; пакетна обробка даних через on-line транзакції; вибір екранних даних за допомогою курсору; інтенсивне використання негативного зображення, підсвічування, кольорового підкреслення і інших указу- телей; друкована копія користувальницької документації; інтерфейс з мишею; вспливаючі вікна; виконання бізнес функцій якомога більшою коліче- ством екранів; підтримка двох мов (вважається як 4 пункту); багатомовна підтримка (вважається як 6 пунктів). 0-все вищеперелічене відсутній 1-присутні від 1 до 3 пунктів 2-присутні від 4 до 5 пунктів 3-присутні 6 і більше пунктів, але немає особливих користувацьких вимог до ефективності; 4-присутні 6 і більше пунктів, і встановлені вимоги до ефективності для кінцевого користувача досить суворі, щоб потрібні були спеціальні дослідження людського фактора (наприклад, мінімізація натискань на клавіші, макси- мизация замовчувань, використання шаблонів); 5-присутні 6 і більше пунктів, і встановлені вимоги до ефективності для кінцевого користувача досить суворі, щоб потрібні були спеціальні інструменти для демон- ції досягнення поставлених цілей.
8	Оперативне обновлення (On-line Update)	- Використання оперативного оновлення основних файлів: 0-відсутнє; 1-присутнє оперативне оновлення від 1 до 3 управляючих файлів. Обсяг оновлень невеликий відновлення просте; 2-присутній оперативне оновлення більш ніж 3 управляючих файлів. Обсяг оновлень невеликий відновлення просте;

		3-присутній оперативне оновлення більшої частини внутрішніх логічних файлів; 4-на додаток присутній спеціально розроблена захист від втрати даних; 5- на додаток процеси відновлення вимагають великих затрат. Процедури відновлення високо автоматизовані з мінімальним втручанням оператора
9	Складність обробки даних(Complex Processing)	Складність обробки даних в ПС оцінюється за наявністю: • чутливих керувань. Наприклад, спеціальних перевірок при обробці даних) і / або спеціальної обробки даних для забезпечення безпеки) • широкої логічної обробки даних; • широкої математичної обробки даних; • великої кількості оброблюваних винятків в результаті незавершених транзакцій, наприклад при незавершених транзакцій, викликаних припиненням телеобробки, зниклих значеннях даних, невдалих виправлення; • складною обробкою при управлінні множинними можливостями введення / виведення, наприклад, для мультімедіа або незалежних пристроїв. Складність обробки даних: 0-все вищеперелічене відсутній; 1-є щось одне з перерахованого вище 2-присутні будь-які 2 з перерахованих вище пунктів; 3-присутні будь-які 3 з перерахованих вище пунктів; 4-присутні якісь 4 з перерахованих вище пунктів 5-присутні всі 5 з перерахованих вище пунктів.
10	Повторне використання (Reusability)	Оцінка вихідного коду з точки зору його повторного використання в інших ПС: 0-в ПС немає коду, призначеного для повторного вико вання; 1-в ПС присутній код, призначений для повторного використання; 2-менше 10% коду ПС розглядаються як необхідні для більш, ніж одного користувача; 3-10% коду ПС і більш розглядаються як необхідні для більш, ніж одного користувача; 4-ПС спеціально розроблено і / або

		документовано, щоб полегшити повторне використання коду і його настроювання поль зователем проводиться на рівні вихідних кодів; 5-ПС спеціально розроблено і / або документовано, щоб полегшити повторне використання коду і його налаштування для застосування проводиться за допомогою підтримки (технічно-го обслуговування) параметрів користувача.
11	Легкість установлення (інсталяції) (Installation Ease)	Ступінь легкості процесу установки (інсталяції) і налаштування ПС: 0 - немає особливих вимог користувача, і не потрібно спеціальної установчої програми; 1 - немає особливих вимог користувача, але потрібно спеціальна установча програма; 2 - вимоги до переносу даних з одного носія на інший і до інсталяції сформульовані користувачем, керівництво по перенесенню і інсталяції представлені і протестовані. Вплив перенесення даних на проект не вважається важливим; 3 - вимоги до переносу даних з одного носія на інший і до інсталяції сформульовані користувачем, керівництва по перенесенню і інсталяції представлені і протестовані. Вплив перенесення даних на проект вважається важливим; 4 - на додаток до пункту 2 надані і протестовані автоматизовані інструменти для перенесення даних і інсталяції; 5 на додаток до пункту 3 надані і протестовані автоматизовані інструменти для перенесення даних і інсталяції.
12	Простота використання (Operational Ease)	Простота використання - ефективність ПС при виконанні основних операцій таких, як пуск, резервне копіювання, резервной відновлення, і т.п .: 0-немає особливих вимог користувача за винятком нормальної процедури резервного копіювання; 1-4-один, кілька або всі з перерахованих нижче пунктів можуть бути застосовані до даного ПС. Кожен пункт має вагу рівну 1, крім особливо обумовленого випадку:

		− ефективні процедури завантаження, резервного копіювання і відновлення надані, але втручанняоператора все ж потрібно; − ефективні процедури завантаження, резервного копіювання і відновлення надані, втручання оператора не потрібно (вважати цей пункт з вагою 2); − ПС мінімізує необхідність монтування стрічки; − ПС мінімізує необхідність обробки документів; 5-ПС спроектовано для автоматичного функціонування. Це означає, що не потрібне втручання оператора в управління системою, за винятком процедур завантаження і виключення. ПС має функцію автоматичного відновлення після помилок.
13	Поширюваність (Multiple Sites)	Можливість установки ПС на різні обчислювальні системи в різних організаціях: 0 - ПС не розраховане на використання більш ніж одним користувачем або на установку більш ніж на один комп'ютер; 1 - при проектуванні вимоги по установці ПС на декілька комп'ютерів були враховані, причому, дане ПС може виконуватися тільки на ідентичному апаратному та програмному забезпеченні; 2 - при проектуванні вимоги по установці ПС на декілька комп'ютерів були враховані, причому, дане ПС може виконуватися тільки на схожому (сумісному) апаратному та / або програмному забезпеченні; 3 - при проектуванні вимоги по установці ПС на декілька комп'ютерів були враховані, причому, дане ПС може виконуватися на різному апаратному та / або програмному забезпеченні; 4 - наведені і протестовані документація і план підтримки по установці ПС на кілька комп'ютерів, а саме ПС удов- летворяет пунктам 1 або 2 цього переліку; - наведені і протестовані документація і план підтримки по установці ПС на кілька комп'ютерів, а саме ПС удов- летворяет пункту 3 даного переліку.

14	Легкість зміни (Facilitate Change)	Легкість пристосування ПС до змін, що вносяться користувачем, оцінюється по наявності: - можливості організації гнучких запитів і звітів, що обробляють прості вимоги, наприклад, примі няющих логічні операції and / or тільки до одного внутрішнього логічного файлу (вважається як 1 пункт); - можливості організації гнучких запитів і звітів, що обробляють вимоги середньої складності, наприклад, застосовують логічні операції and / or більш ніж до одного внутрішнього логічного файлу (вважається як 2 пункти); - можливості організації гнучких запитів і звітів, що обробляють складні вимоги, наприклад, примі няющих комбінації логічних операцій and / or до од- ному або більш внутрішньому логічному файлу (вважається як 3 пункти); - ділова інформація зберігається в таблицях, які управляються і підтримуються користувачем в діалогів говом режимі (on-line), але зміни вступають в силу тільки з наступного робочого дня; - ділова інформація зберігається в таблицях, які управляються і підтримуються користувачем в діалогів говом режимі (on-line), і зміни вступають в силу негайно (вважається як 2 пункти). Легкість зміни: 0 - все вищеперелічене відсутні; 1 - присутній 1 пункт з перерахованих вище; 2 - присутні 2 пункти з перерахованих вище; 3- присутні 3 пункту з перерахованого вище; 4 - присутні 4 пункту з перерахованого вище; 5 - присутні 5 пунктів з перерахованого вище.

Крок 4. Обчислення множника корегування складності

CAF = 0.65 + (0.01 * N), де N є сумою зважених факторів середовища.

Оскільки доводиться мати справу з 14 передбачуваними факторами середовища, кожен з яких має вагу, що змінюється в діапазоні від 0 до 5, найменше значення для N може бути 0 (жоден з 14 не використовується); а

найбільше значення для N може бути 70 (кожен з 14 факторів має максимальну вагу, рівний 5). Виходячи з цих граничних умов, приходимо до висновку, що мінімум CAF = 0,65+(0,01*0) = 0,65. Максимум CAF = 0,65+(0,01* 70) = 1,35. (1,35 – 0,65 = 0,70). Рання оцінка розмірів і трудовитрат може відхилятися за умови використання фактора на величину +/- 35%.

Крок 5. Перетворення в рядки LOC

Метод функціональних точок забезпечує спосіб попередньої оцінки розміру потенційних програм або програмних систем. При цьому здійснюється аналіз майбутніх функціональних властивостей з користувальницької точки зору. Мови програмування є дуже різними з точки зору їх характеристик, однак існує деяка середня кількість операторів, які виконуються, необхідних для реалізації однієї функціональної точки (див. Додаток Г).

6.2. Практична робота № 6

Тема: Метод функціональних точок.

Мета роботи: Навчитися оцінювати характеристики програм на основі методу функціональних точок.

Завдання 1.

Для розгляду представлено реалізацію програми «Заміна цифр на символ» при розв’язанні задачі: визначити клас з методом, який виконує заміну в рядку всіх цифр на символ-замінник; рядок і символ-замінник прийняти через вхідні параметри; рядок повертається через return.

Текст програми для реалізації можливого вирішення поставленої задачі, розроблено з використанням мови програмування C # (див. рис.6.1).

