Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Економіка ІТ-індустрії та підприємництво

Практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інженерія програмного забезпечення інформаційних систем»
спеціальності 121 Інженерія програмного забезпечення

Укладачі: М. О. Сидоров, П. Ю. Родіонов, О.І. Марченко

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2024

УДК 330.342.24
У39

Укладачі: Сидоров Микола Олександрович, проф. каф ІПІ, д.т.н., проф.
Родіонов Павло Юрійович, доц. каф. ІПІ, к.е.н.
Марченко Олена Іванівна, ст. викл. каф. ІПІ

Рецензенти Солдатова М.О., заст. декана ФІОТ з навчально-виховної роботи, к.т.н., доц.
Войтко С.В., зав. каф. міжнародної економіки, д.е.н., проф.

Відповідальний редактор Олійник Ю.О., доц. каф. ІПІ, к.т.н.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол No 6 від 28.03.2024 р.)
за поданням вченої ради Факультету інформатики та обчислювальної техніки
(протокол No 8 від 19.02.2024 р.)

Сидоров М.О., Родіонов П.Ю., Марченко О.І.

У39 Економіка ІТ-індустрії та підприємництво [Електронний ресурс] : практикум : навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Інженерія програмного забезпечення інформаційних систем» спец. 121 Інженерія програмного забезпечення / М.О.Сидоров, П.Ю.Родіонов, О.І.Марченко; КПІ ім. Ігоря Сікорського. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2024 . – 100 с.

Навчальний посібник містить теоретичний матеріал та практичні завдання, наведено приклади реалізації, що необхідні для виконання практичних з дисципліни «Економіка ІТ-індустрії та підприємництво».
Навчальний посібник призначений для здобувачів ступеня бакалавр за спеціальністю 121 Інженерія програмного забезпечення при вивченні дисциплін, пов’язаних з економікою програмного забезпечення.

УДК 330.342.24

Реєстр. № НП 23/24-326. Обсяг 2 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

ВСТУП

Навчальний посібник містить сукупність практичних робіт та завдань з навчальної дисципліни «Економіка ІТ-індустрії та підприємництво». Ці практичні роботи і завдання сприяють засвоєнню набутих знань, формуванню відповідних умінь і навичок.

У посібнику наведено систематизоване викладення основних відомостей щодо теоретичних засад та практичних аспектів розвитку економіки програмного забезпечення.

Враховуючи, значний обсяг загального матеріалу з дисципліни, практичну спрямованість на підготовку інженера з програмного забезпечення та приймаючи до уваги рекомендації SWEBOK, зміст дисципліни зосереджено на одному розділі запропонованому SWEBOK, а саме «Software Engineering Economics, Estimation Techniques».

Цей розділ спрямовано на оцінювання розміру програмного забезпечення, який застосовується для розрахунку, аналізу та прогнозування багатьох економічних параметрів будь якого програмного забезпечення. Наприклад, ресурси або час, необхідні для реалізації вимог, зусилля, графік виконання, оцінка витрат на управління, оцінка вартості програмного забезпечення та його супроводження, вартості технічного обслуговування, тощо.

Таким чином, набуті знання та навички будуть мати практичне спрямування, а бакалавр може їх використати при виконанні функціональних обов’язків інженера з програмного забезпечення; знати та вміти застосовувати: неалгоритмічні методи оцінки витрат на створення і супроводження програмного забезпечення; алгоритмічні методи оцінки витрат на створення і супроводження програмного забезпечення; користуватися відповідними ресурсами для розрахунків економічних параметрів щодо створення і супроводження програмного забезпечення.

Навчальний посібник призначений для студентів спеціальності 121 «Інженерія програмного забезпечення» кафедри інформатики та програмної інженерії всіх освітніх програм та всіх форм навчання.

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Посібник призначений для виконання практичних робіт з дисципліни «Економіка ІТ-індустрії та підприємництва». Навчальний посібник складається з семи практичних робіт. Кожна практична робота містить теоретичний матеріал, наведені приклади виконання завдань та контрольні питання. Посібник містить список рекомендованої літератури, що може бути корисним для поглибленого вивчення дисципліни.

Перед початком виконання практичної роботи студентам слід ознайомитися з теоретичною частиною роботи. Після цього кожен студент отримує індивідуальне завдання до виконання. Практичні роботи слід виконувати послідовно, оскільки роботи викладено у логічній послідовності відповідно до мети задачі курсу.

Загальні рекомендації до виконання практикуму:

- кожна практична робота виконується та оформлюється у вигляді окремого документу з назвою ГрупаАА_ПРВ_ВаріантСС_ ПрізвищеІніціали, де АА – номер групи, В – номер лабораторної роботи, СС – номер варіанту;
- кожна практична робота починається з титульного аркушу, на якому вказують тему практичної роботи, номер варіанту, прізвище та ініціали виконавця (приклад Додаток А);
- кожна практична робота повинна містити зміст;
- наводиться опис варіанту Завдання;
- якщо в роботі наводяться діаграми, графіки, малюнки, формули - обов’язкове обґрунтування використання та пояснення відношення до Завдання практичної роботи;
- оформленні звітів згідно стандартів оформлення документації;
- завантаження робіт та захист відбувається у терміни визначені викладачем та відповідним РСО дисципліни.

ПРАКТИЧНА РОБОТА №1

ДОСЛІДЖЕННЯ ЕКОНОМІЧНОГО СТАНУ ІНДУСТРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета: Ознайомитися з загальними показниками економічного стану індустрії програмного забезпечення і виконати дослідження (Systematic Literature Review) сучасного стану галузі.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Економіка програмного забезпечення (software economics) - це розділ економіки інженерії програмного забезпечення, що вивчає методи і засоби оцінки вартості програмного забезпечення, застосовуючи які інженери програмного забезпечення реалізують ефективні с точки зору економіки рішення щодо створення і супроводження програмного забезпечення. Економіка інженерії програмного забезпечення (software engineering economics) окрім методів, які враховують вартість, вивчає методи і засоби щодо ринків, вигоди, ризиків, альтернатив, невизначеності, неповних знань та цінності додаткової інформації, наслідки конкуренції, реклами, тощо. Подібно загальної економіки існує два розділи економіки програмного забезпечення:

Макроекономіка (Економіка інженерії програмного забезпечення) - вивчає методи і засоби, щодо прийняття рішень в національному і світовому масштабах, для одного чи декілька проєктів.

Наприклад:

1. Наскільки можливо «масштабувати» архітектуру програмного забезпечення, щоб спробувати захопити нові ринки?

2. Як широко можна охопити домен новою перспективною лінійкою продуктів з існуючій архітектурою та набором багаторазових компонентів?

3. Які ринкові сектори слід намагатися захопити чи покинути?

Мікроекономіка (Економіка програмного забезпечення) - вивчає методи і засоби, щодо прийняття рішень в місцевих масштабах, - окрема особистість або команда в контексті конкретного проєкту.

Наприклад:

1. Які технічні рішення треба обрати для ефективної з точки зору вартості архітектури програмного забезпечення: загальні структури (наприклад, шари, труби і фільтри, дошка); розподілені системи (наприклад, сервер клієнт, трирівнева, брокер); інтерактивні системи (наприклад, Model-View-Control, Presentation-Abstraction-Control)?

2. Як розвивати портфель інвестицій особистості чи команди: в продуктивність: в зрілість процесів; в інструменти; в модульність; в бази знань про програмне забезпечення?

3. Як поділити ролі в команді щоб досягти ефективності проєкту?

Економіка програмного забезпечення - це галузь прикладної мікроекономіки, яка вивчає, як ресурси використовуються для виробництва програмних систем та послуг. Основні етапи розвитку Економіки програмного забезпечення показано на рисунку 1.

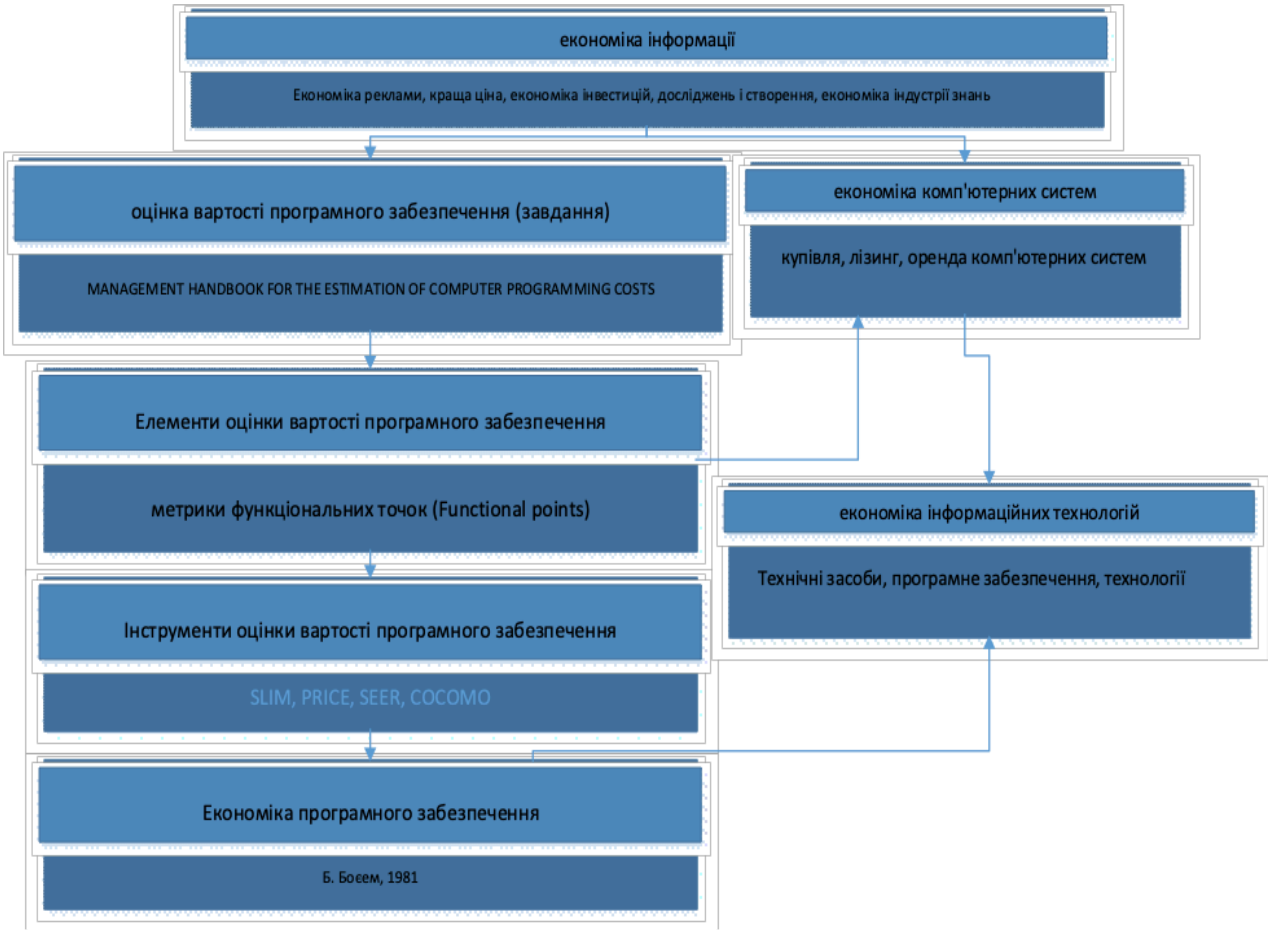


Рисунок 1 – Діаграма розвитку економіки програмного забезпечення

Витрати на програмні проєкти можна класифікувати на наступні категорії:

- постійної вартості;
- змінної вартості;
- вартості розробки;
- сервісної вартості;
- вартість конкурентоспроможності.

Етапи систематичного огляду мають три основні фази (рис. 2.):

- планування огляду;
- проведення огляду;
- звітування про огляд.

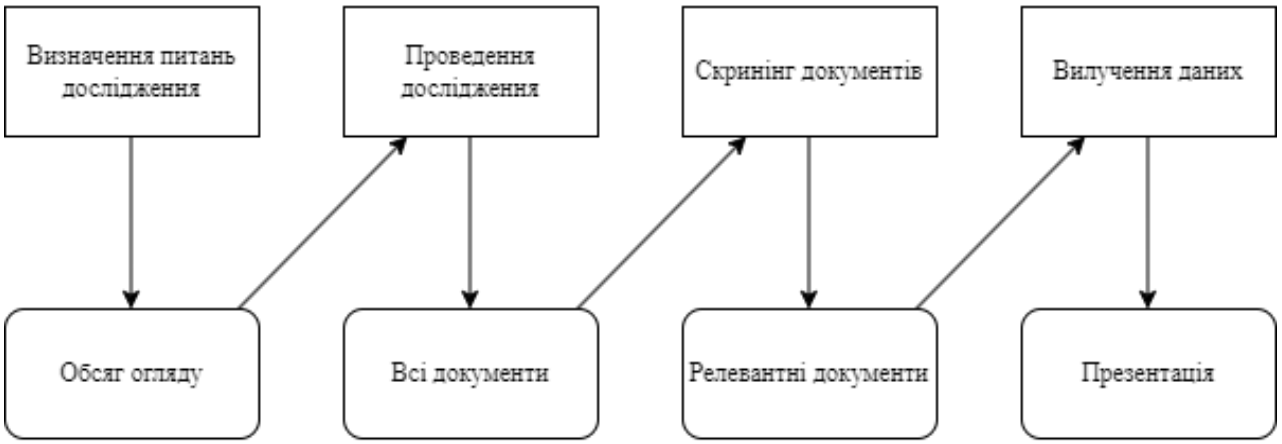


Рисунок 2 – Етапи проведення дослідження

Етапи, пов'язані з плануванням огляду:

- визначення потреби в огляді;
- специфікування питань дослідження;
- розробка протоколу огляду;
- оцінка протоколу огляду.

Етапи дослідження економічного стану

індустрії програмного забезпечення

Етапи, пов'язані з проведенням огляду:

- ідентифікація досліджень;
- підбір первинних досліджень;
- оцінка якості дослідження;
- вилучення та моніторинг даних;

- синтез даних.

Етапи, пов’язані зі звітуванням про огляд:

- визначення механізмів розповсюдження;

- форматування основного звіту;

- оцінка звіту.

Кожен етап процесу (прямокутник) має результат (овал), кінцевим результатом процесу є графічне представлення результатів дослідження.

Визначення дослідницьких питань (обсяг дослідження)

Щоб почати формулювання питань досліджень, необхідно бути знайомим з основою відповідного домену. Тому важливо формулювати питання і ключові слова у відповідності до особливостей домену дослідження, тоді можуть бути забезпечені релевантні дані з багатьох статей. В табл. 1. наведено питання досліджень, які були сформульовані для пошуку публікацій щодо стилів програмування і різонерів. Оскільки література по темі досліджень є англомовною, питання сформульовано англійською.

Цілі досліджень знайшли своє відображення в дослідницьких питаннях (RQ), як показано в таблиці 1.

Таблиця 1 – Дослідницькі питання

ДП 1	Що таке онтологія стилевого програмування чи стандартна онтологія кодування?
ДП 2	Що таке онтологія як частина статичного аналізу коду?
ДП 3	Що таке методи повторного використання онтології?
ДП 4	Що таке категоризація онтології чи інженерія онтології?
ДП 5	Які основні характеристики автоматизованих міркувань?
ДП 6	У чому суть логіки опису?

Пошук для первинних досліджень

Наступний крок – це пошук публікацій за темою дослідження. Первинні дослідження виконуються шляхом застосування пошукових стрічок для пошуку (див. таблицю 2).

Для ручного пошуку відокремлюємо терміни і створюємо пошукові стрічки. Можлива наступна стратегія:

- відокремити головні пошукові терміни;
- перевірити відповідні ключові слова на релевантність для відомих публікацій;
- розглянути альтернативні форми, такі як синоніми і релевантні ключові слова;
- застосовуючи булеві оператори OR або AND створити пошукові стрічки.