Номера строк	Строки программы
1	using System;
2	class Mymetod
3	{
4	public string change(string sinp, char sim)
5	{
6	string buf;
7	int i;
8	for (i=0,buf=""; i<sinp.Length; i++)
9	if(char.IsDigit(sinp[i]))
10	buf += sim;
11	else
12	buf += sinp[i];
13	return buf;
14	}
15	public static void Main()
16	{
17	string str;
18	char sim, rep;
19	do
20	{
21	Console.WriteLine("Исходная строка:");
22	str = Console.ReadLine();
23	Console.Write("Заменитель: ");
24	sim = char.Parse(Console.ReadLine());
25	str = change(str,sim);
26	Console.WriteLine("Модифицированная строка:");
27	Console.Write(str);
28	Console.Write("\nДля повтора намите клавишу Y: ");
29	rep = char.Parse(Console.ReadLine());
30	Console.WriteLine();
31	}while(rep == 'Y' || rep == 'y');
32	}
33	}

Рис. 6.1. Реалізація задачі “Заміна цифр на символ” з використанням мови програмування C #.

Заповнити робочий лист згідно кроків 1)-5) (див табл. 6.7):

Таблиця 6.7

Робочий лист аналізу за методом функціональних точок

Крок 1	Підрахунок кількість функцій в кожній категорії			
Крок 2	Застосування вагових коефіцієнтів складності			
	Простий	Середній	Складний	Функціональні точки
Кількість вводів	_*3	_*4	_*6	
Кількість виводів	_*4	_*5	_*7	
Кількість запитів	_*3	_*4	_*6	
Кількість файлів	_*7	_*10	_*15	
Кількість інтерфейсів	_*5	_*7	_*10	
Загальна кількість функціональних точок				
Фактори середовища				Рейтинг (0,1,2,3,4,5)
Канали передачі даних				
Розподілені обчислення				
Вимоги до продуктивності				
Конфігурація з обмеженням				
Частота транзакцій				
Інтерактивний запит і/або запис				
Ефективність на рівні кінцевого користувача				
Інтерактивне оновлення				
Складна обробка				
Повторне використання				
Спрощення перетворення/установки				
Спрощення операції				
Використання не декількох вузлів				
Потенціал зміни функції				
Всього(N)				
Крок 4. Обчислення корегуючого множника складності (CAF) **CAF=0,65+(0,01*N)**				
Крок 5. Обчислення скорегованих функціональних точок (AFP) **AFP=FP*CAF**				
Крок 6. Перетворення рядка LOC (додатково) **LOC=AFP*LOC**				

Завдання 2.

Для розгляду представлено реалізацію програми «Заповнення масиву в шаховому порядку». Текст програми для реалізації можливого вирішення поставленої задачі, розроблено з використанням мови програмування C # (див. рис. 6.2).

Заповнити робочий лист згідно кроків 1)-5) (див табл. 6.7).

Номер рядку	*Рядки програми*
1	using System;
2	using System.IO;
3	class MyMetod
4	{
5	public static void form(int[] a,bool etalon)
6	{
7	Stream Reader sr=new StreamReader("in.txt");
8	int i,e0,el;
9	e0= int.Parse(sr.ReadLine());
10	e1= int.Parse(sr.ReadLine());
11	for (i = 0; i < a.Length; i++, etalon =!etalon)
12	a[i] = (etalon) ? el: e0;
13	}
14	public static void desk(int[ ][ ] d)
15	{
16	bool etalon = false;
17	int i,j;
18	for (i = 0; i < d.Length; i++, etaIon =!etaIon)
19	form(d[i], etalon);
20	}
21	public static void print(int[ ][ ] a)
22	{
23	int i,j;
24	for (i = 0; i < a.Length; i++, Console.WriteLine())
25	for (j = 0; j < a[]-Length; j++)
26	Console.Write(" {0,2}", a[i][j]);
27	}
28	public static void Main()
29	{
30	int [ ] [ ] d;
31	int n,m,i;

```
32      char rep;
33       do
34      {
35      Console.Write("Строк:");
36      n = int.Parse(Console.ReadLine());
37      Console.Write("Столбцов:");
38      m = int.Parse(Console.ReadLine());
39      d= new int[n][ ];
40      for (i = 0; i < d.Length; i++)
41      d[i] = new int[m];
42      desk(d);
43      print(d);
44      Console.Write("\nДля повтора нажмите клавишу Y: ");
45      rep = char.Parse(Console.ReadLine());
46      Console.WriteLine();
47      } while (rep = = 'Y' || rep = = 'y');
48      }
49      }
```

Рис. 6.2. Реалізація програми “Заповнення масиву в шаховому порядку” з використанням мови програмування C #.

Завдання 3.

Здійснити перетворення коду з мови програмування на мову Basic Assembler SLOC з розрахунку на одну функціональную точку.

Визначити кількість рядків програми на мові:

Варіант 1. Java, якщо відомо, що ця ж програма містить 500 LOC на мові C.

Варіант: 2. Delphi, якщо відомо, що ця ж програма містить 20 LOC на мові C++.

Варіант: 3. C++, якщо відомо, що ця ж програма містить 120 LOC на мові Java.

Варіант: 4. C, якщо відомо, що ця ж програма містить 135 LOC на мові C++.

Варіант: 5. Oracle, якщо відомо, що ця ж програма містить 90 LOC на мові Foxpro.

Варіант: 6. Access, якщо відомо, що ця ж програма містить 64 LOC на мові Oracle.

Варіант: 7. Pascal, якщо відомо, що ця ж програма містить 200 LOC на мові Delphi.

Варіант: 8. Java, якщо відомо, що ця ж програма містить 36 LOC на

мові C#.

Варіант: 9. Access, якщо відомо, що ця ж програма містить 345 LOC на мові DBaseIV.

Варіант: 10. Basic, якщо відомо, що ця ж програма містить 180 LOC на мові Pascal.

В таблиці показників для перетворення коду з однієї мови програмування на іншу мову програмування (див. Додаток Г) приведені показники SLOC для перетворення коду з мови програмування на мову Basic Assembler SLOC з розрахунку на одну функціональную точку.

Перший і другий стовпці таблиці представляють метод трансляції SLOC, застосовуваний в різних мовах по відношенню до середнього кількості рядків SLOC в базовій мові асемблера. (Потрібно звернути увагу, що SLOC і LOC є взаємозамінними.) Вважається за краще виконувати трансляцію з мов програмування на базову мову асемблера тому, що в цьому випадку операції порівняння у багатьох проектах можуть виконуватися ідентично. Ці дані можуть бути корисними також у разі трансляції проекту зі звичайної мови програмування на мову перетворення.

6.3. Контрольні запитання

1. Що розуміють під зовнішніми вводами, використовуючи метод функціональних точок?
2. Що розуміють під зовнішніми виводами, використовуючи метод функціональних точок?
3. Що розуміють під зовнішніми запитами, використовуючи метод функціональних точок?
4. Які файли відносять до локальних внутрішніх логічних файлів?
5. Які файли відносять до зовнішніх інтерфейсних файлів?
6. Як застосовуються вагові множники складності?
7. За якою шкалою оцінюються фактори середовища?
8. Скільки факторів середовища оцінюють?
9. Які саме передбачувані фактори середовища ви знаєте?
10. За якою формулою вираховується множник корегування складності?
11. Як здійснюється перетворення в рядки LOC?

РОЗДІЛ 7

МЕТОД ФУНКЦІОНАЛЬНИХ ТОЧОК ДЛЯ ПС З ГРАФІЧНИМ КОРИСТУВАЦЬКИМ ІНТЕРФЕЙСОМ

7.1. Теоретичний матеріал з теми розділу

Зазвичай функціональність даних представляється файлами, таблицями баз даних, об'єктами та іншими одиницями зберігання інформації. При аналізі за методом функціональних точок розглядаються два види груп даних:

Внутрішній логічний файл (ILF – Internal Logical File) – логічно пов'язана група даних, що визначається користувачем і знаходиться всередині меж ПС.

Зовнішній інтерфейсний файл (EIF – External Interface File) – логічно пов'язана група даних, що забезпечує ПС інформацією, але лежить за її межами і підтримується іншою ПС.

Транзакції – це елементарні процеси, тобто найменші одиниці активності, мають сенс для користувача, які відбуваються всередині ПС і які породжуються вхідною і вихідною інформацією. В аналізі, заснованому на методі функціональних точок, виділяють три види транзакцій:

зовнішній ввід (EI – External Input) – процес введення даних і керуючої інформації в ПС. Керуюча інформація необхідна для правильної обробки даних. Дані, що надходять на вхід ПС, використовуються для підтримки внутрішнього логічного файлу. Зазвичай, процеси виду EI використовуються для *додавання, зміни або видалення* інформації;

зовнішній вивід (EO – External Output) – процес, що генерує дані або керуючу інформацію, які надходять на вихід ПС. Зазвичай процес виду EO є формування різних *екранів, звітів, повідомлень*;

зовнішній запит (EQ – External Inquiry) – діалоговий ввід, який призводить до негайної відповіді ПС в формі діалогового виводу. При цьому діалоговий ввід в ПС не зберігається, а діалоговий вивід не вимагає виконання обчислень. У цьому полягає головна відмінність EQ від EI і EO.

Кожній з виявлених характеристик функціональності ПС (EI, EO, EQ, ILF, або EIF) ставиться у відповідність низький, середній або високий рівень складності, а потім присвоюється певна числова оцінка.