Таблиця 2 – Базові форми пошукових стрічок

("автоматизоване обґрунтування" OR "онтологія" OR "логіка опису")
((“онтологія” OR “таксономія”) AND (“стиль програмування” OR “стандарт кодування” OR “конвенція про кодування”))

Перегляд публікацій

Незважаючи на релевантність, пошукові стрічки дають забагато результатів (рис. 2), які потрібно переглядати вручну. Для цього необхідні критерії, у відповідності до яких публікації або будуть включені до подальшого дослідження, або ні.

Виділення даних і візуалізація результатів

Аналізуються дані за категоризаціями та областю досліджень. Для візуалізації результатів зазвичай застосовують діаграми і відповідні тексти.

ЗАВДАННЯ

1. Створити групу зі студентів, розподілити ролі в групі.
2. Обрати аспект дослідження за погодженням викладача.
3. Спираючись (але не обмежуючись) показниками економічного стану індустрії програмного забезпечення, які наведено в матеріалах до Лекції 1, здійснити їх аналіз.

4. Виконати дослідження, показників економічного стану індустрії програмного забезпечення, які обрані, шляхом аналізу відповідних первинних досліджень.

5. Оформити результати аналізів в формі презентації та звіту з обов'язковими посиланням на походження значень показників. В презентації та звіті відобразити наступне: обґрунтування щодо вибору аспекту дослідження та характерних показників індустрії програмного забезпечення; питання дослідження; результати пошуку; релевантність відокремлених первинних досліджень; зміни значень показників протягом часу.

6. Захистити результати виконаної роботи.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення економіки програмного забезпечення.

2. Що вивчає мікроекономіка як галузь економічної науки?

3. Що вивчає макроекономіка як галузь економічної науки?

4. Які види IT-компаній переважають на вітчизняному ринку програмного забезпечення?

5. Які країни світу є найбільшими за обсягом інвестицій в український сектор IT?

6. Якою є найбільша частка в структурі витрат IT компаній?

7. Охарактеризуйте розвиток економіки програмного забезпечення.

8. Поясніть можливості та призначення «Management Handbook for the Estimation of Computer Programming Costs».

9. Охарактеризуйте інструменти для оцінювання витрат на програмне забезпечення.

10. В який період були створені перші автоматизовані інструменти для оцінювання витрат на програмне забезпечення були побудовані?

ПРАКТИЧНА РОБОТА №2

МЕТРИКИ РОЗМІРУ. МЕТРИКА LINES OF CODE

Мета роботи: Ознайомитися з загальними поняттями щодо вимірювань та метрикою розміру Lines of Code. Напрацювати вміння застосування засобів вимірювання метрики. Отримати загальні вміння щодо застосування метрики в економіці програмного забезпечення.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Вимірювання це загальнонауковий, емпіричний метод пізнання, який визначається як процес, за допомогою якого числа або символи присвоюються атрибутам сутностей в реальному світі, таким чином, щоб описати їх чітко, відповідно до певних правил.

Сутність може бути об'єктом, наприклад, людина, або специфікація програмного забезпечення, або подія, наприклад, тестування програмного проєкту.

Атрибут є властивість сутності, така як довжина або функціональність (в специфікації), вартість, або тривалість (етапу тестування).

Величина це іменований атрибут сутності. Атрибути іменують тому, що сутність звичайно має декілька атрибутів (властивостей).

Міра - кількісна одиниця для виміру деякого атрибута продукту або процесу, наприклад, лексема або стрічка для довжини програми, функціональна крапка для розміру, людина/місяць для продуктивності процесу.

Метрика – кількісний вираз ступеня, в якій система, компонент або процес має відповідний атрибут.

Наприклад, довжина програми в лексемах. Таким чином, довжина програми це властивість, лексема – міра, довжина програми в лексемах - метрика.

Метрика ПЗ - чітка міра, що дозволяє оцінити певні властивості конкретного фрагменту програмного коду. Для кожної метрики зазвичай існують її еталонні показники, що вказують, при яких граничних значеннях варто звернути увагу на дану ділянку коду. Метрики коду розділяються на

категорії та можуть оцінювати досконало різні аспекти програмної системи: складність та структурованість програмного коду, зв'язковість компонентів, відносний обсяг програмних компонентів та ін. Найбільш проста для розуміння метрика - кількість рядків коду в програмній системі в сукупності з іншими метриками може слугувати для отримання формалізованих даних для оцінки коду. Наприклад, можна побудувати співвідношення між кількістю коду класу та кількістю методів/власності в класі, отримавши характеристику, показуючи, наскільки методи оцінки класу є обсягами. Крім того, такі оцінки можна використовувати в сукупності з метриками складності (наприклад, цикломатичної складності Мак-Кейба) для визначення найбільш складних частин у програмному коді та прийняття відповідних рішень.

Існують два основних типи вимірювань і метрик: прямі і непрямі.

Прямі вимірювання (метрики) є вимірювання (метрики), яке не залежить від вимірювання будь-якого іншого атрибута. Наприклад, кількість операндів, розмір документації (в сторінках).

Непрямі вимірювання (метрики) є вимірювання (метрики), яке включає вимір одного або більше інших атрибутів. Наприклад, рівень коментарів в програмі F = Ncom / Nloc.

Шкала вимірювань, це асоційована с атрибутом множина значень.

Метрика Lines of Code (LOC), це пряма, розмір-орієнтована метрика продукту (тексту програмного забезпечення), яка визначає розмір продукту в кількості логічних або фізичних строк коду.

Міра LOC вперше була введена приблизно в 1960 році і використовувалася для оцінки економічності (вартості), продуктивності та якості.:

1. Економічність програм вимірювалася за допомогою "долар на LOC".

2. Продуктивність вимірювали у вигляді «кількість рядків коду за одиницю часу".

3. Якість вимірювали через «кількість дефектів на KLOC", де "K" був символом для 1000 рядків програмного коду.

Міра LOC була досить ефективною для дослідження всіх трьох типів властивостей процесу створення програм.

Коли вперше була введена міра LOC, була лише одна широко використовувана мова програмування - асемблер. Програми були невеликими, а зусилля кодування складало близько 90% від усієї роботи. Фізичні та логічні висловлювання для процесора та асемблера, це практично одне и теж. Але, потім почали з'являтися проблеми (рис. 3).

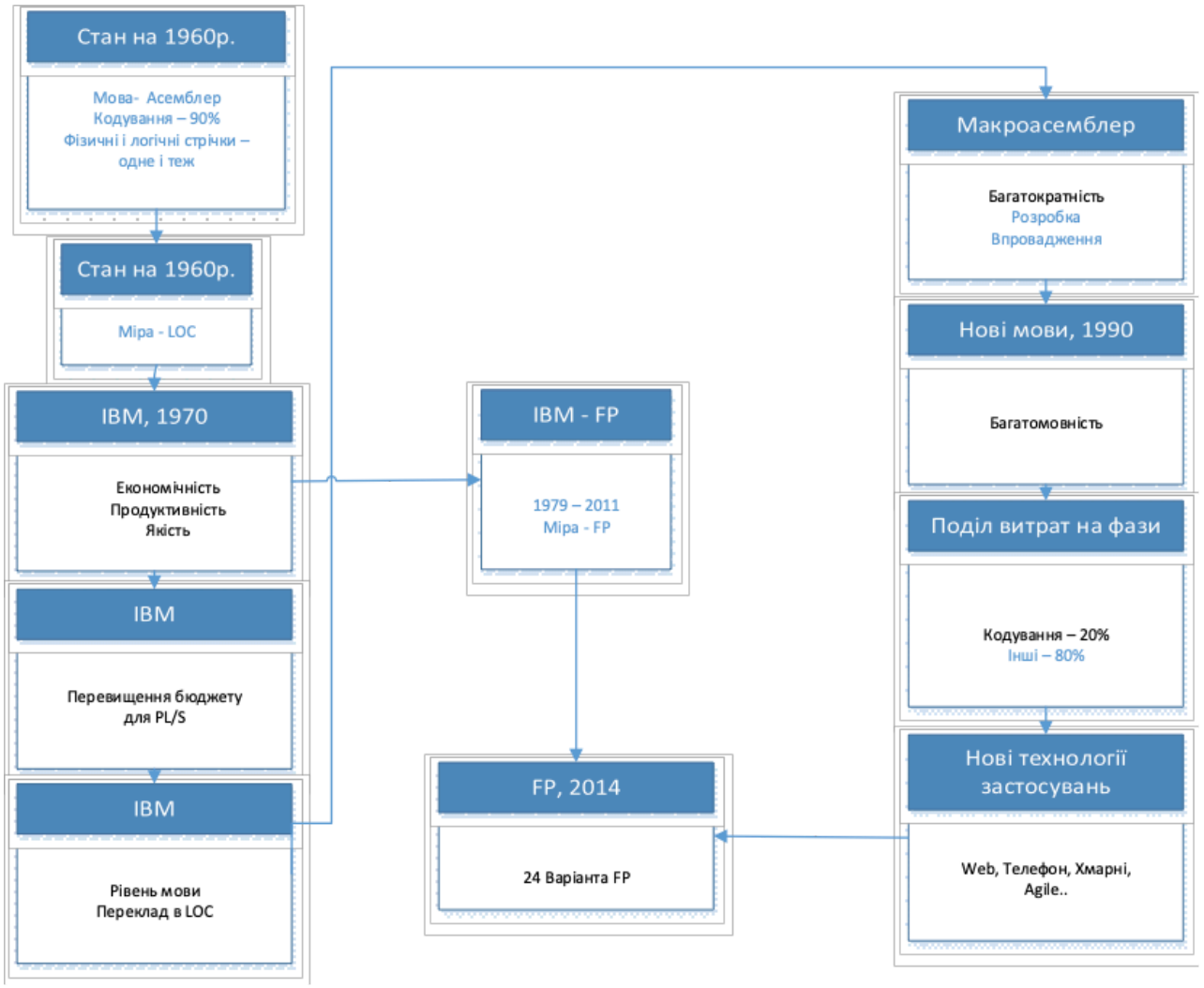


Рисунок 3 – Розвиток застосування міри LOC

Нарешті дійшли до наступного висновку: постійне використання міри LOC є суттєвим бар'єром, який затримує прогрес «інженерії програмного забезпечення» від ремесла до справжньої інженерної дисципліни.

Методика розв'язання типових задач

Зусилля (людина/місяць) Effort = aa * size^{bb},

де aa, bb коефіцієнти,

size – розмір продукту в KLOC.

Вартість (грн.) Cost = Effort * salary,

де salary – заробітна плата.

Час на розробку Schedule = cb * $Effort^{db}$,

де cb, db –коефіцієнти.

Типи проєктів

Тип «Органічний» являє собою відносно невеликий (до 25 тис. рядків коду) та простий проєкт, який виконується невеликою командою.

Тип «Напівроздільний» являє собою середній за розміром (до 75 тис. рядків коду) та складністю проєкт, в якому команда має змішаний рівень досвіду і відносно жорсткі вимоги.

Тип «Вбудований» являє собою проєкт, який виконується в умовах жорстких технічних, програмних та експлуатаційних обмежень (таблиця 3).

Таблиця 3 – Типи проєктів

Тип проєкту	ab	bb	cb	db
Органічний	2.4	1.05	2.5	0.38
Напівроздільний	3.0	1.12	2.5	0.35
Вбудований	3.6	1.20	2.5	0.32

ЗАВДАННЯ

1. Застосовуючи вимірювачі у відповідних середовищах програмування (Visual Studio, Code Counter for Java, CodeCounter, та інші), на прикладі власних програмних текстів виконати вимірювання розміру.

2. Здійснити відповідні економічні розрахунки.

3. Дослідити рівні мов програмування C# та Java.

4. Захистити виконану роботу.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення поняття типи метрик за типами складових життєвого циклу.

2. Охарактеризуйте типи метрик розміру продукту.

3. В чому полягає сутність поняття вимірювання?

4. Поясніть поняття властивість, атрибут, сутність та величина.

5. Дайте визначення економічності (вартості), продуктивності та якості.

6. Які існують типи вимірювань та метрик?

7. Охарактеризуйте поняття шкали вимірювань та наведіть приклади типів шкал.

8. Сутність та історичні аспекти розвитку метрики LOC.

9. Які існують переваги та недоліки метрики LOC?

10. Дайте визначення поняття рівня мови програмування та поясніть його зв’язок з метрикою Lines of Code.

ПРАКТИЧНА РОБОТА №3
АНАЛІЗ ФУНКЦІОНАЛЬНИХ ТОЧОК

Мета: Навчитися оцінювати економічні характеристики програмних продуктів на основі аналізу функціональних точок.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Метод функціональних точок (рис. 4) вимірює розмір програмного забезпечення засновуючись на його функціональності, яка специфікується в вимогах і відповідає потребам користувача. Тому, функціональність ідентифікується з точки зору користувача, а саме, визначаються елементарні, ідентифіковані користувачем в специфікації вимог функції та процеси, вони зважуються за деякими факторами складності, все підсумовується, щоб отримати остаточне значення розміру.

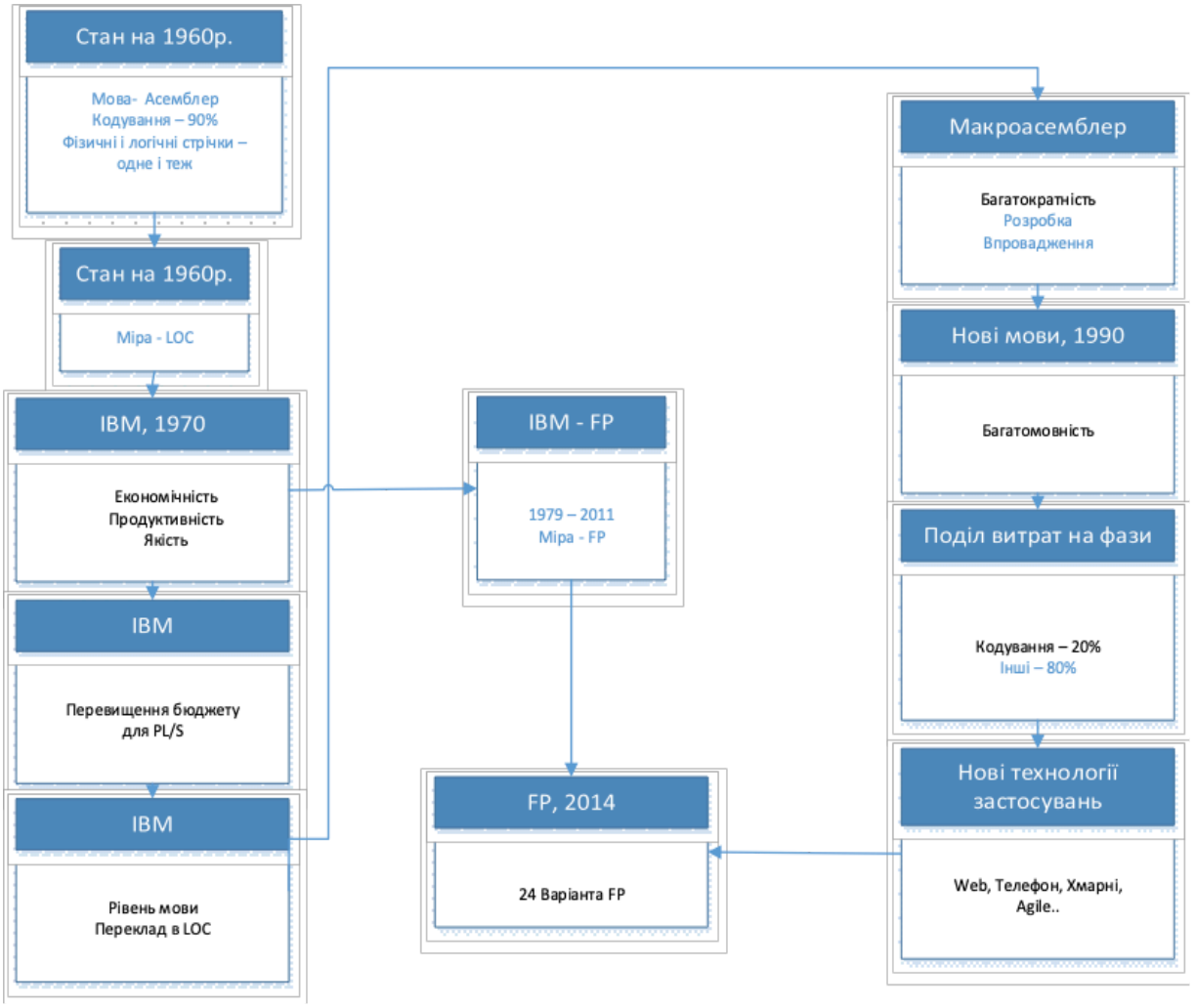


Рисунок 4 – Виникнення функціональних точок

Тому, що вибір процесів базується на уявленні користувачів, результат не залежить від багатьох технічних деталей розробки, таких, наприклад, як апаратне забезпечення, система керування базою даних, стиль програмування або мова програмування.