Для зовнішніх і внутрішніх файлів (ILF і EIF) складність визначається і ранжується за допомогою кількості типів елементів записів

(RET – Record Element Types) і кількості типів елементів даних (DET – Data Element Types), що входять до відповідних логічних груп даних. При цьому під кількістю RET розуміється кількість різних форматів записів, що використовуються в даному файлі, а під кількістю DET – кількість різних полів в записах.

Рівні складності для внутрішніх і зовнішніх файлів (ILF і EIF) в залежності від кількості RET і DET наведені в табл. 7.1.

Таблиця 7.1

Рівні складності для внутрішніх і зовнішніх файлів (ILF і EIF)

	1 - 19 DET	20 - 50 DET	51 і більше DET
1 RET	Низький (Low)	Низький (Low)	Середній (Average)
2 - 5 RET	Низький (Low)	Середній (Average)	Високий (High)
6 і більше RET	Середній (Average)	Високий (High)	Високий (High)

Для транзакційних функцій (EI, EO і EQ) складність визначається і ранжується за допомогою кількості типів використовуваних файлів (FTR – File Types Referenced), тобто кількості ILF і EIF, які беруть участь в транзакційному процесі, а також кількості типів елементів даних DET (відмінних один від одного полів записів), таких, що додаються, модифікуються, витираються або створюються в вихідних даних.

Рівні складності для зовнішніх входів (EI) в залежності від числа FTR і DET наведені в табл. 7.2.

Таблиця 7.2

Рівні складності для зовнішніх входів (EI)

	1 - 4 DET	5 - 15 DET	16 і більше DET
0 - 1 FTR	Низький (Low)	Низький (Low)	Середній (Average)
2 FTR	Низький (Low)	Середній (Average)	Високий (High)
3 і більше FTR	Середній (Average)	Високий (High)	Високий (High)

Рівні складності для зовнішніх виходів (EO) в залежності від числа FTR і DET наведені в табл. 7.3.

Таблиця 7.3

Рівні складності для зовнішніх виходів (EO)

	1 - 5 DET	6 - 19 DET	20 і більше DET
0 - 1 FTR	Низький (Low)	Низький (Low)	Середній (Average)
2 - 3 FTR	Низький (Low)	Середній (Average)	Високий (High)
4 і більше FTR	Середній (Average)	Високий (High)	Високий (High)

Рівні складності для зовнішніх запитів (EQ) визначаються з використанням наступного простого алгоритму:

1) вхід зовнішнього запиту розглядається аналогічно зовнішньому входу (EI) див. табл. 7.2;

2) вихід зовнішнього запиту розглядається аналогічно зовнішнього виходу (EO) див. табл. 7.3;

3) в якості результату використовується найбільше значення з отриманих в пп. 1) і 2) рівнів складності.

Вагові коефіцієнти рівнів складності для різних характеристик функціональності ПС наведені в табл. 7.4.

Таблиця 7.4

Вагові коефіцієнти рівнів складності

	НИЗЬКИЙ (LOW)	СЕРЕДНІЙ (AVERAGE)	ВИСОКИЙ (HIGH)
ILF	7	10	15
EIF	5	7	10
EI	3	4	6
EO	4	5	7
EQ	3	4	6

Таким чином, загальна формула для обчислення ненормованої кількості функціональних точок (UFPC – Unadjusted Function Point Count) наступна:

$$\begin{aligned} UFPC = {} & (3 * LEI) + (4 * AEI) + (6 * HEI) + (4 * LEO) \\ & + (5 * AEO) + (7 * HEO) + (7 * LILF) + (10 * AILF) \\ & + (15 * HILF) + (5 * LEIF) + (7 * AEIF) + \\ & (10 * HEIF) + (3 * LEQ) + (4 * AEQ) + (6 * HEQ) \end{aligned} \qquad (7.1)$$

де

UFPC – ненормована кількість функціональних точок;

LEI – кількість зовнішніх входів низького рівня складності;

AEI – кількість зовнішніх входів середнього рівня складності;

HEI – кількість зовнішніх входів високого рівня складності;

LEO – кількість зовнішніх виходів низького рівня складності;

AEO – кількість зовнішніх виходів середнього рівня складності;

HEO – кількість зовнішніх виходів високого рівня складності;

LILF – кількість внутрінніх логічних файлів низького рівня складності;

AILF – кількість внутрішніх логічних файлів середнього рівня складності;

HILF – кількість внутрішніх логічних файлів високого рівня складності;

LEIF – кількість зовнішніх інтерфейсних файлів низького уровня складності;

AEIF – кількість зовнішніх інтерфейсних файлів середнього рівня складності;

HEIF – кількість зовнішніх інтерфейсних файлів високого рівня складності;

LEQ – кількість зовнішніх запитів низького рівня складності;

AEQ – кількість зовнішніх запитів середнего рівня складності;

HEQ – кількість зовнішніх запитів високого рівня складності.

Для отримання остаточного результату аналізу, нормованої кількості функціональних точок (AFPC – Adjusted Function Point Count), необхідно також врахувати ряд загальних вимог до проекту, для чого отримане за формулою (1) ненормована кількість функціональних точок множиться на розрахований спеціальним чином нормуючий фактор (VAF – Value Adjustment Factor) .

У методі функціональних точок нормуючий фактор (VAF) визначається шляхом аналізу 14 основних характеристик системи (GSC – General System Characteristics), метою яких і є облік загальних вимог до проекту.

Кожна з характеристик системи оценюється експертним способом числом від 0 (якщо вона не присутня або не має значення для даного ПС) до 5 (якщо вона має дуже сильний вплив на дане ПС). Значення всіх 14

характеристик підсумовуються для отримання підсумкової ступеня впливу (TDI – Total Degree of Influence). Нормуючий фактор (VAF) розраховується за формулою

$$VAF=0.65+0.01*\text{TDI}. \qquad (7.2)$$

Таким чином, нормуючий фактор може приймати значення від 0,65 до 1,35, а нормована кількість функціональних точок представляє собою добуток ненормованої кількості функціональних точок на нормуючий фактор

$$AFPC=UFPC*VAF \qquad (7.3)$$

Надалі нормована кількість функціональних точок може бути використана для отримання оцінки кількості рядків вихідного коду (SLOC – Source Lines o Code) в ПС за допомогою Бекфай'р-методу або методу зворотного запуску (Backfire Method).

Бекфай'р-метод заснований на використанні так званого "мовного множника", який представляє собою середню кількість рядків вихідного коду конкретного алгоритмічної мови, що припадає на одну нормовану функціональну точку. Кейперс Джонс (Capers Jones) – експерт з метрик ПЗ, глава фірми Software Productivity Research, статистичними методами визначив значення "мовних множників" для основних мов програмування. Таким чином, якщо мова реалізації вибрана, то можна оцінити кількість рядків вихіднго коду розроблюваної ПС шляхом множення нормованої кількості функціональних точок на відповідний мовний множник

$$SLOC = AFPC \times \text{LM}, \qquad (7.4)$$

де LM – мовний множник мови програмування.

Приклад розрахунку за методом функціональних точок

Розглянемо приклад розрахунку кількості функціональних точок для ПС, що реалізує функції телефонного довідника. Припустимо, що в довіднику необхідно зберігати номер телефону, прізвище та ініціали власника, а також його адресу. Повинні бути передбачені можливості пошуку записів і сортування списку. Зовнішній вигляд єдиної екранної форми ПС наведено на рис. 7.1.

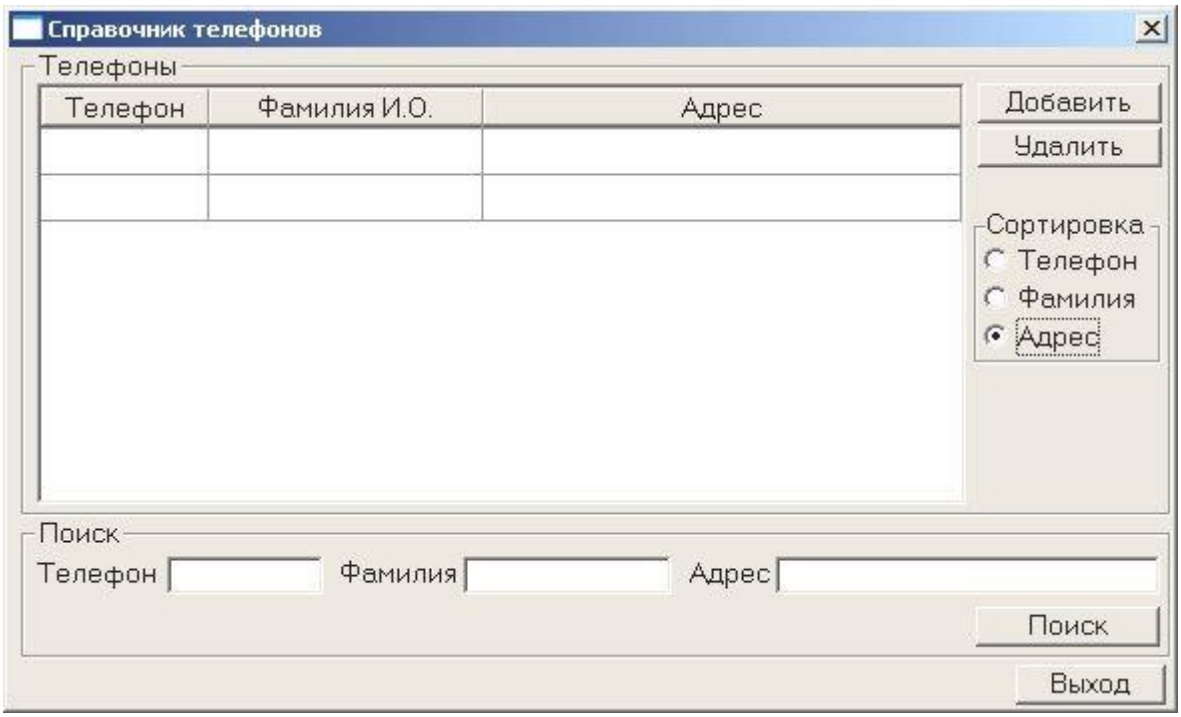


Рис.7. 1. Екранна форма телефонного довідника

Зробимо підрахунок кількості функціональних точок за п'ять кроків.

1. Встановлення меж даної ПС не викликає труднощів, так як вона є повністю локальною, і обмін даними з іншими ПС в ній не передбачається.

2. У ПС є один внутрішній логічний файл (ILF) для зберігання інформації довідника. Причому, дані можуть зберігатися як в звичайному файлі, так і в таблиці СУБД.

Число типів елементів записів (RET) для цього файлу може бути дорівнює одиниці, якщо дані у файлі зберігаються в вигляді однотипних записів: “Телефон”, “Прізвище” та “Адреса”, і нехай вони представлені в символьному форматі. У випадку, якщо номер телефону буде представлений, як ціле число, а прізвище та адреса в символьному форматі, то тоді внутрішній логічний файл буде мати два RET. Для визначеності будемо вважати, що внутрішній логічний файл має два RET.

Число типів елементів даних (DET) внутрішнього логічного файлу буде дорівнює трьом незалежно від формату представлення номера телефону (“Телефон”, “Прізвище”, “Адреса”). Таким чином, рівень складності внутрішнього логічного файлу низький (див. Табл. 7. 1).

Зовнішніх інтерфейсних файлів (EIF) дана ПС не має.

3. У ПС є два зовнішніх вводи (EI): “Додавання запису” і “Видалення запису”, оскільки саме ці дві функції ПС модифікують дані у внутрішньому логічному файлі. Так як зовнішній ввод «Додавання запису»

посилається на один внутрішній логічний файл і має п’ять елементів даних (поля “Телефон”, “Прізвище”, “Адреса”, кнопка “Добавити” і повідомлення, яке підтверджує факт додавання запису), то рівень складності цього введення низький (див. табл. 7.2). Аналогічно, рівень складності зовнішнього вводу “Видалення запису” також низький, оскільки є один FTR і п'ять DET (поля “Телефон”, “Прізвище”, “Адреса”, кнопка “видалити” і повідомлення, яке підтверджує факт видалення запису).

У програмі є два зовнішніх запити (EQ): “Вивід списку” від сортованих записів і “Пошук запису” в довіднику. Зовнішній запит “Вивід списку” має низький рівень складності, так як посилається на один внутрішній логічний файл і має чотири елементи даних (“Телефон”, “Прізвище”, “Адреса” і група радіо-кнопок “Сортування”). Рівень складності “Пошуку запису” в довіднику також низький (один внутрішній логічний файл і п'ять елементів даних: “Телефон”, “Прізвище”, “Адреса”, кнопка “Пошук”, повідомлення про відсутність шуканої інформації).

У ПС є також один зовнішній вивід (EO): вивід інформаційного повідомлення при спробі додати запис з існуючим номером телефону. Рівень складності цього зовнішнього вивводу - низький, так як він має один FTR і два DET: номер телефону і саме повідомлення.

Отримані дані зведемо в табл. 7.5. і розрахуємо ненормовану кількість функціональних точок UFPC за формулою (1).

Таблиця 7.5

Дані для розрахунку числа UFPC телефонного довідника

Характеристика	Рівень складності			Разом
	низький			
	Кількість	Ранг	Підсумок	
Зовнішні вводи (EI)	2	3	6	6
Зовнішні висновки (EO)	1	4	4	4
Зовнішні запити (EQ)	2	3	6	6
Внутрішні логічні файли (ILF)	1	7	7	7
Зовнішні інтерфейсні файли (EIF)	0	5	0	0
Разом (UFPC)				23

4. Підрахуємо тепер за допомогою табл. 5. і формули (7.2) підсумкову

ступінь впливу (TDI) загальних характеристик системи і нормуючий фактор (VAF).

В результаті з'ясуємо, що для телефонного довідника важливі наступні характеристики:

- «Діалоговий ввід даних» (п. 6), який оцінюється з вагою 5, оскільки всі 100% транзакцій в ПС є інтерак- нормативними;
- «Ефективність для кінцевого користувача» (п. 7), яка оцінюється з вагою 1, оскільки в ПС є функції автоматичного встановлення курсору, скролінгу і інтерфейс з мишею;
- «Простота використання» (п. 12), яка оцінюється з вагою 5, оскільки в ПС всі функції автоматизовані за винятком завантаження/вимикання і є автоматичне відновлення після помилок;
- «Поширеність» (п. 13), яка оцінюється з вагою 2, оскільки ПС розраховане на роботу на сумісному апаратному / програмному забезпеченні;
- «Легкість зміни» (п. 14), яка оцінюється з вагою 2, оскільки ПС зберігає інформацію в таблицях, підтримуваних користувачем в діалоговому режимі.

Інші характеристики або не присутні, або не мають значення для даного ПС і тому мають вагу рівний 0.

Нормуючий фактор (VAF) визначиться як

$$VAF = 0{,}65 + 0{,}1 \times \mathrm{TDI} = 0{,}65 + 0{,}1 \times (5 + 1 + 5 + 2 + 2) = 0{,}8$$

5. Таким чином, нормоване кількість функціональних точок для телефонного довідника обчислюється за формулою (3):

$$AFPC = \mathrm{UFPC} * VAF = 23 * 0{,}8 = 18{,}4$$

На закінчення, оцінимо кількість рядків вихідного коду з використанням Бекфайер-методу, виходячи з того, що програму необхідно розробити з використанням мови програмування С ++:

$$SLOC = 18{,}4 \times 53 = 975{,}2 \approx 975.$$

Таким чином, закінчена програма телефонного довідника буде містити приблизно 975 рядків вихідного коду на мові програмування С ++.

7.2. Практична робота № 7

Тема: Метод функціональних точок для ПС з графічним користувацьким інтерфейсом

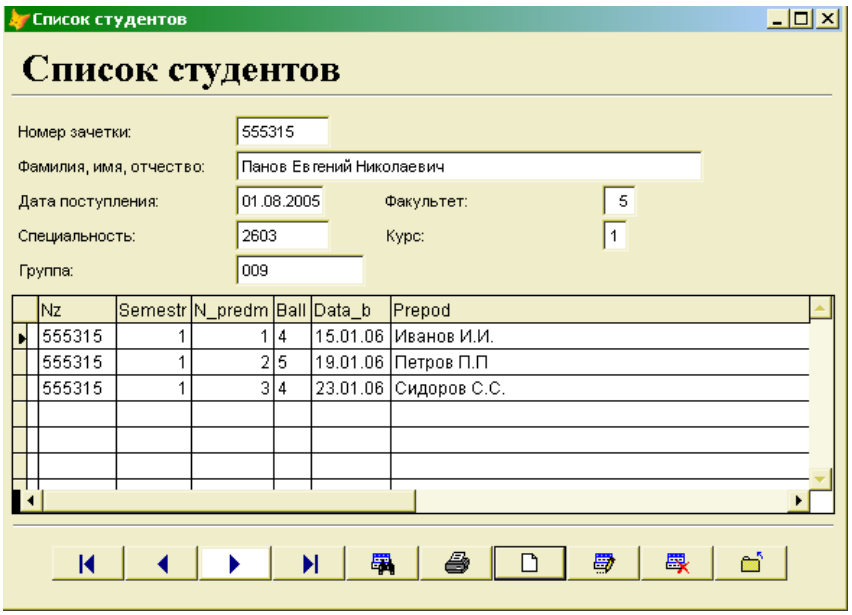
Мета роботи: Навчитися оцінювати характеристики програм, застосовуючи метод функціональних точок до ПС з графічним користувацьким інтерфейсом.

Завдання.

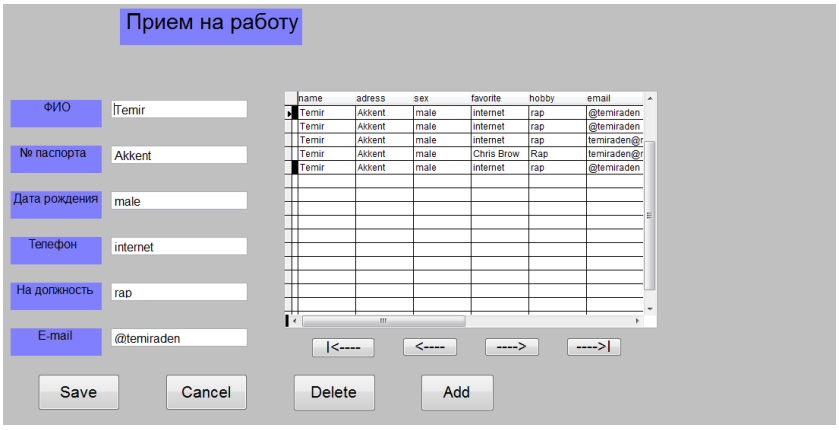
Оцінити ПС з графічним користувацьким інтерфейсом методом функціональних точок.

Варіант 1.