Функціональні точки - це міра для оцінювання розміру програмного забезпечення, подібно до того, як година міра для вимірювання часу, милі - відстані або Цельсій - температури. Функціональні точки - це інтервальні міри, подібні до інших мір, таких як кілометри, Фаренгейт, години тощо.

Функціональні точки підраховуються, засновуючись на результаті виконання особливого типу логічного проєктування, який називається аналіз функціональних точок (Function Points Analysis - FPA).

Процедура підрахунку функціональних точок в цілому складається з двох етапів:

1. Аналіз функціональних точок - Function Points Analysis.

2. Підрахунок кількості функціональних точок.

На першому етапі виконується наступне:

- з'ясовується тип підрахунку;

- визначається межа програмного забезпечення;

- ідентифікуються типи процесів і функцій.

На другому етапі виконується наступне:

- підраховуються, зважуються та підсумовуються функції і процеси, отримується значення невідрегульованих функціональних точок;

- значення коригуються деякими коефіцієнтами коригування значень, і отримується остаточна загальна кількість функціональних точок.

Організації можуть застосовувати аналіз функціональних точок як інструмент :

- для визначення розміру придбаного програмного продукту , шляхом підрахунку розміру всіх функцій програм, що входять до нього (застосовуючи для супроводження);

- для користувача, щоб визначити переваги програмного продукту для їхніх організацій, шляхом підрахунку функцій, які конкретно відповідають їхнім

вимогам (підраховуючи за що сплачено, чи за що сплатити);

- для вимірювання функціональності компонентів програмного продукту з метою підтримки процесів аналізу якості та продуктивності створення і супроводження;

- для оцінки вартості та витрат, необхідних для розробки програмного забезпечення та супроводження.

Функціональна точка є нормалізованим фактором для порівняння програмних продуктів.

Основні положення методу функціональних точок

Метод функціональних точок був вперше запропонований в 1983 р. Аланом Альбрехтом (Allan Albrecht) і на даний момент є основним для оцінки функціонального розміру програмних продуктів, як вже готових, так і таких, що знаходяться на стадії проєктування або супроводження.

Функціональні точки є мірою функціональності програмних продуктів з позиції користувача. Функціональність визначається і вимірюється шляхом виконання наступного:

- аналізу логічних груп даних, які використовуються і підтримуються програмним продуктом і характеризують, по суті, функціональність даних;

- аналізу функціональності введеної і виведеної користувачем інформації, тобто функціональності здійснюваних транзакцій.

Таким чином, загальна функціональність складається з двох складових (рис. 5).

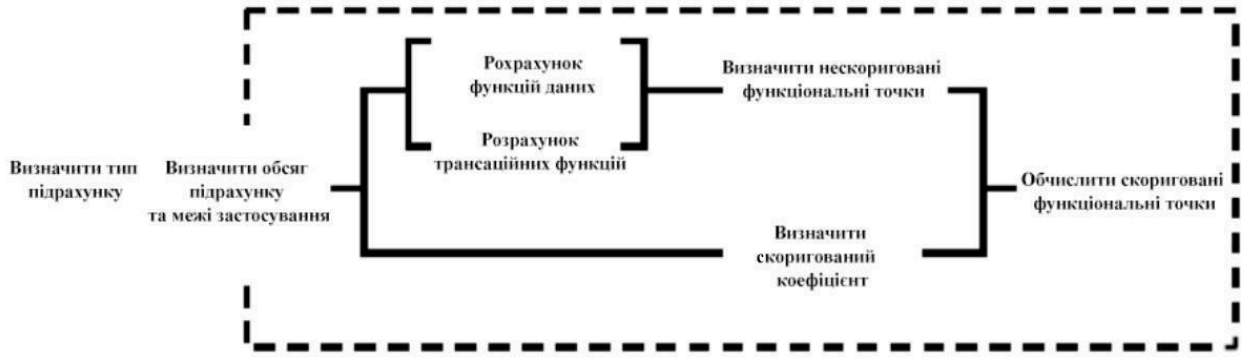


Рисунок 5 – Схема аналізу функціональних точок

Зазвичай функціональність представляється специфікаціями вимог користувача.

Підрахунок функціональних точок може бути пов'язаний або з проєктами, або з програмними продуктами. Тому, існує три типи підрахунку функціональних точок:

- проєкту створення програмного продукту,
- проєкту удосконалення програмного продукту,
- програмного продукту.

На рисунку 6 показано два типи функцій: функції даних (Data Functions) та функції транзакцій (Transaction Functions).

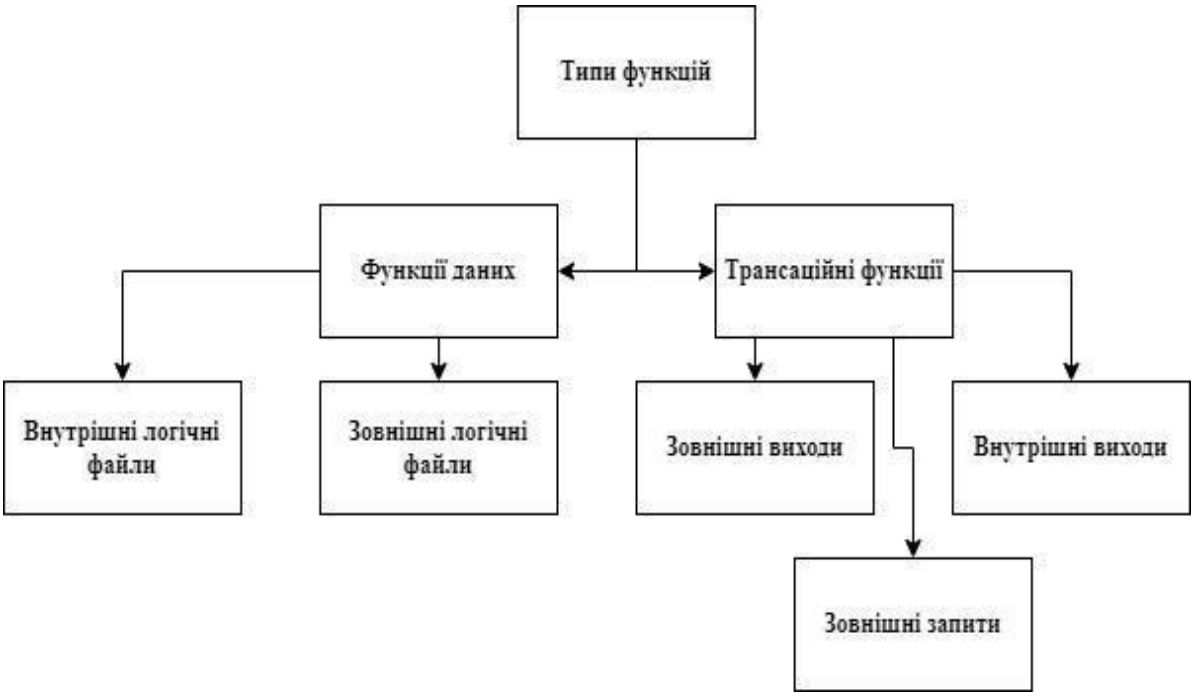


Рисунок 6 – Функції даних та транзакційні функції

Послідовність виконання при застосуванні аналіза методом функціональних точок:

1. Визначення типу оцінки.
2. Визначення області оцінки і масштабу проєкту.
3. Підрахунок функціональних точок, пов'язаних з даними.
4. Підрахунок функціональних точок, пов'язаних з транзакціями.
5. Визначення сумарного кількості не вирівняних функціональних точок (UFP).
6. Визначення коригуючого фактора (VAF).

7. Розрахунок кількості вирівняних функціональних точок (AFP).

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 кордонів ПЗ;

- зовнішній інтерфейсний файл (EIF - External Interface File) - логічно пов'язана група даних, або блоки керуючої інформації, на які посилається продукт, але які підтримуються поза продукту.

Рівні складності для внутрішніх і зовнішніх файлів (ILF і EIF) в залежності від кількості RET і DET наведені в таблиці 4.

Таблиця 4 – Рівні складності для внутрішніх і зовнішніх файлів (ILF і EIF)

Типи елементів запису	Елементи даних		
	1-19	20-50	>51
1 RET	Низький (7)	Низький (7)	Середній (10)
2 до 5 RET	Низький (7)	Середній (10)	Високий (15)
6 або більше RET	Середній (10)	Високий (15)	Високий (15)

Транзакції - це елементарні процеси, тобто найменші одиниці активності, мають сенс для користувача, вони відбуваються всередині ПЗ і породжують вхідну і вихідну інформацію. В аналізі, заснованому на методі функціональних точок, виділяють три види транзакцій:

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

- зовнішнє виведення (EO - External Output) - процес, що генерує дані або керуючу інформацію, які надходять на вихід ПЗ. Зазвичай процес виду EO є формування різних екранів, звітів, повідомлень;

- зовнішній запит (EQ - External Inquiry) - діалогове введення, яке призводить до негайної відповіді ПЗ в формі діалогового виведення. При цьому діалогове введення в самому ПЗ не зберігається, а діалоговий висновок не вимагає виконання обчислень. У цьому полягає головна відмінність EQ від EI і EO.

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, що беруть участь в транзакційному процесі, а також кількості типів елементів даних DET (відмінних один від одного полів записів), що додаються, модифікуються, стертих або створюваних в вихідних даних (таблиця 5).

Таблиця 5 – Рівні складності для зовнішніх входів (EI)

	1-4 DET	5-15 DET	16 І і більше DET
0-1 FTR	Низкий	Низкий	Середній
2 FTR	Низкий	Середній	Високий
3 і більше FTR	Середній	Високий	Високий

Рівні складності для зовнішніх виходів (EO) в залежності від числа FTR і DET наведені в таблиці 6.

Таблиця 6 – Рівні складності для зовнішніх виходів (EO)

	1-5 DET	6-19 DET	20 і більше DET
0-1 FTR	Низкий	Низкий	Середній
2-3 FTR	Низкий	Середній	Високий
4 і більше FTR	Середній	Високий	Високий

Рівні складності для зовнішніх запитів (EQ) визначаються наступним чином:

1) вхід зовнішнього запиту розглядається аналогічно зовнішньому входу (EI) , відображено в таблиці 5;

2) вихід зовнішнього запиту розглядається аналогічно зовнішнього виходу (EO), відображено в таблиці 6;

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

Вагові коефіцієнти рівнів складності для різних характеристик функціональності програмного забезпечення наведені в таблиці 7.

Таблиця 7 – Вагові коефіцієнти рівнів складності

	Низький	Середній	Високий
ILF	7	10	15
EIF	5	7	10
EI	3	4	6
EO	4	5	7
EQ	3	4	6

Загальна формула для обчислення нескоригованої кількості функціональних точок (UFPC - Unadjusted Function Point Count):

$$UFPC = (3*LEI) + (4*AEI) + (6*HEI) + (4*LEO) + (5*AEO) + (7*HEO) + (7*LILF) + (10*AILF) + (15*HILF) + (5*LEIF) + (7*AEIF) + (10*HEIF) + (3*LEQ) + (4*AEQ) + (6*HEQ)$$

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 внутрішніх 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 рівня складності;

AEIF - кількість зовнішніх інтерфейсних файлів середнього рівня

складності;

HEIF - кількість зовнішніх інтерфейсних файлів високого рівня складності;

EQ - кількість зовнішніх запитів низького рівня складності;

AEQ - кількість зовнішніх запитів середнього рівня складності;

HEQ - кількість зовнішніх запитів високого рівня складності.

Для отримання остаточного результату аналізу, тобто скоригована кількості функціональних точок (AFPC - Adjusted Function Point Count), необхідно також врахувати ряд загальних вимог до проєкту, для чого отримана Нескоригована кількість функціональних точок множиться на спеціальним чином розрахований нормуючий фактор (VAF - Value Adjustment Factor).

У методі функціональних точок нормуючий фактор визначається шляхом аналізу 14 основних характеристик програмних продуктів (GSC - General System Characteristics), метою застосування яких і є облік загальних вимог до програмних продуктів.

Основні характеристики системи (GSC), кожна з наведених характеристик системи оцінюється експертним способом, числом від 0 (якщо вона не присутня або не має значення для даного ПЗ) до 5 (якщо вона має дуже сильний вплив на дане ПЗ):

1. Обмін даними (0 - продукт являє собою автономне додаток; 5 - продукт обмінюється даними по більш, ніж одному телекомунікаційному протоколу).

2. Розподілена обробка даних (0 - продукт не переміщає дані; 5 - розподілена обробка даних виконується декількома компонентами системи).

3. Продуктивність (0 - призначені для користувача вимоги по продуктивності не встановлені; 5 - час відгуку сильно обмежена критично для всіх бізнес-операцій, для задоволення вимог необхідні спеціальні проєктні рішення і інструменти аналізу.

4. Обмеження по апаратних ресурсів (0 - немає обмежень; 5 - продукт цілком повинен функціонувати на певному процесорі і не може бути розподілений).

5. Транзакційна навантаження (0- транзакцій небагато, без піків; 5 - число транзакцій велике і нерівномірно, потрібні спеціальні рішення і інструменти).

6. Інтенсивність взаємодії з користувачем (0 - всі транзакції обробляються в пакетному режимі; 5 - більше 30% транзакцій - інтерактивні).

7. Ергономіка (Ефективність роботи кінцевих користувачів) (0 - немає спеціальних вимог; 5 - вимоги щодо ефективності дуже жорсткі).

8. Інтенсивність зміни даних(ILF) користувачами (0 - не потрібні; 5 - зміни інтенсивні, жорсткі вимоги по відновленню).

9. Складність обробки (0 - обробка мінімальна; 5 - вимоги безпеки, логічна і математична складність, багатопоточність).

10. Повторне використання(0 - не вимагається; 5 - продукт розробляється як стандартний багаторазовий компонент).

11. Зручність інсталяції (0 - немає вимог; 5 - установка і оновлення ПЗ проводиться автоматично).

12. Зручність адміністрування(0 - не вимагається; 5 - система автоматично самовідновлюється).

13. Можливість портування (0 - продукт має тільки 1 інсталяцію на єдиному процесорі; 5 - система є розподіленою і передбачає установку на різні «залізо» і ОС).

14. Гнучкість (0 - не вимагається; 5 - гнучка система запитів і побудова довільних звітів, модель даних змінюється користувачем в інтерактивному режимі).

Кожна з наведених характеристик системи оцінюється експертним способом, числом від 0 (якщо вона не присутня або не має значення для даного ПЗ) до 5 (якщо вона має дуже сильний вплив на дане ПЗ).