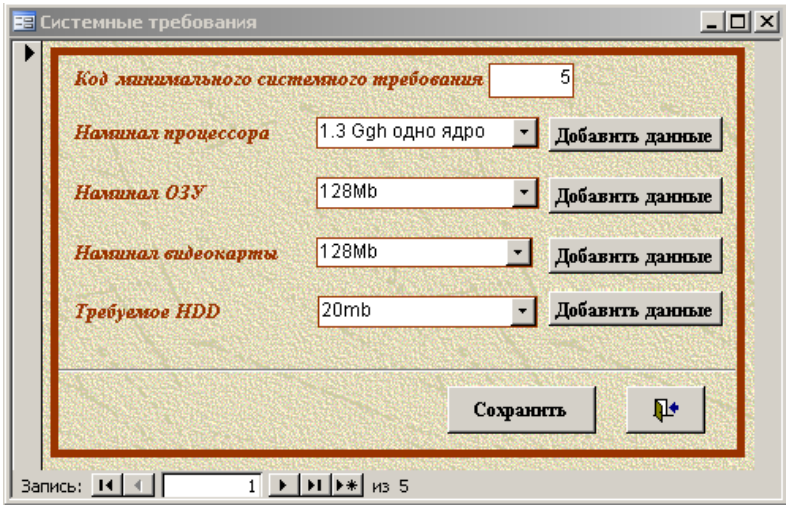
Варіант 2.



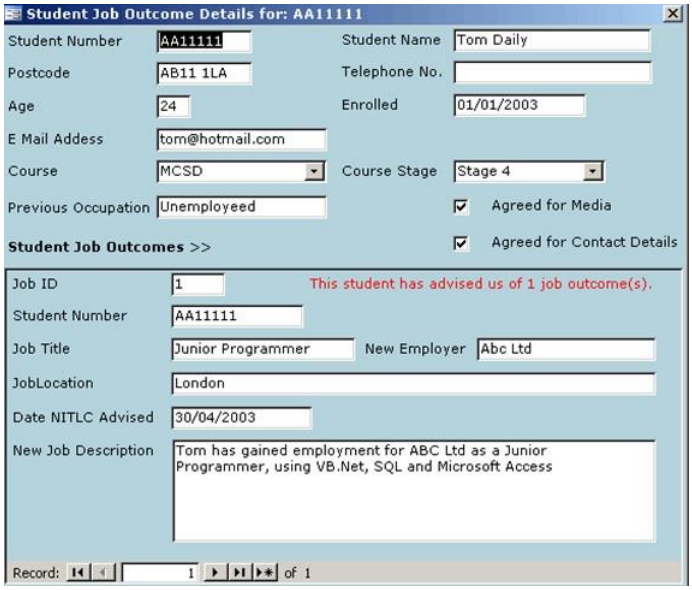
Варіант 3.



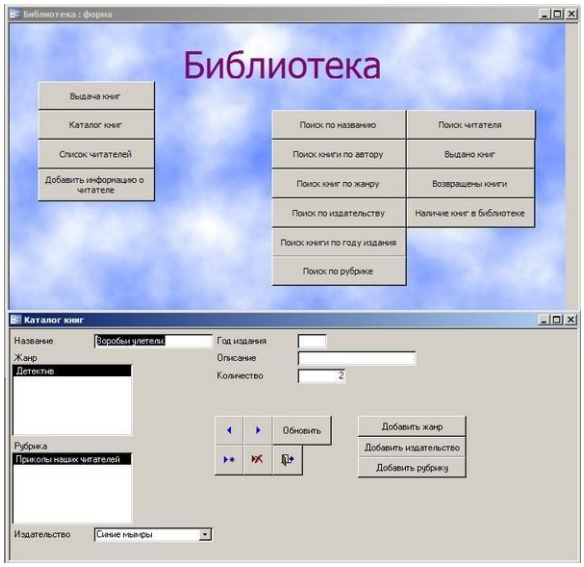
Варіант 4.



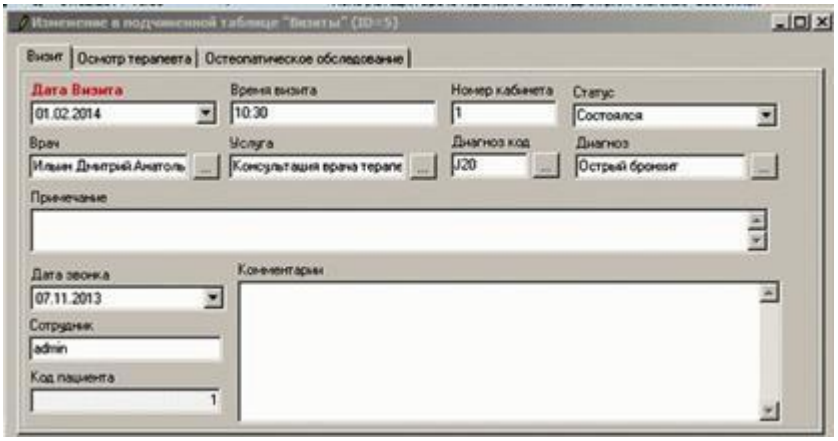
Варіант 5.



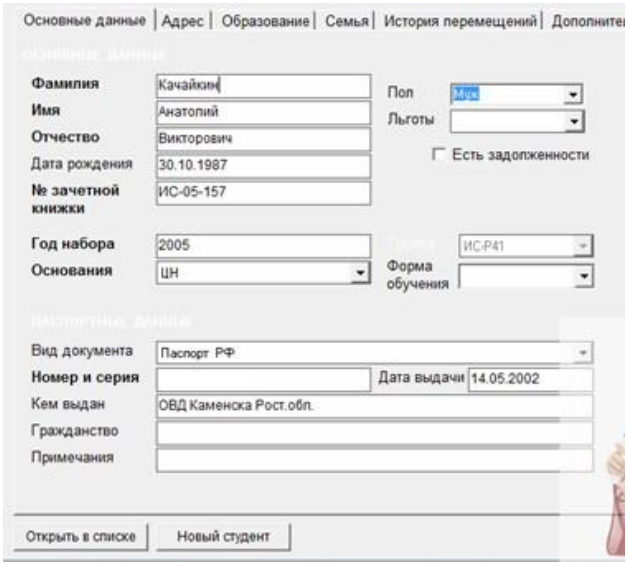
Варіант 6.



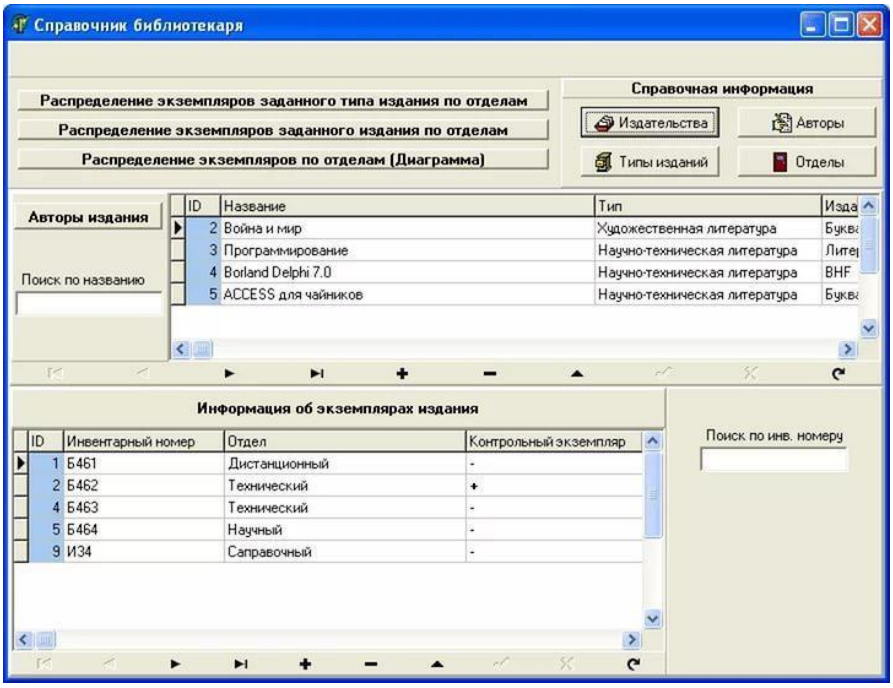
Варіант 7.



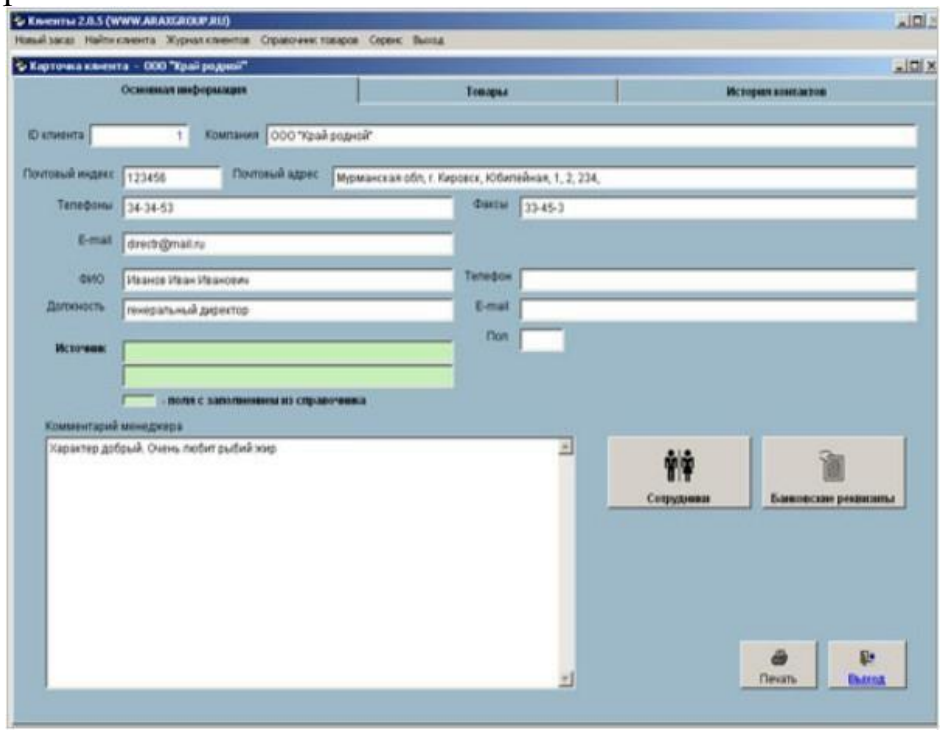
Варіант 8.