Значення всіх 14 характеристик підсумовуються для отримання підсумкового ступеня впливу (TDI - Total Degree of Influence):

$$TDI = \Sigma DI$$

Нормуючий фактор (VAF) розраховується за формулою:

$$VAF = (TDI * 0.01) + 0.65$$

Таким чином, нормуючий фактор може приймати значення від 0.65 до 1.35, а нормована кількість функціональних точок представляє собою добуток ненормований кількості функціональніх точок на нормуючій фактор:

$$AFPC = UFPC \cdot VAF$$

Надалі нормовану кількість функціональних точок може бути використано для отримання оцінки кількості рядків вихідного коду (SLOC - Source Lines of Code).

Статистичними методами визначено значення «мовних множників – рівня мови» для основних мов програмування. Таким чином, якщо мова реалізації обрана, то можна оцінити кількість рядків вихідного коду, шляхом множення нормованої кількості функціональних точок на відповідний множник:

$$SLOC = AFPC \cdot LM$$

де LM -множник мови програмування

Початкова оцінка кількості вирівняних функціональних точок враховує лише нову функціональність, що реалізується у продукті.

Підрахунок функцій проекту розробки (Development Functional Point, DFP) відбувається за формулою:

$$DFP = (UFP + CFP) * VAF,$$

де CFP (Conversion Functional Point) — функціональні точки, підраховані для додаткової функціональності, яка потрібна для встановлення продукту, наприклад, міграції даних.

Наступна формула використовується для проєктів удосконалення (Enhancement Functional Point, EFP) за формулою:

$$EFP = [(ADD + CHGA + CFP) * VAFA] + (DEL* VAFB),$$

де:

ADD - це нескоригована кількість функціональних точок тих функцій, які були додані в рамках проєкту вдосконалення;

CHGA - це невідкоригована кількість функціональних точок тих функцій, які були змінені в рамках проєкту вдосконалення (відображає функції після модифікацій);

CFP - кількість функціональних точок, додана в результаті перетворення;

VAFA - величина фактора вирівнювання програми після завершення проєкту;

DEL - невідкоригована кількість функціональних точок тих функцій, які були видалені проєктом вдосконалення. Важливо враховувати абсолютне значення DEL, а не від'ємне значення.

VAFB - величина фактора вирівнювання розрахункового до початку проєкту.

На практиці часто буває що VAFA = VAFB = VAF, таким чином, рівняння стає таким:

$$EFP = (ADD + CHGA + CFP + DEL) * VAF)$$

Проєкт після вдосконалення програми

$$AFP = [(UFPB + ADD + CHGA) - (CHGB + DEL)] * VAFA$$

де:

UFPB = нескоригована кількість функціональних балів перед покращенням.

AFP = підрахунок точки функції програми

DEL = кількість видалених функціональних точок (від'ємне значення).

Усі інші абревіатури такі ж, як і раніше.

Звичайно, розрахунок покращення може додавати та/або видаляти функції з UFPB. Додана функціональність може бути за рахунок нових компонентів або додана функціональність може бути за рахунок збільшення розміру існуючих компонентів.

Приклад

Для виконання лабораторної роботи використано додаток, який розроблено як комплекс лабораторних робіт та який призначено для ведення відомості складу. Взаємозв'язок між акторами та варіантами використання наведено на рис.7.

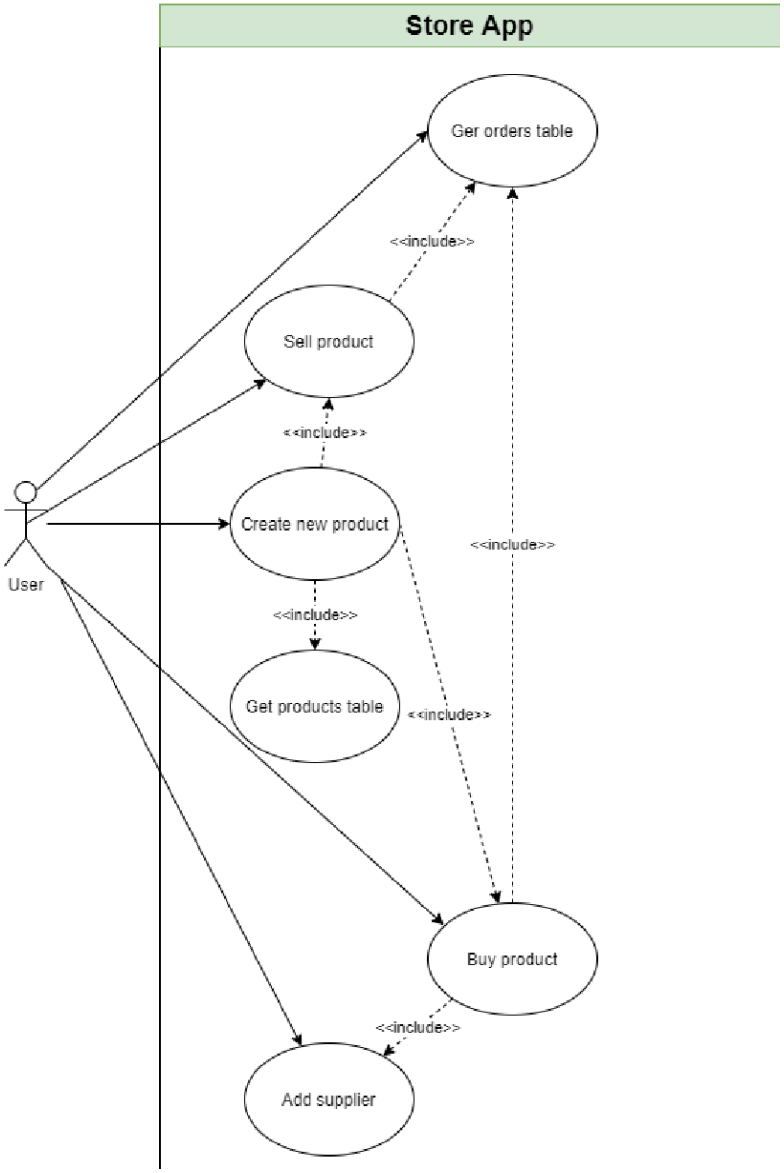


Рисунок 7 – Взаємозв'язок між акторами та варіантами використання

Бекенд проєкту реалізований з використанням технології ASP.NET, фронтенд з використанням React бібліотеки. Репозиторій проєкту доступний за посиланням: https://github.com/.../.NetLabs

Далі описано визначення кількості внутрішніх логічних файлів (Internal Logical Files, ILF) та зовнішні інтерфейсні файли (External Interface Files, EIF).

Дані, якими оперує програма зберігаються у створеній раніше БД, яка є єдиним ILF в програмі. У продукті використовуються 3 таблиці з даними:

1. Products - таблиця продуктів
2. Sell orders - таблиця накладних продажу
3. Buy orders - таблиця накладних купівлі

Оскільки додаток невеликий, та у ньому не застосовуються зовнішні інтерфейсні файли. Отже, ILF - 1, EIF - 0.

Опис транзакцій

Зовнішні входи (External inputs)

1. Додавання нового продукту з такими полями: назва, кількість, ціна за одиницю

2. Формування накладної купівлі товару з полями: кількість товару

3. Формування накладної продажу товару з полями: кількість товару, клієнт

4. Додавання постачальника з полем постачальник

Зовнішні виходи (External Outputs)

1. Обрахунок кількості проданих та наявних товарів і визначення статусу для накладних

Зовнішні запити (External Inquiries)

1. Отримання запису продукту із полями: ідентифікатор продукту, назва продукту, ціна, кількість

2. Отримання запису накладної продажу із полями: ідентифікатор накладної, ідентифікатор продукту, дата, ціна, кількість, покупець, статус.

3. Отримання запису накладної купівлі із полями: ідентифікатор накладної, ідентифікатор продукту, дата, ціна, кількість, постачальник, статус.

Далі слід здійснити підрахунок функціональних точок пов’язаних з даними.

Визначимо кількість DET і RET та оцінку кількості нескоригованих функціональних точок за допомогою матриці складності та оцінку UFP (для ILF та EIF). Матриця складності представлена у табл. 8.

Таблиця 8 – Матриця складності

	1-19 DET	20-50 DET	50+ DET
1 RET	Низький	Низький	Середній
2-5 RET	Низький	Середній	Високий
6+ RET	Середній	Високий	Високий

Оцінка даних нескоригованих функціональних точок для внутрішніх логічних файлів (ILFs) і зовнішніх інтерфейсних файлів (EIFs) наведено у таблиці 9.

Таблиця 9 – Оцінки даних нескоригованих функціональних точок

Складність даних	Кількість UFP (ILF)	Кількість UFP (EIF)
Низький	7	5
Середній	10	7
Високий	15	10

Розрахунок скоригованих функціональних точок для внутрішніх логічних файлів показано на рисунку 8.

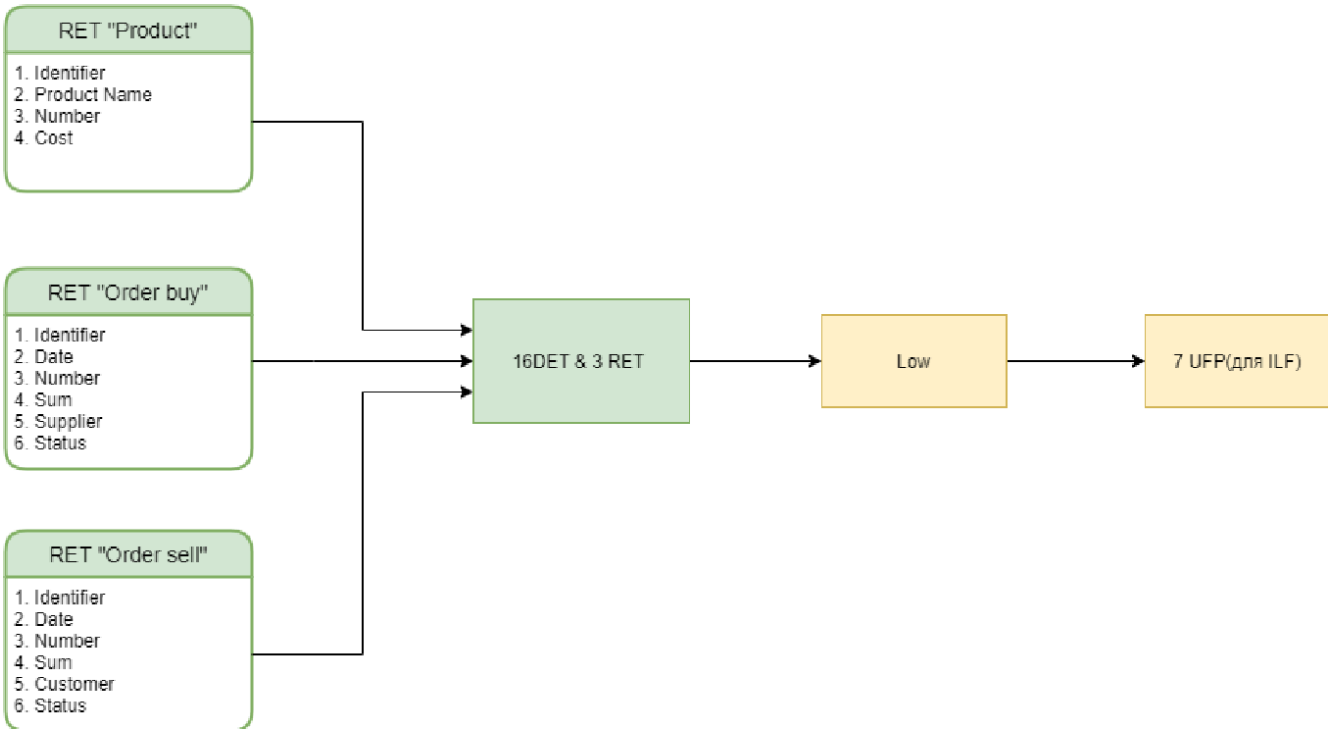


Рисунок 8 – Розрахунок скоригованих функціональних точок для внутрішніх логічних файлів

Матрицю складності зовнішніх вхідних транзакцій (EI) наведено у таблиці 10.

Таблиця 10 – Матриця складності зовнішніх вхідних транзакцій

EI	1-4 DET	5-15 DET	16+DET
0-1 FTR	Низький	Низький	Середній
2 FTR	Низький	Середній	Високий
3+ FTR	Середній	Високий	Високий

Матриця складності зовнішніх вихідних транзакцій і зовнішніх запитів (EO &EQ) представлена у таблиці 11.

Таблиця 11 – Матриця складності зовнішніх вихідних транзакцій і зовнішніх запитів

EO&EQ	1-5 DET	6-19 DET	20+ DET
0-1	Низький	Низький	Середній
2-3	Низький	Середній	Високий
4+	Середній	Високий	Високий

Матриця складності в нескоригованих функціональних точках наведена у таблиці 12.

Таблиця 12 – Матриця складності в нескоригованих функціональних точках

Складність транзакцій	Кількість UFP (EI, EQ)	Кількість UFP (EO)
Низький	3	4
Середній	4	5
Високий	6	7

Визначимо складність для EI транзакцій (рис. 9):

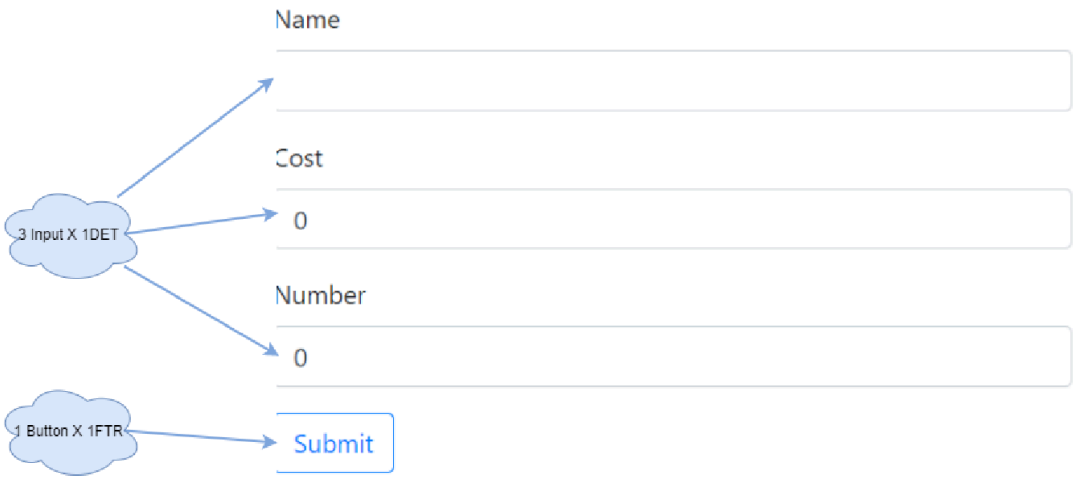


Рисунок 9 – Складність для EI транзакцій

Складність транзакції “Додавання нового товару”: Low (рис. 10).

Рисунок 10 –. Створення накладної продажу

Складність транзакції “Створення накладної продажу ”: Low (рис. 11).

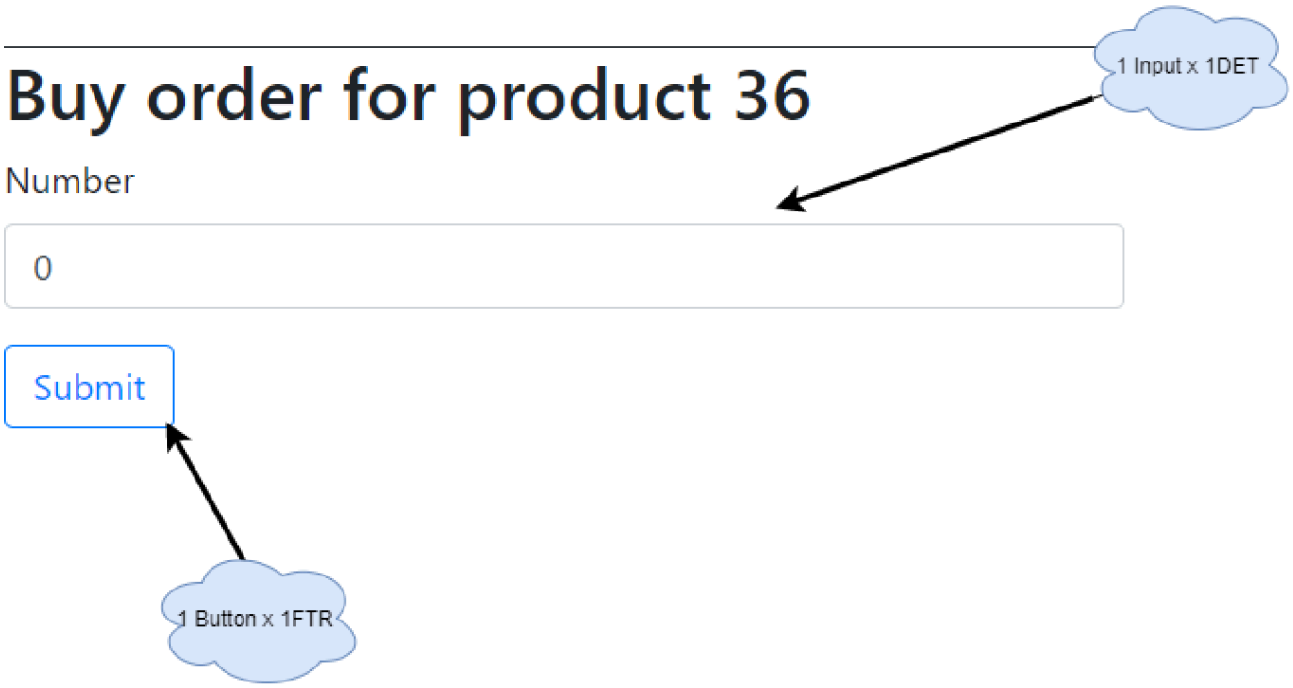


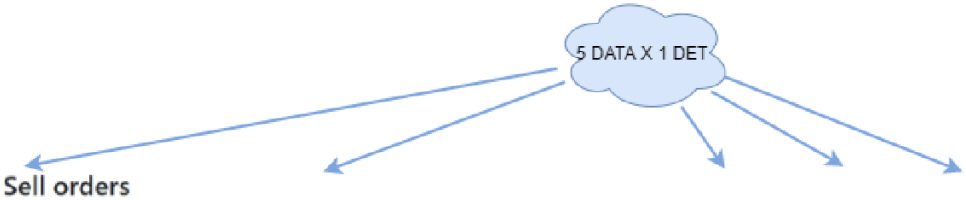
Рисунок 11 – Створення накладної покупки

Складність транзакції “Створення накладної покупки”: Low (рис. 12).

Рисунок 12 – Додавання постачальника

Складність транзакції “Додавання постачальника”: Low

Визначимо складність для EQ (рис. 13):

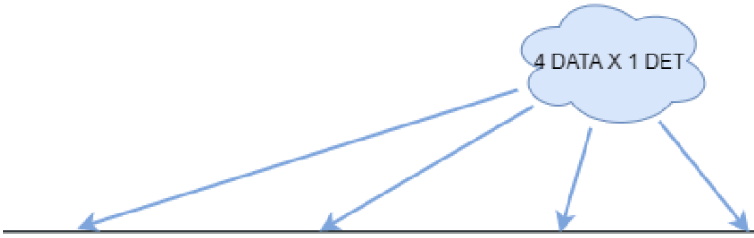


Sell orders

Id	Date	Product id	Number	Sum	Customer	Status
69	2020-12-23T22:43:46.1698374	36	100	10000	customer	Closed
70	2020-12-23T22:45:22.3814164	36	50	5000	customer 1	Closed
71	2020-12-24T15:12:48.1016512	36	10	1000	customer	Closed
72	2020-12-24T17:13:58.958917	37	20	3000	customer	Closed
73	2020-12-24T17:14:19.9808381	36	10	1000	customer	Closed
74	2021-10-10T00:17:04.9682144	36	200	20000	New customer	Closed
75	2021-10-10T00:17:09.3898628	36	200	20000	New customer	Not closed

Рисунок 13 – Таблиця записів накладних продажу

Складність транзакції “Отримати запис накладної продажу”: Low (рис. 14)



Products

#	Product	Cost	Number	Actions
36	Product 1	100	200	Sell Buy
37	new product	150	4	Sell Buy
38	product 2	200	200	Sell Buy
39	Product 1	100	500	Sell Buy

Рисунок 14 – Таблиця записів накладних продуктів

Складність транзакції “Отримати запис продукту”: Low (рис. 15).

id	Date	Product id	Number	Sum	Supplier	Status
52	2020-12-23T22:44:55.9591989	36	10	1000	supplier 1	Closed
53	2020-12-23T22:45:22.3814633	36	50	5000	supplier 1	Closed
54	2020-12-24T17:14:19.9808633	36	10	1000	supplier 23	Closed
55	2020-12-24T17:14:45.8227968	36	100	10000	supplier 23	Closed
56	2020-12-24T17:14:46.4828604	36	100	10000	Supplier	Closed
57	2021-10-10T00:17:04.9689722	36	200	20000	Test supplier	Closed
58	2021-10-10T00:17:09.3898926	36	200	20000		Submit

Рисунок 15 – Таблиця записів накладних покупки

Складність транзакції “Отримати запис накладної покупки”: Low

Визначимо складність для ЕО (рис. 16):

Sell orders

Id	Date	Product id	Number	Sum	Customer	Status
69	2020-12-23T22:43:46.1698374	36	100	10000	customer	Closed
70	2020-12-23T22:45:22.3814164	36	50	5000	customer 1	Closed
71	2020-12-24T15:12:48.1016512	36	10	1000	customer	Closed
72	2020-12-24T17:13:58.958917	37	20	3000	customer	Closed
73	2020-12-24T17:14:19.9808381	36	10	1000	customer	Closed
74	2021-10-10T00:17:04.9682144	36	200	20000	New customer	Closed
75	2021-10-10T00:17:09.3898628	36	200	20000	New customer	Not closed

Рисунок 16 – Записи накладних продажу (транзакція “Отримання статусу накладної”)

Складність транзакції “Отримання статусу накладної”: Low

Визначимо ненормовану кількість функціональних точок UFPC (табл. 13):

Таблиця 13 – Ненормована кількість функціональних точок UFPC

Параметр	Кількість	Ваговий коефіцієнт	Total
EI	4	Low - 3	12
EQ	3	Low - 3	9
EO	1	Low - 4	4
ILF	1	Low - 7	7
EIF	0	Low - 5	0
UFPC			32

Визначимо основні характеристики системи:

1) Обмін даними - 0, продукт є автономним додатком

2) Розподілена обробка даних - 1, готує дані для обробки іншими засобами, СУБД

3) Продуктивність - 0, жодних спеціальних вимог по продуктивності не було встановлено

4) Обмеження по апаратних ресурсів - 0, немає обмежень

5) Транзакційна навантаження - 0, небагато транзакцій

6) Інтенсивність взаємодії з користувачем - 5, з розглянутих транзакцій 3 із семи вимагають користувацького вводу

7) Ергономіка - 0, немає спеціальних вимог

8) Інтенсивність зміни даних - 0, відсутнє

9) Складність обробки даних - 0, мінімальна обробка даних

10) Повторне використання - 0, не вимагається

11) Зручність інсталяції - 0, немає особливих потреб

12) Зручність адміністрування - 0, не вимагається

13) Можливість портування - 3, виконується на різному апаратному забезпеченні

14) Гнучкість - 0, не вимагається

Підсумковий ступінь впливу розраховується наступним чином:

$TDI = 9$

VAF = (TDI * 0.01) + 0.65 = (9*0.01) + 0.65 = 0.74

Нормована кількість функціональних точок розрахована за формулою:

AFPC = UFPC * VAF = 32 * 0.74 = 23.68

ЗАВДАННЯ

1. Проаналізувати власний програмний застосунок та скласти перелік та навести докладний опис всіх внутрішніх логічних (ILF) і зовнішніх інтерфейсних (EIF) файлів, а також всіх транзакцій (EI, EO, EQ).

2. Виконати оцінки кількості RET і DET для внутрішніх логічних (ILF) і зовнішніх інтерфейсних (EIF) файлів, а також оцінки кількості FTR і DET для зовнішніх вводів (EI), виведень (EO) і запитів (EQ), оцінити складність та кількість ненормованих функціональних точок.

3. Провести аналізу ступенів впливу основних характеристик системи.

4. Розрахувати нормовану кількість функціональних точок та економічних характеристик.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які метрики зазвичай використовують для оцінки розміру?

2. Які метрики розміру називають внутрішніми/зовнішніми і чому?

3. Які переваги є у функціональних точок у порівнянні з рядками вихідного коду при оцінці розміру на ранніх стадіях проєкту?

4. Що розуміється під функціональністю даних?

5. Що розуміється під функціональністю транзакцій?

6. Що таке внутрішній логічний файл (ILF)?

7. Що таке зовнішній інтерфейсний файл (EIF)?

8. У чому основна відмінність між внутрішнім логічним файлом (ILF) і зовнішнім інтерфейсним файлом (EIF)?

9. Що таке зовнішнє введення (EI)?

10. Що таке зовнішній виведення (EO)?

11. Що таке зовнішній запит (EQ)?

12. У чому основна відмінність між зовнішнім введенням (EI) і зовнішнім запитом (EQ)?

13. У чому основна відмінність між зовнішнім виводом (EO) і зовнішнім запитом (EQ)?

14. Як оцінюються рівні складності внутрішніх логічних (ILF) і зовнішніх інтерфейсних файлів (EIF)?

15. Як оцінюються рівні складності зовнішніх вводів (EI), виведень (EO) і запитів (EQ)?

16. Яким чином проводиться аналіз ступенів впливу основних характеристик системи і обчислюється нормирующий фактор VAF?

20. У чому принципова відмінність ненормованого кількості функціональних точок від нормованого? Як сильно вони можуть відрізнятися?

21. З яких основних кроків (етапів) складається процес аналізу функціональних точок?

22. Що служить вихідною інформацією для процесу аналізу функціональних точок?

ПРАКТИЧНА РОБОТА № 4

ДОДАТКОВІ ВИДИ ТОЧОК ДЛЯ ОЦІНКИ РОЗМІРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета: Навчитися оцінювати розмір програмного забезпечення шляхом застосування додаткових видів точок, а саме Feature points, Object points та Use-case points.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Оскільки метод функціональних точок вперше був використаний для інформаційних систем, фахівці думали про нього як про метод, який можна використовувати для такого роду програмного забезпечення. З цієї причини виникла величезна кількість модифікованих концепцій цього методу.

Сьогодні існує близько 35 різних версій, що застосовуються промисловістю. Найбільш поширені наступні:

- Feature points;
- Full function points;
- Object points;
- Data points;
- Story Points;
- Use-case points.

Feature points. Цей тип точок призначений для визначення функціонального розміру програмного забезпечення, особливо для програмного забезпечення в реальному часі або вбудованих програм. Метод функціональних точок був створений для оцінювання розміру програмного забезпечення Management Information Systems (MIS) але він був неточний для оцінювання розміру систем реального часу, або систем, які містять реалізацію складних алгоритмів. Тому Capers Jones розробив додатковий до Function Point від точок, який назвав Feature points. Найбільш помітна різниця між Function Point та Feature points полягає в тому, що останні використовують додатковий компонент, алгоритми, додаючи його до набору з п'яти компонентів Function

Point: inputs, outputs, inquiries, external interface files, and internal logical files. Більш важкі системи, схоже, мають більшу алгоритмічну складність, ніж програмне забезпечення MIS, але менше inputs, outputs, inquiries, external interface files, and internal logical files. Тому підрахувати такі системи за допомогою Function Point може бути трохи оманлива. Метод Feature Point є хорошою альтернативою для цього питання, оскільки він компенсує ці проблеми, враховуючи алгоритми. Кожному алгоритму також присвоюється значення ваги, як і решті компонентів. Крім того, значення, призначені для файлів логічних даних, зменшуються, оскільки вони менш значущі для більш важких типів систем. Таким чином, Unadjusted Feature Point обчислюються наступним чином:

Unadjusted Feature Point = (External Inputs * Weight) +
(External Outputs * Weight) + (Logical Internal Files * Weight) +
(Logical Interface Files * Weight) + (Inquiries * Weight) +
(Algorithms * Weight)

Посібник з практики підрахунку Function Point дає конкретні вказівки щодо визначення "ступеня впливу" від 0 до 5 для кожної з чотирнадцяти загальних характеристик системи. Вони також пропонуються для використання в Value Adjustment Factor для Feature Points. Цей розрахунок надає the Adjusted Feature Point для Feature Points.

Таким чином:

Value Adjustment Factor = Total Degree of Influence * .01 + .65

Нарешті,

Adjusted Feature Points = (Unadjusted Feature Points) *

(Value Adjustment Factor)

Full function points (FFP), метод який є розширенням FPA для систем реального часу. Він базується на роботі, проведеній the Software Engineering Management Research Laboratory at the Université du Québec à Montréal and

Software Engineering Laboratory in Applied Metrics (SELAM), і був представлений в 1997 році.

Програмне забезпечення систем реального часу містять більше даних управління та під процесів, ніж програмне забезпечення ІС. Оскільки FFP є продовженням FPA, усі правила IFPUG включені до FFP, але незначна кількість підмножин правил IFPUG, що стосуються концепцій управління, була значно розширена. FFP вирішує це шляхом введення додаткових типів Data Functions та Transaction Functions.

FFP, як і FPA, вимірює функціональний розмір, оцінюючи Data Functions та Transaction Function. Однак, на відміну від FPA, Transaction Function оцінюються на рівні під процесу. Тобто, в Processing Logic Form ідентифікуються підпроцеси, які класифікуються та оцінюються. Значення факторів функціонального розміру (FTR) нараховуються на рівні під процесу. Додатково підпроцеси в рамках кожного процесу Transaction Function класифікують на один із чотирьох типів:

- зовнішній контроль (ECE);
- зовнішній вихід управління (ECX);
- зчитування внутрішнього контролю (ICR);
- запис внутрішнього контролю (ICW).

Object Points (OP), які також називаються application points, щоб уникнути непорозумінь з об'єктами в об'єктно-орієнтованих системах, засновані на кількості вікон, звітів, модулів 3GL та правил. Для обчислення Object Points обчислюють кількість екранів, звітів та модулів, які було створено за допомогою інтегрованих середовищ інженерії програмного забезпечення (CASE). Зважаючи за допомогою таблиць результати обчислень та сумуючи з урахування повторно використаного коду отримують кількість Object Points.

Модель композиції програми (Application Composition Model – ACM) включає оцінку прототипу користувальницького інтерфейсу. Вона використовується на ранніх стадіях розробки проєкту та орієнтована на проєкти, що створюються із застосуванням сучасних інструментальних засобів та UML.

Мета моделі: оцінка інтерфейсу користувача, взаємодії із системою, продуктивності. Така оцінка дозволяє вибрати кілька можливих варіантів подальшого розвитку проєкту. ACM використовує як метрику розміру програмного забезпечення об'єктні одиниці.

Об'єктна одиниця визначається шляхом підрахунку кількості екранів інтерфейсу користувача, звітів і компонентів. Кожен із цих вимірів оцінюється за складністю.

Складність екрану оцінюється в залежності від кількості таблиць даних і кількості подання екранів (простий екран – 1, складний – 3). Складність звіту також оцінюється в залежності від кількості його подань (простий – 2, середній – 5, складний – 8).

Окремі компоненти, що відображають обчислювальні компоненти, архітектурні перебудови, генерацію графіки, вважаються складними та мають коефіцієнт 10.