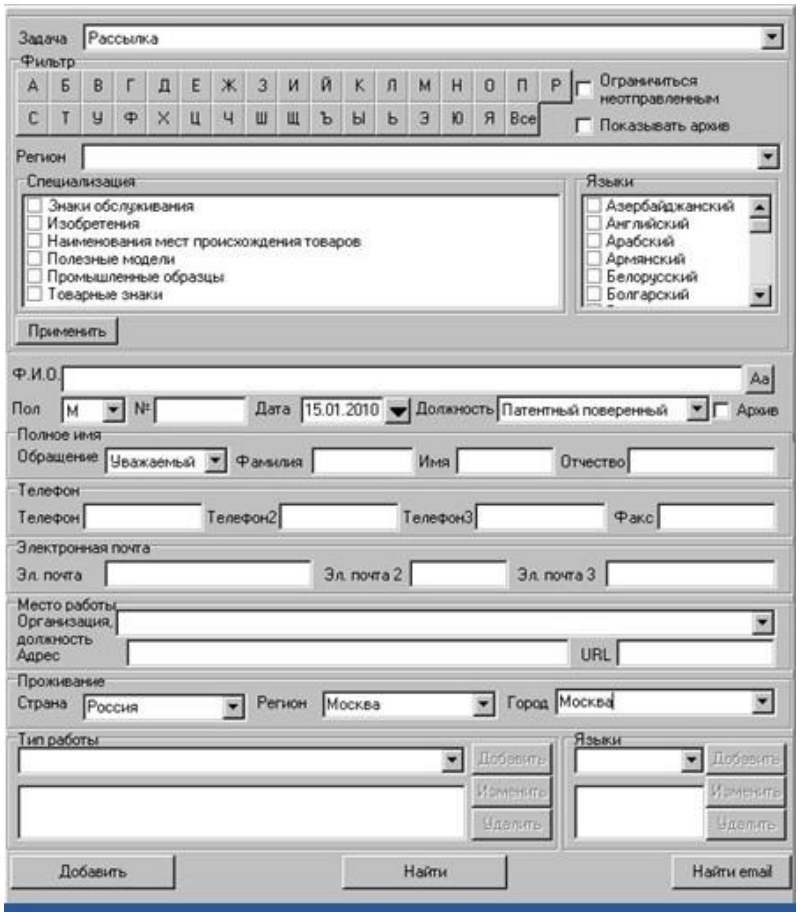
Варіант 9.



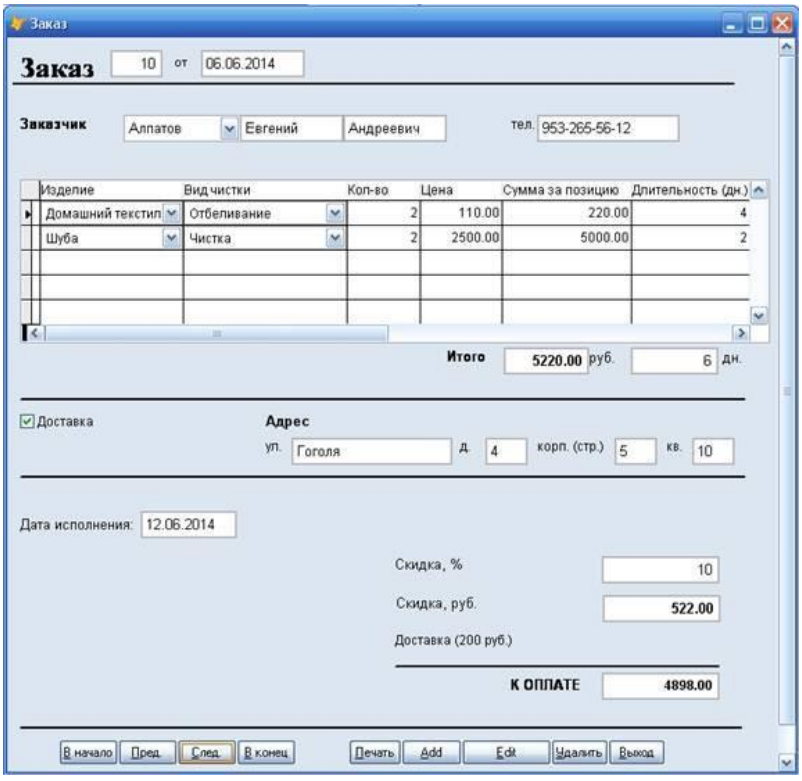
Варіант 10.



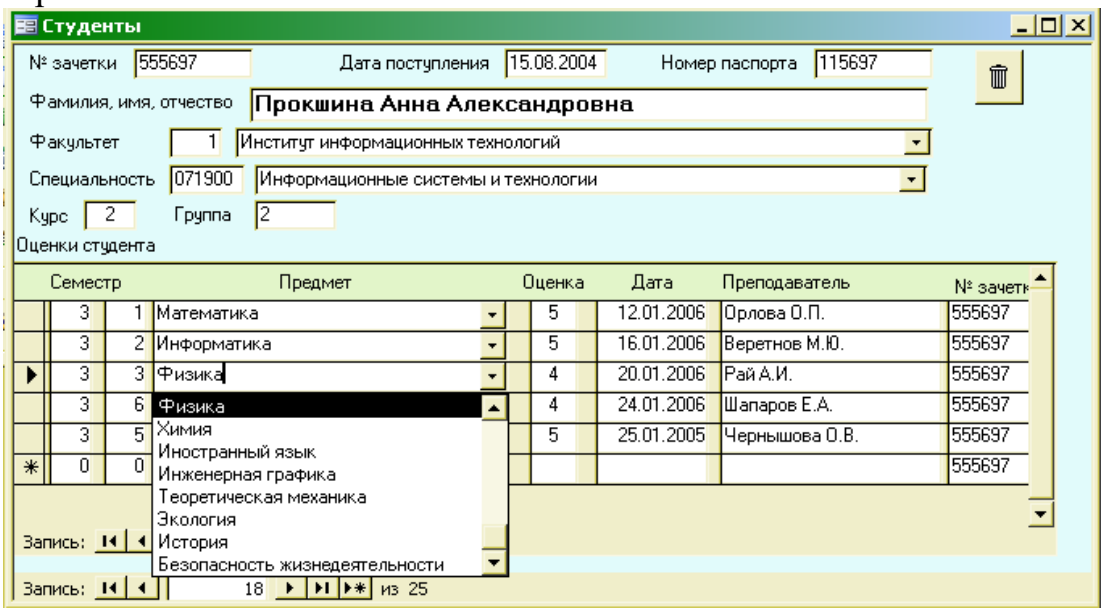
Варіант 11.



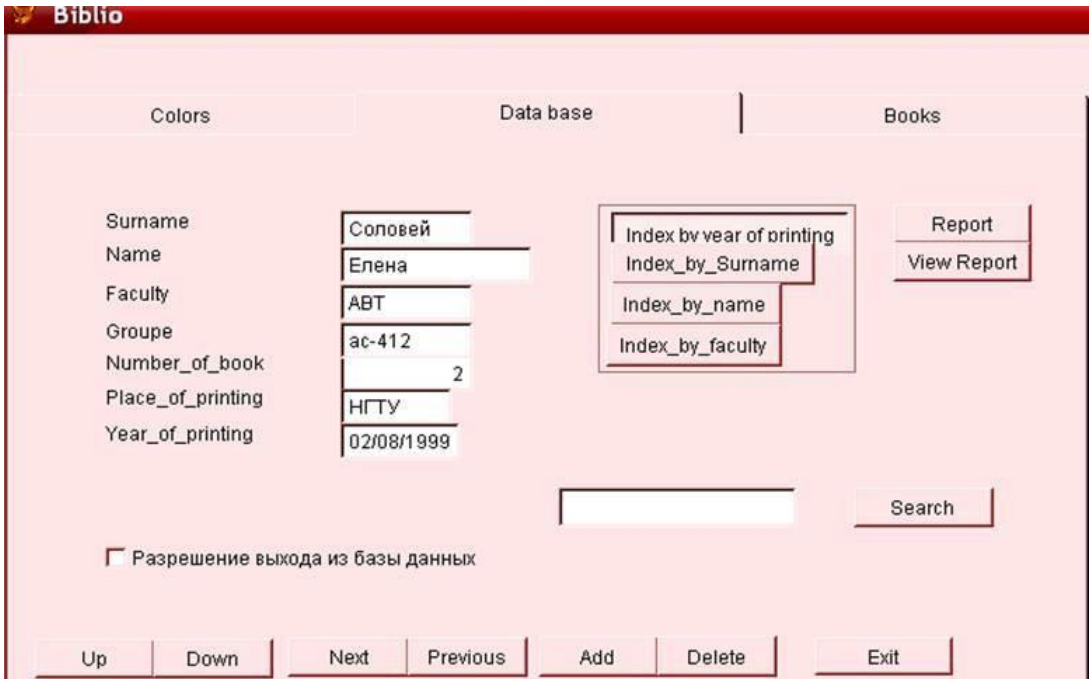
Варіант 12.