Якщо припустити, що у проєкті буде використано r відсотків об'єктів із раніше створених проєктів, кількість нових об'єктних одиниць у проєкті Object Points (OP) можна розрахувати, як:

$$OP_{new} = OP \cdot \frac{100 - r}{100},$$

де OP – кількість об'єктних одиниць у раніше розроблених проєктах;

OPnew – кількість нових об'єктних одиниць у розглянутому проєкті.

Трудовитрати розраховуються за формулою:

$$E = \frac{OP_{new}}{PROD},$$

де E – трудовитрати у людино-місяцях;

PROD – продуктивність, встановлюється залежно від досвідченості працівника та зрілості середовища розробки.

Оцінюється за п'ятиінтервальною шкалою:

- дуже низька - 4;

- низька – 7;

- номінальна – 13;

- висока – 25;

- дуже висока – 50.

Story Points використовуються командами Agile під час спринтерської сесії. Кожної історії користувача (функція програмного забезпечення) призначається значення Story Points , засноване на рівні складності конкретної історії (функції). Балі - відносна міра виражається відповідно до числового діапазону, який, як правило, є обмеженим набором чисел. Числа вибираються на основі сприйняття колективом розміру роботи, яку необхідно виконати. Визначення розміру базується на рівні розуміння ступені складності проєкту, можливості команди та кількості роботи. Особливістю Story Points є те, що кожна команда визначає їх так, як вважає за потрібне. Одна команда може вирішити визначити Story Points як ідеальний робочий день (тобто день без будь-яких перерв - без зустрічей, без електронної пошти, без телефонних дзвінків, і так далі). Інша команда може визначити Story Points як ідеальний тиждень роботи або може визначити Story Points в іншої мірі складності історії.

Use-case points. У 1993 р. Густавом Карнером (Gustav Karner) з Objectory (нині Rational Software) був розроблений метод, заснований на аналізі Use-cases, а також Activity діаграм для визначення та оцінки проєктів, що створено за об'єктно-орієнтованим методом. З використанням Use-case points для оцінювання розміру існує два метода:

1. Ідентифікувати actors і Use-case.

2. Обчислити, як для функціональних точок, але використовуючи Use-case діаграми та Activity діаграми, кількість input, output файлів і data inquiries.

Відповідно до першого методу Use-case points обчислюються наступним чином:

1. Ідентифікуються і зважуються actors (UAW). Цей крок, полягає в тому, щоб класифікувати акторів як простих, середніх або складних. Простий актор представляє іншу систему з визначеним API інтерфейсом програмування,

середній актор це інша система, що взаємодіє через протокол, такий як TCP/IP, і складний актор може бути особа, яка взаємодіє через графічний інтерфейс або веб-сторінку. Вагові коефіцієнти присвоюється кожному типу актора (табл. 14).

Таблиця 14 – Оцінка акторів (UAW)

Тип	Коефіцієнт
Простий (використовує систему за допомогою перерозподіленого API)	1
Середній (використовує більш складний або гнучкий API)	2
Складний (в основному означає взаємодію з кінцевим користувачем)	3

2. Ідентифікуються і зважуються Use-case (UUCW). На цьому кроці кожен Use case визначається як простий, середній або складний, залежно від кількості транзакцій в описі Use case, включаючи вторинні сценарії. Транзакція є сукупність видів діяльності, яка або виконується повністю, або взагалі не виконується. Підрахування кількості транзакцій можна здійснити, підраховуючи кроки Use cas.

Іншим механізмом вимірювання складності Use case є підрахунок кількості класів, які будуть використано в містах транзакцій, як тільки буде визначено, які класи реалізують конкретний Use case. Use case простий складності – такий, що реалізований 5 або меншій кількістю класів, Use case середній складності - на 5-10 класів, а складний Use case, це більш ніж на десять класів. Ваги такі, як і раніше.

3. Обчислити Unadjusted Use-case points:

$$UUCP = UAW + UUCW$$

4. Обчислити значення Adjustment Factors (VAF). Коефіцієнти технічних факторів відповідають технічним факторам коригування складності методу FPA, а фактори зовнішнього середовища це множник для кількісної оцінки нефункціональних вимог, таких, наприклад, як простота використання та мотивація програміста. Фактори, що впливають на продуктивність,

асоціюються з вагою, а значення призначається кожному фактору залежно від ступеня впливу. Так 0 означає відсутність впливу, 3 є середнім, 5 означає сильний вплив.

Technical Complexity Factor Фактор технічної складності (TCF) обчислюється множенням значення кожного з них фактор (T1-T9) за його вагою, а потім додаючи всі ці числа, щоб отримати суму, що позначається як TFactor. Застосовується наступна формула:

$$TCF=0.6 + (0.01 * TFactor)$$

Environmental Factor фактор середовища (EF) обчислюється множенням значення кожного фактора (F1- F8) за його вагою та додаванням результатів, щоб отримати суму, що позначається EFactor. Застосовується наступна формула:

$$EF = 1.4 + (-0.03 * EFactor)$$

5. Обчислити Adjusted Use-case points (UCP = UUCP * VAF).

Data Points (Точки даних), пов'язані з підрахунком функціональних можливостей систем на основі бази даних, які були б корисними для визначення вартості програмного забезпечення, яке містить дуже велику базу даних. Будь-яке програмне забезпечення завжди складається з двох частин. Одна частина зберігає дані, а інша частина обробляє. Програма, велика чи маленька, містить деякі дані.

Можна розрахувати точки даних для програмного забезпечення з наступних десяти параметрів:

1) Обсяг даних. Фактична середня кількість трансакцій, які за день обробляються нашим програмним забезпеченням.

2) Тип системи управління базами даних, яка використовується для управління даними. Наприклад, RDBMS, ORDBMS, бази даних SQL тощо. Щоб визначити вагу бази даних, ми можемо враховувати різні фактори, такі як надані функції безпеки, підтримка PL/SQL, ємність керування транзакціями, керування відновленням, ємність зберігання тощо.

3) Архітектура середовища, в якому буде здійснюватися доступ до даних. Наприклад, однокористувацький, багатокористувацький, веб-орієнтований, клієнт-сервер тощо.

4) Загальна кількість таблиць (відношень). Наприклад, прості таблиці, розділені таблиці, вкладені таблиці тощо.

5) Складність ключів. Складність ключів можна визначити з атрибутами, що містять первинні або зовнішні ключи.

6) Складність відносин. Кількість об'єктів, крім таблиць. Наприклад, збережені процедури, збережені функції, тригери, методи тощо. Для цих типів об’єктів можна обчислити загальну кількість рядків кодів або обчислити FP за допомогою будь-якого стандартного методу, наприклад FPA, COCOMO або будь-якого іншого методу.

7) Загальна кількість індексів. Складність індексів залежить від загальної кількості полів, що використовуються для створення індексу.

8) Обсяг щоденних транзакцій, що обробляють базу даних.

9) Загальна кількість складних атрибутів. Наприклад, для зберігання зображень, документів, аудіо- та відео файлів тощо.

Для визначення ваги кожного компонента бази даних відповідно до їх складності та обсягу можна використовувати вагові показники. Рішення про вагові показники залежить від організації, тому що компанія, яка фактично розробляє програмне забезпечення, краще знає про вартість усіх компонентів, які використовуються при розробці системи. Розрахунок DCP відбувається за наступною формулою:

$$DPC = \sum_{i=1}^{n} (W_i * V_i)|$$

Після розрахунку кількості точок даних (DPC) його можна додати до Function Point Count, щоб оцінити остаточну вартість програмного забезпечення.

ЗАВДАННЯ

1. Використовуючи власні або запропоновані викладачем програмні застосунки обчислити наступне:

- кількість Feature points;

- Кількість Object Points;

- кількість Use-case points за першим і другим методами.

2. Визначити економічний вплив отриманих результатів на реалізацію програмного застосунка.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які види додаткових точок існують і чому.

2. Поясніть потребу у розробці кожного виду додаткових точок.

3. Як обчислюються кількість алгоритмів.

4. Поясніть сутність Feature Point Analysis.

5. Поясніть концепцію Full Function Points.

6. Поясніть як враховується повторне використання при обчисленні точок додаткових видів.

7. Поясніть сутність Object Points.

8. Поясніть сутність Object Points Analysis.

9. Поясніть сутність Use-case points.

10. Поясніть сутність Use-case points Analysis за першим методом.

11. Поясніть сутність Use-case points Analysis за другим методом.

ПРАКТИЧНА РОБОТА №5

ОЦІНЮВАННЯ ВАРТОСТІ І ІНШИХ ХАРАКТЕРИСТИК ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА МЕТОДОМ АНАЛОГІЇ

Мета: навчитися здійснювати оцінку проєкту за рахунок використання методу аналогій.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Методи і моделі оцінки вартості програмного забезпечення можна розділити на дві групи: неалгоритмічні методи і алгоритмічні моделі. До неалгоритмічних методів належать Price-to-win, оцінка по Паркінсону, експертна оцінка, оцінка по аналогії. До алгоритмічних моделей належать SLIM і COCOMO.

Сутність неалгоритмічних методів полягає в тому, що при оцінці вартості програмного забезпечення використовуються певні схеми і принципи, а не математичні формули.

Аналогією зазвичай вважають структурні зв'язки між доменом-джерелом (базою) та цільовими доменами. Щоб встановити аналогію, загальні під структури двох доменів ідентифікуються та відображаються один в одній, внаслідок чого виникає або ні аналогічне відношення.

Встановлення аналогії, як правило, регулюється певними обмеженнями, наприклад систематичність, структурна узгодженість або обмеження можливих відображень одне в одне. Загально прийнятого набору таких принципів немає. Основне поняття, яке застосовано в методі аналогії - відстань між історичними проєктами і новим проєктом. Відстань вимірює, наскільки відрізняється два проєкти (один з них історичний, а другий, новий) з точки зору значень їх атрибутів. Використовується метрика відстані на основі стандартизованих значень k атрибутів для цих проєктів. Найвідоміша така метрика відстані - евклідова або прямолінійна відстань, яка має прямолінійне геометричне значення як відстань двох точок у k-мірному евклідовому просторі.

Оцінка за аналогією здійснюється шляхом використання інформації про характеристики проєктів, які було виконано в минулому для оцінки

характеристик нового програмного забезпечення, яке буде розроблятися. Основною ідеєю оцінки вартості за методом аналогії є використання історичної інформації від завершених проєктів з відомими витратами.

Методика розв'язання типових задач, виконання типових завдань

Спочатку необхідно охарактеризувати новий проєкт, атрибутами, ідентичними тим, які притаманні виконаним проєктам, зареєстрованим в базі даних. Приклади атрибутів проєкту є довжина вихідного коду, мова програмування, досвід персоналу, інші. Потім, порівнювати з обраними виконаними проєктами.

Оцінка зусиль проєкту програмного забезпечення за аналогією зазвичай включає ряд кроків:

1. Вимірювання або оцінка значень показників для цільового проєкту.

2. Пошук у сховищі виконаних проєктів, проєктів подібних цільовому та вибір одного або декількох проєктів як вихідних аналогів.

3. Використання значення зусиль і вартості проєктів як вихідних аналогів в якості початкової оцінки для цільового проєкту.

4. Порівняння відомих метричних значень для цільового та вихідних проєктів.

5. Коригування значення зусиль і вартості з урахуванням відмінностей між ціллю та джерелами.

В таблиці 15 наведено приклад множини показників для цільового проєкту.

Таблиця 15 – Показники для цільового проєкту

	Зусилля	Атрибут 1	Атрибут 2	…	Атрибут k
Проєкт 1	E_1	X_{11}	X_{12}	…	X_{1k}
Проєкт 2	E_2	X_{21}	X_{22}	…	X_{2k}
Проєкт 3	E_2	X_{31}	X_{32}	…	X_{3k}
….	…	…	…	…	…
Проєкт n	E_n	X_{n1}	X_{n2}	…	X_{nk}
Новий проєкт	Невідомий	Y_1	Y_2	…	Y_k

Пошук у серед виконаних проєктів, проєктів подібних цільовому здійснюється шляхом використання поняття відстань. Відстань вимірює, наскільки відрізняється два проєкти з точки зору значень їх атрибутів. Використовується метрика відстані на основі стандартизованих значень атрибутів k для цих проєктів.

Найвідоміша така метрика відстані - евклідова або прямолінійна відстань, яка має прямолінійне геометричне значення як відстань двох точок у k-мірному евклідовому просторі:

$$d_{new,i} = \left\{ \sum_{j=1}^{k} (Y_j - X_{ij})^2 \right\}^{1/2}, \qquad i = 1, 2, \ldots, n$$

Евклідова або прямолінійна відстань в трьох мірному просторі продемонстрована на рисунку 17. Зазначимо, що можуть також використовуватися інші метрики відстані.

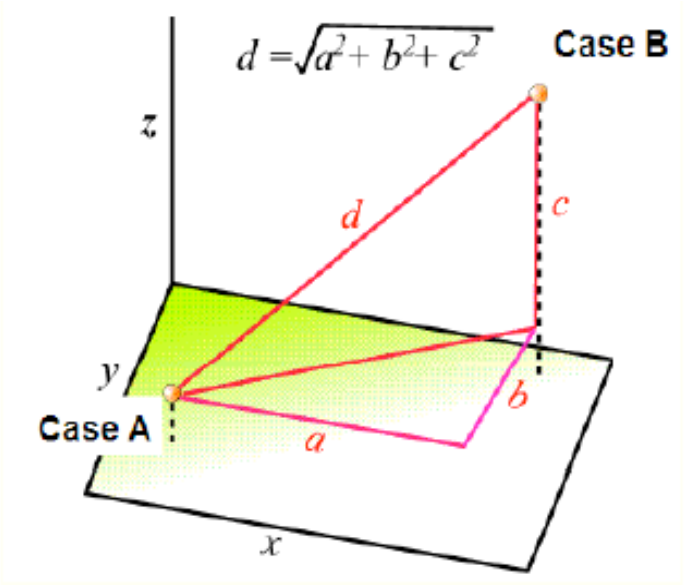


Рисунок 17 – Евклідова або прямолінійна відстань

Оцінка зусиль за допомогою аналогій базується на завершених проєктах, які є подібними до нового. Користувач методу повинен обчислити відстані нового проєкту від усіх проєктів бази даних та визначити кілька «сусідських» проєктів, тобто тих, з якими відносно невелике значення відстані. Оцінка зусиль зрештою отримують деякі поєднання зусиль сусідніх проєктів. Зазвичай використовується середня статистика (зважена або проста) або медіана цих значень зусиль. Відсутні значення потребують подальшого коригування методом аналогій. Оскільки взаємозв'язок між зусиллям і розміром є хорошим встановлену, статистику можна також коригувати, використовуючи сусідній за розміром проєкт. Наприклад, у випадку одного «сусіда» лінійне регулювання розміру виконується наступним чином:

$$EFFORT_{NEW} = \frac{SIZE_{NEW}}{SIZE_{NEIGHBOUR}} \times EFFORT_{NEIGHBOUR}$$

Калібрування методу, заснованого на аналогії, вимагає виявлення найкращих конфігурацій доступних параметрів методу. Параметри, які можуть бути скориговані, є:

(а) метрика відстані, за якою проєкти бази даних будуть відсортовані відповідно до їх подібність до оцінюваної (наприклад, евклідова відстань, відстань Манхеттена);

(b) кількість найближчих проєктів (аналогій);

(c) набір атрибутів для оцінювання аналогій;

(d) статистичні дані, які будуть обчислені з зусиль найближчих проєктів і будуть виконувати функції оцінки зусиль для нового проєкту.

Крім того, статистика може бути скоригована за розміром.

Можна використовувати такі набори даних, як база даних Кемерера (табл. 16), база даних Альбрехта (табл. 17), COCOMO Dataset та інші.

Таблиця 16 – База даних Кемерера

Програмне забезпечення	Апаратне забезпечення	Місяці	Рядки коду, тис	Рядки коду, людино-місяці	Кориговані функціональні точки	Некореговані функціональні точки	Зусилля
1	1	17	253,6	884	1217.1	1010	287
1	2	7	40,5	491	507,3	457	82,5
1	4	15	450	406	2306,8	2284	1107,3
1	1	18	214,4	2467	788,5	881	86,9
1	2	13	449,9	1338	1337,6	1583	336,3
1	3	5	50	595	421,3	411	84
2	3	5	43	1853	99,9	97	23,2
1	2	11	200	1535	993	998	130,3
1	1	14	289	2491	1592,9	1554	116
3	1	5	39	542	240	250	72
1	1	13	254,2	983	1611	1603	258,7
1	2	31	128,6	557	789	724	230,7
1	5	20	161,4	1028	690,9	705	157
1	1	26	164,8	667	1374,5	1375	246,9
3	1	14	60,2	861	1044,3	976	69,9

Таблиця 17 – База даних Альбрехта

Вхід	Вихід	Файл	Запити	Функціональні точки	Рядки коду, тис	Некореговані функціональні точки	Зусилля
25	150	60	75	1750	130000	1750	102,4
193	98	36	70	1902	318000	1902	105,2
70	27	12	0	428	20000	535	11,1
40	60	12	20	759	54000	660	21,1
10	69	9	1	431	62000	478	28,8
13	19	23	0	283	28000	377,33	10
34	14	5	0	205	35000	256,25	8
17	17	5	15	289	30000	262,73	4,9
45	64	16	14	680	48000	715,79	12,9
40	60	15	20	794	93000	690,43	19
41	27	5	29	512	57000	465,45	10,8
33	17	5	8	224	22000	298,59	2,9
28	41	11	16	417	24000	490,59	7,5
43	40	35	20	682	42000	802,35	12
7	12	8	13	209	40000	220	4,1
28	38	9	24	512	96000	487,62	15,8
42	57	5	12	606	40000	550,91	18,3
27	20	6	24	400	52000	363,64	8,9
48	66	50	13	1235	94000	1073,91	38,1
69	112	39	21	1572	110000	1310	61,2

Вхід	Вихід	Файл	Запити	Функціональні точки	Рядки коду, тис	Некореговані функціональні точки	Зусилля
25	28	22	4	500	15000	476,19	3,6
61	68	11	0	694	24000	694	11,8
15	15	3	6	199	3000	189,52	0,5
12	15	15	0	260	29000	237,68	6,1

Для розрахунку відстаней можна діяти наступними шляхами, а саме, по перше, обчислювати вручну, по другу, запрограмувати калькулятор, або по третє, використати одну з відомих систем оцінювання.

Приклад виконання

1. Загальні відомості про проєкт що використовується.

Датасет у вигляді електронної таблиці, яка буде використовуватись для розрахунків (рис. 18).

Атрибути:

- довжина розробки (місяців);
- кількість транзакцій;
- кількість сутностей в моделі даних системи;
- ненормовані функціональні точки;
- нормовані функціональні точки.

Мови програмування (на яких розроблені проєкти для датасету):

Проєкт №1 – C.

Проєкт №2 – Java.

Проєкт №3 – C#.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Project	TeamExp	ManagerExp	YearEnd	Length	Transactions	Entities	PointsNonAdjust	Adjustment	PointsAjust	Language	Effort
2	1	2	2	88	3	126	49	175	38	180	3	651
3	2	4	4	85	16	116	170	286	27	263	1	7252
4	3	2	3	84	13	134	77	211	13	165	2	2275
5	4	1	4	86	8	89	200	289	33	283	1	3983
6	5	1	1	84	3	33	72	105	19	88	1	2282
7	6	1	3	86	17	317	119	436	34	432	2	9135
8	7	4	1	85	14	175	277	452	37	461	1	3948
9	8	2	3	87	27	173	332	505	19	424	1	14987
10	9	1	3	86	6	42	31	73	27	67	2	1267
11	10	2	1	84	17	146	112	258	40	271	1	9051
12	11	0	0	86	4	140	94	234	24	208	1	2149
13	12	3	1	85	14	148	324	472	39	491	1	4277
14	13	2	4	85	18	88	170	258	34	255	1	5180
15	14	2	-1	87	10	224	110	334	28	309	2	6783
16	15	2	3	85	8	106	39	145	6	103	1	2331
17	16	3	4	87	14	9	386	395	21	340	2	4494
18	17	1	4	88	4	158	59	217	18	180	3	847
19	18	1	3	86	12	132	89	221	5	155	2	3647
20	19	2	0	86	6	71	235	306	37	312	1	3542
21	20	4	3	86	6	79	128	207	27	190	1	3927
22	21	3	4	87	10	101	57	158	9	117	2	1155
23	22	4	5	86	26	482	227	709	26	645	2	9100
24	23	4	4	86	14	68	316	384	20	326	2	3437
25	24	1	4	85	12	253	52	305	34	302	1	5152
26	25	4	4	85	1	40	60	100	18	83	1	805
27	26	4	4	85	36	886	241	1127	34	1116	1	23940
28	27	4	4	86	12	318	269	587	34	581	2	14973
29	28	2	3	86	24	473	182	655	40	688	2	13860
30	29	3	1	85	12	172	88	260	30	247	1	7854
31	30	4	4	85	12	229	169	398	39	414	3	1400
32	31	3	2	86	6	101	45	146	15	117	2	1876
33	32	2	4	87	34	661	132	793	23	698	3	2352

Рисунок 18 – Дані для розрахунків

Проєкт №3 виступатиме в нас у ролі «нового» проєкту і для нього ми рахуватимемо трудомісткість. А оскільки вона вже й відома, то ми спочатку порахуємо приблизну, а потім порівняємо її із реальною (рис 19):

Project	TeamExp	ManagerEx	YearEnd	Length	ransaction	Entities	ntsNonAd	djustmen	ointsAjus	Language	Effort
81	3	2	85	8	119	48	167	26	152	2	1617

Рисунок 19 – Дані щодо проєкту №3

Відстань рахуватиметься за евклідовою відстанню. Вона слугуватиме індикатором «наскільки наш проєкт схожий на вже відомі». Підрахунки робитимуться у Excel, бо це значно полегшує роботу із подібного роду даними (рис. 20).

Project	TeamExp	ManagerExp	YearEnd	Length	Transactions	Entities	PointsNonAdjust	Adjustment	PointsAjust	Language	Effort		Distance
1	2	2	88	3	126	49	175	38	180	3	651		32,68027
2	4	4	85	16	116	170	286	27	263	1	7252		203,5706
3	2	3	84	13	134	77	211	13	165	2	2275		58,00862
4	1	4	86	8	89	200	289	33	283	1	3983		236,8523
5	1	1	84	3	33	72	105	19	88	1	2282		126,4397
6	1	3	86	17	317	119	436	34	432	2	9135		441,759
7	4	1	85	14	175	277	452	37	461	1	3948		482,1214
8	2	3	87	27	173	332	505	19	424	1	14987		521,7384
9	1	3	86	6	42	31	73	27	67	2	1267		149,2783
10	2	1	84	17	146	112	258	40	271	1	9051		165,9669
11	0	0	86	4	140	94	234	24	208	1	2149		101,0099
12	3	1	85	14	148	324	472	39	491	1	4277		534,0122
13	2	4	85	18	88	170	258	34	255	1	5180		186,8154
14	2	-1	87	10	224	110	334	28	309	2	6783		259,644
15	2	3	85	8	106	39	145	6	103	1	2331		59,46427
16	3	4	87	14	9	386	395	21	340	2	4494		462,3127
17	1	4	88	4	158	59	217	18	180	3	847		70,76016
18	1	3	86	12	132	89	221	5	155	2	3647		72,33257
19	2	0	86	6	71	235	306	37	312	1	3542		286,9146
20	4	3	86	6	79	128	207	27	190	1	3927		105,119
21	3	4	87	10	101	57	158	9	117	2	1155		44,76606
22	4	5	86	26	482	227	709	26	645	2	9100		837,2258
23	4	4	86	14	68	316	384	20	326	2	3437		389,6948

Рисунок 20 – Розрахунок евклідової відстані

Наступним кроком буде вибір 3 проєктів із найменшою дистанцією до «нового» проєкту (рис. 21).

Project	TeamExp	ManagerExp	YearEnd	Length	Transactions	Entities	PointsNonAdjust	Adjustment	PointsAjust	Language	Effort	Distance
1	2	2	88	3	126	49	175	38	180	3	651	32,68027
54	2	1	85	9	119	42	161	25	145	2	2569	11,09054
59	1	4	87	8	124	52	176	14	139	2	2723	20,85665
81	3	2	85	8	119	48	167	26	152	2	1617	

Рисунок 21 – Вибір 3 проєктів із найменшою дистанцією до «нового» проєкту

Останнім кроком буде безпосередньо розрахунок трудомісткості проєкту №3. Воно буде дорівнювати середньому значенню трудомісткості вищезазначених проєктів (рис. 22).

Project	TeamExp	ManagerExp	YearEnd	Length	Transactions	Entities	PointsNonAdjust	Adjustment	PointsAjust	Language	Effort	Distance
1	2	2	88	3	126	49	175	38	180	3	651	32,68027
54	2	1	85	9	119	42	161	25	145	2	2569	11,09054
59	1	4	87	8	124	52	176	14	139	2	2723	20,85665
81	3	2	85	8	119	48	167	26	152	2	1617	1981

Рисунок 22 – Розрахунок трудомісткості проєкту №3

Як можна побачити, різниця незначна.

Але в завданні є умова про мову програмування. Один з наведених проєктів (під номером 1) не відповідає їй і менеджер вирішив не враховувати показники для оцінки «нового» проєкту.

Тобто для розробки нового проєкту можна враховувати оцінки тільки двох аналогій.

Далі навести порівнянн по атрибутам проєктів аналогій та «нового» проєкту.

Зауваження: чи достатньо двох проєктів для розробки на їх основі «нового» проєкту – вирішувати розробникам.

ЗАВДАННЯ

Використовуючи власні наробки, або веб сервіси для спільної розробки програмного забезпечення (наприклад GitHub), або проєктуючи за завданням викладача відповідні застосунки виконати наступне:

1. Вибрати проєкти та створити базу даних проєктів. Навести посилання на вибрані проєкти, для можливості викладачем перегляду проєктів і оцінювання вірності результатів що будуть отримані студентом під час виконання розрахунків.

2. Вибрати проєкт, який відкладаємо в сторону – він буде як «новий проєкт» що використовується для оцінки зусиль проєкту програмного забезпечення за аналогією.

3. Визначити необхідні для застосування методу аналогії атрибути (характеристики) «нового проєкту», не враховувати в ці атрибути size, Effort та мову програмування проєкту. Обчислити значення відповідних характеристик для всіх проєктів.

4. Внести всі дані у власний Database (табл. 18):

Таблиця 18 – Структура Database

Проєкти	githuburl	att1	att2	att3	…	Мова програмування	Size	Effort
project1								
project2								
…..								
project20								
project_new								

4. Застосовуючи одну або декілька метрик відстані, обчислити відстані усіх проєктів до «нового» проєкту, внести у свій data set. Виявити три найближчі проєкти. Навести розрахунки, найближчі проєкти позначити – наприклад іншим кольором.

5. Оцінити економічні показники нового проєкту. При оцінці враховувати мову програмування аналогічних проєктів (оскільки «новий» проєкт має розроблятися на певній мові і відповідно для застосування методу аналогій повинна бути вибрана ця мова або близька).

КОНТРОЛЬНІ ЗАПИТАННЯ

1. З яких груп складаються методи оцінки вартості програмного забезпечення.
2. В чому сутність неалгоритмічних методів?
3. В чому сутність методу Price-to-win?
4. В чому сутність методу оцінки за Паркінсоном?
5. В чому сутність методу експертної оцінки?
6. В чому сутність методу оцінки за аналогією?
7. В чому сутність методу Learning-Oriented Techniques?
8. В чому сутність методу Neural Networks?
9. В чому сутність методу Dynamics-Based Techniques?
10. Які існують метрики відстані?
11. Переваги та недоліки методу Price-to-win?
12. Які існують переваги та недоліки методу експертної оцінки?
13. Які існують переваги та недоліки методу оцінки за аналогією?
14. В чому сутність Дельфійської методики?

ПРАКТИЧНА РОБОТА №6

КОНСТРУКТИВНА МОДЕЛЬ ВАРТОСТІ COCOMO

Мета: навчитися використовувати інструменти за моделлю COCOMO для розрахунку економічних показників розробки програмного забезпечення

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Модель COCOMO (Constructive Cost Model) спрямовано на три моделі створення програмного забезпечення:

1) розповсюджений або органічний тип (organic projects);

2) напівнезалежний або напіврозподілений тип (semidetached projects);

3) вбудований тип (embedded projects).

Модель COCOMO поділяється на моделі: базовий (basic), проміжний (intermediate), деталізований (advanced).

Модель Basic COCOMO – двопараметрична. Як параметри виступають тип проєкту і обсяг програми (кількість рядків програмного 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 (KSLOC);

SS – середня чисельність персоналу;

P – продуктивність.

Коефіцієнти a_i, b_i, c_i, d_i вибираються з таблиці (див. таблицю 19).

Таблиця 19 –. Значення коефіцієнтів базового рівня моделі COCOMO залежно від типу (моделі) проєкту

Тип проєкту	a_i	b_i	c_i	d_i
Розповсюджений	2,4	1,05	2,5	0,38
Напівнезалежний	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 уточнена за рахунок введення додаткових 15 факторів витрат (Cost Drivers) (CD_k), які згруповані за чотирма категоріями:

1) Характеристики продукту (Product Attributes):

RELY –надійність (Required Software Reliability);

DATA – Розмір БД (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 проєкту (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 (табл. 20) відповідно до його ступеня значущості (рейтингу) в конкретному проєкті.

Таблиця 20 – Значення атрибутів вартості залежно від їх рівня

Атрибути вартості, CD_k	Рейтинг					
	Дуже низький	Низький	Середній	Високий	Дуже високий	Критичний
Характеристики продукту						
1. Необхідна надійність ПЗ	0,75	0,88	1,00	1,15	1,40	n/a
2. Розмір БД	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

Атрибути вартості, CD_k	Рейтинг					
	Дуже низький	Низький	Середній	Високий	Дуже високий	Критичний
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 проєкту						
13. Застосування методів розробки ПЗ	1,24	1,10	1,00	0,91	0,82	n/a
14. Використання інструментарію розробки ПЗ	1,24	1,10	1,00	0,91	0,83	n/a
15. Вимоги дотримання графіку розробки	1,23	1,08	1,00	1,04	1,10	n/a

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

Формула моделі проміжного рівня 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

(KSLOC).

EAF – (Effort Adjustment Factor) – добуток обраних атрибутів вартості з таблиці (див. табл. 2):

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

Коефіцієнти моделі a_i, b_i вибираються з таблиці 20.

Таблиця 21 – Значення коефіцієнтів проміжного рівня залежно від типу проєкту

Тип проєкту, 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.

У 2000 р. методика COCOMO була вдосконалена і отримала назву COCOMO II.

Розрізняють дві стадії оцінки проєкту:

1) попередня оцінка на початковій фазі (Early Design);

2) детальна оцінка після опрацювання архітектури (Post Architecture).

Для розрахунку трудомісткості необхідно спочатку оцінити фактори (чинники) масштабу (Scale Drivers) та множники трудомісткості (Cost Drivers або Effort Multipliers). Фактори масштабу застосовуються на двох стадіях оцінки проєкту. Множники трудомісткості відрізняються для різних стадій оцінки проєкту. 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 проєкту в люд. × міс. має вигляд:

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

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

де ;

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

$A = 2{,}45$ для детальної оцінки;

SF_j – фактори (чинники) масштабу (Scale Factors);

$SIZE$ – обсяг програмного продукту в тисячах рядків вихідного тексту (KSLOC);

EM_j – множники трудомісткості (Effort Multipliers). n=7 – для попередньої оцінки,

n=17 – для детальної оцінки;

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

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

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 22.