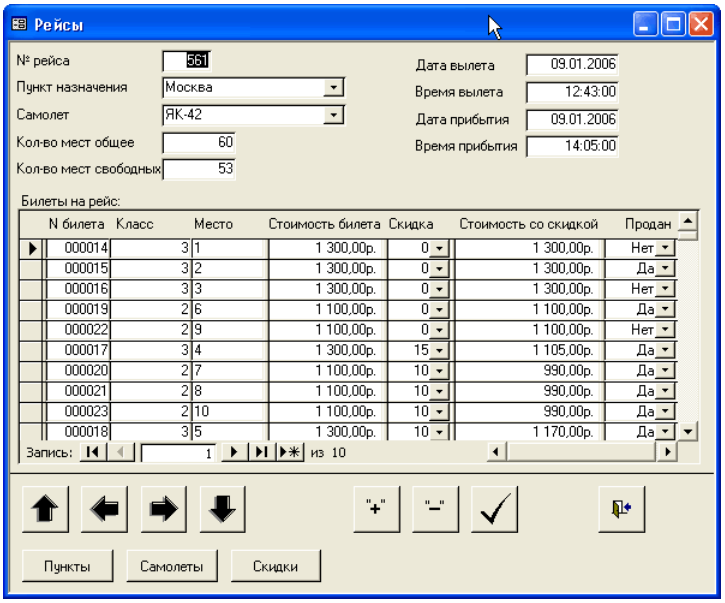
Варіант 13.



Варіант 14.



Варіант 15.



7.3. Контрольні запитання

1. Що таке логічно пов’язана група даних, що визначається користувачем і знаходиться всередині меж ПС?
2. Що таке логічно пов’язана група даних, що забезпечує ПС інформацією, але лежить за її межами і підтримується іншою ПС?
3. Що таке транзакція?
4. Що відображають такі характеристики функціональності як EI, EO, EQ, ILF та EIF?
5. Діалоговий ввід призводить до негайної відповіді ПС в формі діалогового виводу?
6. Чи вимагає діалоговий вивід виконання обчислень діалоговий ввід, який призводить до негайної відповіді ПС в формі діалогового виводу у зовнішньому запиті?
7. Що розуміють під кількістю типів елементів записів RET?
8. Що розуміють під кількістю типів елементів записів DET?
9. Чим визначається складність для транзакційних функцій EI, EO і EQ?
10. Як визначається ненормова кількість функціональних точок?
11. Як визначається нормова кількість функціональних точок?
12. Як визначається нормуючий фактор VAF у методі функціональних точок?

ЗАПИТАННЯ ДЛЯ ТЕСТОВОГО КОНТРОЛЮ ЗНАНЬ

У наступних запитаннях можливий один або кілька варіантів відповідей, а також пропонуються запитання на встановлення відповідності

1. В організаціях, які зайняті розробкою програмної продукції для кожного проекту реєструють наступні показники:

а) загальні трудовитрати
б) об’єм програми
в) вартість розробки
г) об’єм документації
д) помилки, які отримано напротязі року експлуатації
е) кількість людей, які працюють над продуктом
ж)строк розробки
з) кількість людей, які подали заявки для роботи над програмним продуктом
и) кількість вілльних годин від роботи над продуктом

2. Line of Code – це рядок вихідного коду ПЗ, у якому

а) виключаються порожні рядки
б) включаються порожні рядки
в) виключаються коментарі
г) включаються коментарі
д) включаються рядки з операторами
е) виключаються рядки з операторами

3. LOC залежить від мови програмування

а) Так
б) Ні
в) У виняткових випадках
г) Нема правильної відповіді

4. Вибрати правильні твердження при використанні LOC-оцінок

а) Чим більше рядків коду, тим вища продуктивність розробника
б) Продуктивність є оберенено пропорційною до кількості рядків коду
в) Одиниця розміру LOC не відображає функціональні властивості коду
г) Одиниця розміру LOC відображає функціональні властивості коду
д) Підрахунок LOC тим менш ефективний, чим більше коду створюється автоматично
е) Підрахунок LOC тим більш ефективний, чим більше коду створюється автоматично

5. Творець моделі COCOMO

а) Холстед
б) Мак-Кейб
в) Чепіт
г) Баррі Боем

6. Які атрибути вартості належать до характеристик продукту у проміжному рівні моделі COCOMO

а) Необхідна надійність ПЗ
б) Розмір БД додатка
в) Складність продукту
г) Обмеження швидкодії при виконанні програми
д) Обмеження пам’яті
е) Нестійкість оточення віртуальної машини
ж) Необхідний час відновлення
з) Аналітичні здібності
и) Досвід розробки
к) Здібності до розробки ПЗ
л) Досвід використання віртуальних маших
м) Досвід розробки на мовах програмування
н) Застосування методів розробки ПЗ
о) Використання інструментарію розробки ПЗ
п) Вимоги дотримання графіка розробки

7. Встановити відповідність між атрибутами вартості у проміжній моделі COCOMO та їх позначеннями?

1) Характеристики продукту
2) Характеристики програмного забезпечення
3) Характеристики персоналу
4) Характеристики проекту

а) MODP, TOOL, SCED
б) RELY, DATA, CPLX
в) ACAP, AEXP, PCAP, VEXP, LEXP
г) TIME, STOR, VIRT, TURN

8. Встановити відповідність між факторами масштабута їх описанням у моделі COCOMO II

1) Наявність досвіду аналогічних розробок
2) Гнучкість процесу розробки
3) Архітектура і дозвіл ризиків
4) Спрацьованість команди
5) Зрілість процесів

а) FLEX
б) PREC
в) TEAM
г) RESL
д) PMAT

9. Для попередної оцінки моделі COCOMO II вибрати множники трудомісткості, які визначають складність і надійність продукту та розробку для повторного використання

а) PERS
б) PREX
в) PCPX

г) RUSE
д) PDIF
е) FCIL
ж) SCED

10. Для попередної оцінки моделі COCOMO II вибрати множники трудомісткості, які визначають параметри проекту (обладанання і виконання графіка робіт)

а) PERS
б) PREX
в) PCPX
г) RUSE
д) PDIF
е) FCIL
ж) SCED

11. Одними з множників трудомісткості, які доповнені у детальній оцінці моделі COCOMO II є

а) багатоабонентська розробка
б) розмір бази даних
в) можливості програміста
г) можливості аналітика
д) можливість багаторазового використання
е) Зрілість процесів
ж) Гнучкість процесу розробки
з) Спрацьованість команди

12. Встановити відповідність між множинами змінних метрики Чепината їх описанням

1) Множина P
2) Множина M
3) Множина C
4) Множина T

а) Змінні, які приймають участь в управлінні роботою

програмного модуля
б) Змінні, які модифікуються або створюються всередині програми
в) Змінні, які не використовуються у програмі
г) Змінні для розрахунків та забезпечення виводу

13. Цикломатична складність графа програми є

а) цикломатичним числом Мак-Кейбом
б) в деяких випадках цикломатичним числом Мак-Кейба
в) числом компонентів зв’язності
г) в деяких випадках числом компонентів зв’язності

14. Цикломатичне число Мак-Кейба визначає

а) число лінійно незалежних контурів у сильнозв’язному графі
б) число лінійно незалежних контурів у графі програми
в) число компонентів зв’язності у графі програми
г) число розгалужень та циклів у коді програми

15. Число компонентів зв’язності графа є

а) кількістю дуг, які необхідно добавити, щоб перетворити граф у сильнозв’язний
б) кількістю дуг, які необхідно добавити, щоб дві будь-які вершини графа стали взаємно досяжні
в) кількістю дуг у графі
г) кількістю вершин у графі
д) кількістю дуг, які необхідно добавити, щоб перша і остання вершини графа стали взаємно досяжні
е) кількістю незалежних шляхів у базовій множині програми

16. Вибрати правильні твердження про незалежний шлях

а) Це шлях, що вводить новий оператор обробки
б) Це шлях, що вводить нову умову

в) він повинени містити дугу, що не входить у раніше визначені шляхи
г) починається у початковому вузлі і закінчується у кінцевому вузлі
д) незалежні шляхи формуються від найкоротшого до найдовшого
е) незалежні шляхи формуються від найдовшого до найкоротшого

17. У метриці Холстеда, де n1 – число унікальних операторів даної реалізації; n2 – число унікальних операндів даної реалізації; N1 – загальне число всіх операторів; N2 – загальне число всіх операндів, визначають словник програми за формулою:

а) N1+N2
б) n1+n2
в) n1+N1
г) n2+N2

18. Вибрати правильні твердження про спін:

а) ідентифікатор, що з'явився n раз у програмі, має спін, рівний n-1
б) ідентифікатор, що з'явився n раз у програмі, має спін, рівний n-2
в) ідентифікатор, що з'явився n раз у програмі, має спін, рівний n
г) При великому значенні спіна ускладнюється тестування і налагодження
д) При великому значенні спіна полегшується тестування і налагодження

19. Встановити відповідність між позначеннями і їх значеннями у методі функціональних точок

1) Процес введення даних і керуючої інформації в ПС
2) Процес, що генерує дані або керуючу інформацію, які надходять на вихід ПС

3) Діалоговий ввід, який призводить до негайної відповіді ПС в формі діалогового виводу
4) Логічно пов'язана група даних, що визначається користувачем і знаходиться всередині меж ПС
5) Логічно пов'язана група даних, що забезпечує ПС інформацією, але лежить за її межами і підтримується іншою ПС

а) EI
б) EQ
в) ILF
г) EO
д) EIF

20. Який метод оцінки вартості ПЗ оснований за принципом: “Клієнт завжди правий”?

а) Price-to-win
б) Оцінка по Паркінсону
в) Оцінка за аналогією
г) Експертна оцінка
д) SLIM
е) PERT

ДОДАТКИ

Додаток А

Лінійна програма

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
typedef struct _STUDENT
{
  char fio[20];
  char group[10];
  unsigned int grant;
}STUDENT, *PSTUDENT;

char path[] = "students.dat";
STUDENT st;

void main()
{

        char choice;
      fstream file;
      unsigned int num;

        do
        {
              clrscr();
          cout<<"1-add new record\n";
          cout<<"2-search record\n";
              switch(choice = getch())
              {
                    case '1':
                 {
                      file.open(path,ios::app);
                      if(!file)
                      {
                            cout<<"File could not be opened!!!\n";
                      }
                      else
                      {
                            cout<<"fio=";cin>>st.fio;
                            cout<<"group=";cin>>st.group;
```

```
                        cout<<"grant=";cin>>st.grant;
                        file.write((char*)&st,sizeof(STUDENT));
                        file.close();
                    }
                    getch();
                    break;
                }
                    case '2':
                {
                    file.open(path,ios::in);
                    if(!file)
                    {
                        cout<<"File could not be opened!!!\n";
                    }
                    else
                    {
                        cout<<"Enter number of record:";cin>>num;
                        file.seekp(num*sizeof(STUDENT));
                        file.read((char*)&st,sizeof(STUDENT));
                        file.close();
                        cout<<"fio="<<st.fio<<endl;
                        cout<<"group="<<st.group<<endl;
                        cout<<"grant="<<st.grant<<endl;
                    }
                    getch();
                    break;
                }
            }
        }
        while(choice != 27);
}
```

Додаток Б

Програма із вказівниками

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
typedef struct _STUDENT
{
  char fio[20];
  char group[10];
  unsigned int grant;
}STUDENT, *PSTUDENT;

char path[] = "students.dat";
PSTUDENT st;

void main()
{

        char choice;
      unsigned int num;
      fstream *file;

      st = new STUDENT;
      file = new fstream();
       do
       {
            clrscr();
          cout<<"1-add new record\n";
          cout<<"2-search record\n";
            switch(choice = getch())
            {
                  case '1':
                {
                   file->open(path,ios::app);
                   if(!file)
                   {
                        cout<<"File could not be opened!!!\n";
                   }
                   else
                   {
                        cout<<"fio=";cin>>st->fio;
                        cout<<"group=";cin>>st->group;
```

```
                        cout<<"grant=";cin>>st->grant;
                        file->write((char*)st,sizeof(STUDENT));
                        file->close();
                    }
                    getch();
                    break;
                }
                    case '2':
                {
                    file->open(path,ios::in);
                    if(!file)
                    {
                        cout<<"File could not be opened!!!\n";
                    }
                    else
                    {
                        cout<<"Enter number of record:";cin>>num;
                        file->seekp(num*sizeof(STUDENT));
                        file->read((char*)st,sizeof(STUDENT));
                        file->close();
                        cout<<"fio="<<st->fio<<endl;
                        cout<<"group="<<st->group<<endl;
                        cout<<"grant="<<st->grant<<endl;
                    }
                    getch();
                    break;
                }
            }
        }
        while(choice != 27);
    delete file;
    delete st;
}
```

Додаток В

Програма з модулями

```
#pragma argsused
#include "type.h"
#include <iostream.h>
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
char path[] = "students.dat";
STUDENT st;

void main()
{
        char choice;
       unsigned int num;

        do
        {
                clrscr();
                switch(choice=menu())
                {
                        case '1':
                   {
                        cout<<"fio=";cin>>st.fio;
                        cout<<"group=";cin>>st.group;
                        cout<<"grant=";cin>>st.grant;
                        add(path, st);
                        break;
                   }
                        case '2':
                   {
                        cout<<"Enter number of record:";cin>>num;
                        st = search(path,num);
                        print(st);
                        break;
                   }
                }
        }
        while(choice != 27);
}

//---------------------------------------------------------------------------
```

```
#include "Unit1.h"
//---------------------------------------------------------------------------
char menu()
{
  cout<<"1-add new record\n";
  cout<<"2-search record\n";
  return getch();
}
#pragma package(smart_init)

//---------------------------------------------------------------------------
#include "Unit2.h"
//---------------------------------------------------------------------------
void add(char *path, STUDENT st)
{
  fstream file;
  file.open(path,ios::app);
  if(!file)
  {
      cout<<"File could not be opened!!!\n";
  }
  else
  {
      file.write((char*)&st,sizeof(STUDENT));
      file.close();
  }
}
//---------------------------------------------------------------------------
#include "Unit3.h"
//---------------------------------------------------------------------------
STUDENT search(char *path, int num)
{
  fstream file;
  STUDENT st;
  file.open(path,ios::in);
  if(!file)
  {
      cout<<"File could not be opened!!!\n";
  }
  else
  {
      file.seekp(num*sizeof(STUDENT));
      file.read((char*)&st,sizeof(STUDENT));
      file.close();
  }
```

```
  return st;
}
//---------------------------------------------------------------------------
#include "Unit4.h"
//---------------------------------------------------------------------------
void print(STUDENT st)
{
  cout<<"fio="<<st.fio<<endl;
  cout<<"group="<<st.group<<endl;
  cout<<"grant="<<st.grant<<endl;
  getch();
}
```

Додаток Г

Показники для перетворення коду з однієї мови програмування на іншу мову програмування

Язык программирования	Basic Assembler SLOC (уровень)	Средний показатель SLOC из расчета на одну функциональную точку
Basic Assembler	1	320
Autocoder	1	320
Macro Assembler	1,5	213
C	2,5	**128**-150
Пакетные файлы DOS	2,5	128
Basic	3	107
Макросы LOTUS	3	107
ALGOL	3	105-106
COBOL	3	105-107
FORTRAN	3	105-106
JOVIAL	3	105-107
Смешанные языки программирования (по умолчанию)	3	105
Pascal	3,5	91
COBOL (ANSI 85)	3,5	91
RPG	4	80
MODULA-2	4,5	80
PL/1	4,5	80
Параллельный Pascal	4	80
FORTRAN 95	4,5	71
BASIC (ANSI)	5	64
FORTH	5	64
LISP	5	64
PROLOG	5	64
LOGO	5,5	58
Расширенный общий LISP	5,75	56
RPG III	5,75	56
C++	**6**	**53**
JAVA	6	53
YACC	6	53
Ada 95	6,5	49
CICS	7	46
SIMULA	7	46
Языки баз данных	8	40
CLIPPER DB и dBase III	8	40
INFORMIX	8	40
ORACLEи SYBASE	8	40
Access	8,5	38
Dbase IV	9	36
FileMaker Pro	9	36
Языки поддержки принятия решений	9	35
FOXPRO 2.5	9,5	34
APL	10	32
Статистические языки (SAS)	10	32
DELPHI	11	29
Стандартные объектно- ориентированные языки	11	29
OBJECTTVE-C	12	27
Oracle Developer/2000	14	23
SMALLTALK	15	21

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. COSMOS Technical Reference. The Software Cost Modeling System. Version 4.1. Revision E5. East Tennessee State University – Department of Computer and Information Sciences. 1998. – 71 p.
2. Longstreet D. Function Points Analysis. Training Course. 2002. – 109 p.
3. Б. Боэм Характеристики качества программного обеспечения. – М.: Мир, 1981.
4. Боем Б.У. Інженерне проектування програмного забезпечення. – М .: Радио и связь, 1985. – 512 с.
5. Гласс Р., Нуазо Р. Сопровождение программного обеспечения. – М.: Мир, 1983.
6. Дружественное программное обеспечение: сборник под редакцией Васильева Б.М. – М.: Знание, 1989. (Новое в жизни, науке, технике. Вычислительная техника и ее применение. 3/1989).
7. Жаркова Г. А. Программирование на языке С++: учебное пособие для вузов. – Ульяновск: УлГУ, 2009.
8. Жаркова Г. А. Современные системы автоматизации разработки информационных систем: учебно-методическое пособие. – Ульяновск: УлГУ, 2007.
9. Защита программного обеспечения: под редакцией Д. Гроувера. – М.: Мир, 1992.
10. Калбертсон Р., Браун К., Кобб Г. Быстрое тестирование. – М.: Вильямс, 2002.
11. Кинг Д. Создание эффективного программного обеспечения. – М.: Мир, 1991.
12. Леффингуэлл Д., Уилдриг Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. – М.: Вильямс, 2002.
13. Липаев В.В., Потапов А.И. Оценка затрат на разработку программных средств. – М.: Финансы и статистика, 1988.
14. Орлов С.А. Технологии разработки программного обеспечения: Разработка сложных программных систем: Учебное пособие. – СПб.: Питер , 2002.
15. Орлов С.А. Технології розробки програмного забезпечення. – СПб .: Пітер, 2004. – 528 с.
16. Соммервілла Іан Інженерія програмного забезпечення. – М., СПб., Київ: «Вільямс», 2002. – 625 с.
17. Тейер Т., Липов М., Нельсон Э. Надежность программного обеспечения: анализ крупномасштабных разработок. – М.: Мир, 1981.
18. Уилсон С.Ф., Мэйплс Б., Лэндгрейв Т. Принципы проектирования и разработки программного обеспечения: Учебный курс MCSD. - М.: Рус. Редакция, 2002.

19. Фокс Д. Программное обеспечение и его разработка. – М.: Мир, 1985.
20. Програма-калькулятор Costar 7.0 разроблена на основі моделі COCOMO II для автоматизації оцінки вартості розробки програмних продуктов. Доступ до ресурсу: http://www.softstarsystems.com/
21. Function Points Analysis Training Course. Доступ до ресурсу: http://www.softwaremetrics.com
22. International Function Point Users Group. Доступ до ресурсу: http://www.ifpug.org