Таблиця 22 – Опис рівнів значущості факторів масштабу

SFj	Опис	Рівень значущості факторів					
		Дуже низький	Низький	Середній	Високий	Дуже високий	Критичний
1. PREC. Preceden-tedness	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 плат-форма 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. Architectu re / 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	переважно основному колективна робота	висока міра взаємодії	повна довіра, взаємозамі на і взаємодопо мога
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 Рівень 5

Зазначені фактори застосовуються на обох стадіях оцінки проєкту.

Числові значення фактора масштабу в залежності від оцінки його рівня, наведені в таблиці 23.

Таблиця 23 – 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 відрізняються для різних стадій оцінки проєкту:

1) Стадія попередньої оцінки трудомісткості програмного проєкту (Early Design).

Для цієї оцінки необхідно оцінити для проєкту рівень семи множників трудомісткості :

– 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 – нове застосування, інструменти і платформа; Extra High – застосування, 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 проєкта:

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. 23.

Таблиця 24 – 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 проєкту рівень сімнадцяти множників трудомісткості :

– параметри персоналу:

1) Analyst Capability (ACAP) – можливості аналітика;

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

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 проєкту:

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. 25.

Таблиця 25 – Значення множників трудомісткості

№	Множник зусиль	Дуже низький	Низький	Номінальний	Високий	Дуже високий	Екстра високий	Множник зусиль, EMJ
Кадрові фактори								
1	ACAP	Можливість аналітика	1,42	1,29	1,00	0,85	0,71	n/a
2	AEXP	Досвід застосування	1,22	1,10	1,00	0,88	0,81	n/a
3	PCAP	Можливості програміста	1,34	1,15	1,00	0,88	0,76	n/a
4	PCON	Неперервність персоналу	1,29	1,12	1,00	0,90	0,81	n/a
5	PEXP	Досвід платформи	1,19	1,09	1,00	0,91	0,85	n/a
6	LTEX	Досвід мови та інструментів	1,20	1,09	1,00	0,91	0,84	n/a
	Фактори продукту							
7	RELY	Необхідна надійність програмного забезпечення	0,84	0,92	1,00	1,10	1,26	n/a
8	DATA	Розмір бази даних	n/a	0,23	1,00	1,14	1,28	n/a
9	CPLX	Складність програмного продукту	0,73	0,87	1,00	1,17	1,34	1,74
10	RUSE	Необхідна можливість повторного використання	n/a	0,95	1,00	1,07	1,15	1,24
11	DOCU	Відповідність документації	0,81	0,91	1,00	1,11	1,23	n/a
	Фактори платформи							
12	TIME	Обмеження часу виконання	n/a	n/a	1,00	1,11	1,29	1,63
13	STOR	Основне обмеження зберігання	n/a	n/a	1,00	1,05	1,17	1,46

№	Множник зусиль	Дуже низький	Низький	Номінальний	Високий	Дуже високий	Екстра високий	Множник зусиль, EMJ
14	PVOL	Волатильність платформи	n/a	0,87	1,00	1,15	1,30	n/a
	Фактори проєкту							
15	TOOL	Використання програмних засобів	1,17	1,09	1,00	0,90	0,78	n/a
17	SITE	Багатосайтовий розвиток	1,22	1,09	1,00	0,93	0,86	0,80
16	SCED	Необхідний графік розробки	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 проєкту або час розробки проєкту TM в 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 проєкту без урахування множника,

SCED що визначає ущільнення розкладу.

Програмні застосунки для використання моделі COCOMO

Для виконання розрахунків з використання моделі COCOMO можна використовувати інструменти, такі як COCOMO Suite of Constructive Cost Models. Даний інструмент доступний за посиланням:

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

Розглянемо приклад роботи онлайн-калькулятора (рис. 23).

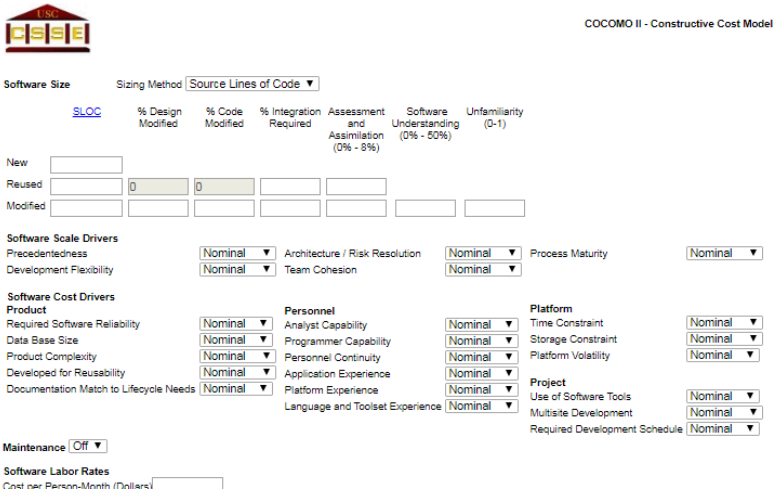


Рисунок 23 – Інтерфейс онлайн калькулятора

На сайті, що доступний за наведеним нижче посиланням знаходяться посилання на інші доступні інструменти по розрахунку за моделями COCOMO, що розроблені в USC CSSE.

https://csse.usc.edu/csse/tools/

Також для розрахунків можна застосовувати інструмент Costar (рис. 24), який розроблено компанією SoftStar на моделі COCOMO II для автоматизації оцінки вартості розробки програмних продуктів. Завантажити програму можна за посиланням: http://www.softstarsystems.com/

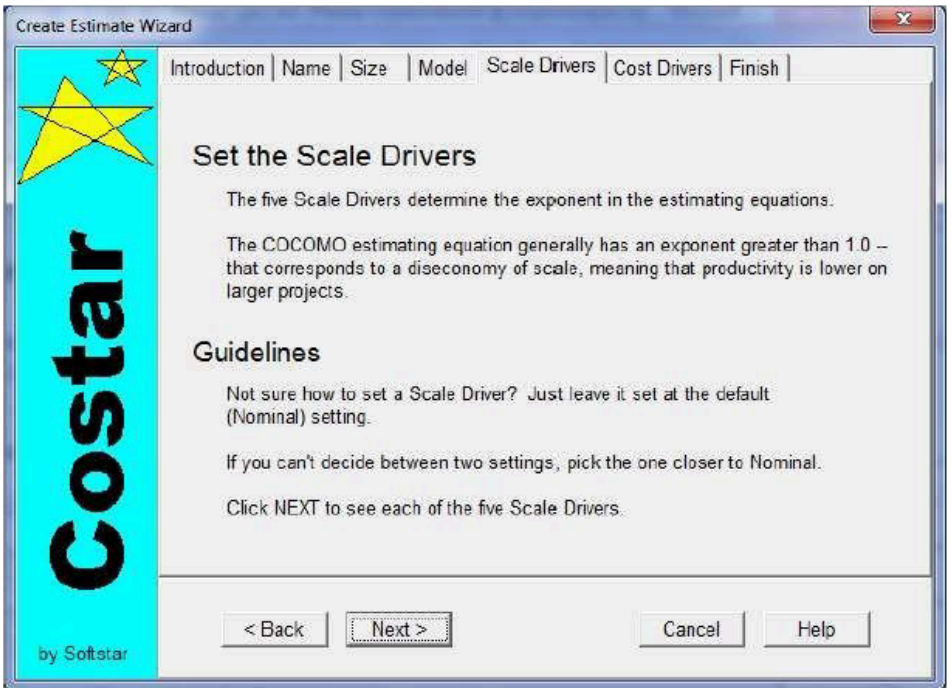


Рисунок 24 – Налаштування 5 факторів масштабу у інструменті Coststar.

Приклад виконання

Програмний застосунок, що виконує роль приклада, являє собою об'єктно-об'єктний маппер у .NET, що базується на конвенціях (рис. 25).

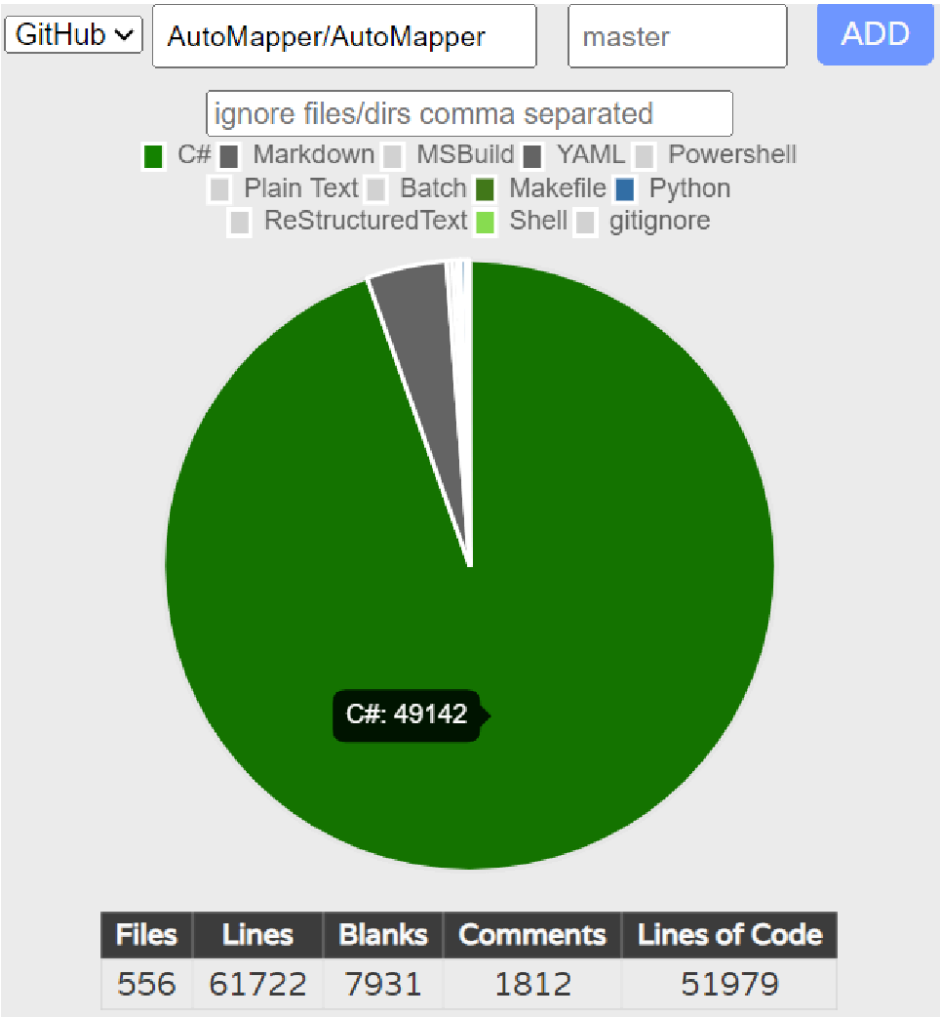


Рисунок 25 – Приклад роботи програмного застосунка

Це бібліотека, створена для вирішення оманливо складної проблеми - позбавлення від коду, який зіставляє один об'єкт з іншим. Розмір проєкту: 51979 рядків програмного коду.

Базова модель COCOMO: AutoMapper - це об'єктно-об'єктний маппер на основі конвенцій. Він розроблений для сценаріїв проекції моделей, щоб згладити складні моделі об'єктів до DTO та інших простих об'єктів. З огляду на його складність і досвід, необхідний для розробки такого проєкту, він, швидше за все, підпадає під категорію "Semi-Detached" в моделі COCOMO.

Розрахунки:

$$PM = a * (SIZE)^b = 3 * 51{,}979^{1{,}12} = 250{,}52$$

$$TM = c * (PM)^d = 2{,}5 * (250{,}52)^{0{,}35} = 17{,}28$$

$$SS = \frac{PM}{TM} = \frac{250{,}52}{17{,}28} = 14{,}5$$

$$P = \frac{SIZE}{PM} = \frac{51{,}979}{250{,}52} = 0{,}2075$$

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 (KSLOC);

SS – середня чисельність персоналу;

P – продуктивність.

Проміжна модель COCOMO:

Вибір значень атрибутів вартості з поясненням – нижче у таблиці 26:

Таблиця 26 – Вибір значень атрибутів вартості

Атрибути вартості, CDk	Рейтинг	Причина
Характеристики продукту		
1. Необхідна надійність ПЗ	Високий (1,15)	AutoMapper є широко використовуваною бібліотекою, тому надійність має вирішальне значення.
2. Розмір БД додатка	Дуже низький (n/a)	AutoMapper не має бази даних.
3. Складність продукту	Високий (1,15)	AutoMapper використовує алгоритм зіставлення на основі

Атрибути вартості, CDk	Рейтинг	Причина
		конвенцій для зіставлення значень джерела і призначення, що додає йому складності.
Характеристики апаратного забезпечення		
4. Обмеження швидкодії при виконанні програми	Середній (1)	AutoMapper розроблено ефективним, але його продуктивність може залежати від того, як він використовується у програмах
5. Обмеження пам'яті	Середній (1)	AutoMapper сам по собі не має великого розміру, але використання пам'яті може залежати від розміру та складності об'єктів, що зіставляються
6. Нестійкість оточення віртуальної машини	Дуже низький (n/a)	AutoMapper не потребує віртуальної машини
7. Необхідний час відновлення	Середній (1)	Хоча AutoMapper не працює у режимі реального часу, продуктивність все одно важлива
Характеристики персоналу		
8. Аналітичні здібності	Високий (0,86)	Розробка AutoMapper або участь у ньому вимагає хороших аналітичних навичок для розуміння об'єкт-об'єктного зіставлення та конвенцій
9. Досвід розробки	Високий (0,91)	Досвід роботи зі сценаріями мапування має вирішальне значення
10. Здібності до розробки ПЗ	Високий (0,86)	AutoMapper є добре розробленим проєктом
11. Досвід використання віртуальних машин	Дуже низький (1,21)	AutoMapper не вимагає використання віртуальних машин

Атрибути вартості, CDk	Рейтинг	Причина
12. Досвід розробки на мовах програмування	Високий (0,95)	AutoMapper написаний на C#, тому досвід роботи з цією мовою є важливим.
Характеристики проєкту		
13. Застосування методів розробки ПЗ	Високий (0,91)	Враховуючи складність проєкту, швидше за все, будуть використані формальні методи
14. Використання інструментарію розробки ПЗ	Середній (1)	Буде використано стандартні засоби розробки програмного забезпечення, такі як IDE та контроль версій
15. Вимоги дотримання графіку розробки	Середній (1)	Оскільки це проєкт з відкритим вихідним кодом, розробка може не бути строго запланованою

Розрахунки було здійснено за наступною формулою:

$$EAF = \prod_{k=1}^{15} CD_k = 1{,}172$$

$$PM = EAF * a * (SIZE)^b = 1{,}172 * 3 * 51{,}979^{1{,}12} = 293{,}61$$

$$TM = c * (PM)^d = 2{,}5 * (293{,}61)^{0{,}35} = 18{,}27$$

Де:

EAF (Effort Adjustment Factor) – добуток обраних атрибутів вартості

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

TM (Time at Month) – час розробки в календарних місяцях;

SIZE – обсяг програмного продукту в тисячах рядків вихідного тексту (KSLOC);

COCOMO II попередня та детальна оцінка:

Фактори масштабу у табличному вигляді (табл. 27):

Таблиця 27 – Фактори масштабу

Чинник масштабу	Значення	Причина
PREC (Precedentedness)	Very High (1,24)	AutoMapper - це добре відпрацьована, широко використовувана бібліотека. Можна з упевненістю припустити, що команда має значний досвід роботи з продуктом і платформою.
FLEX (Development Flexibility)	High (2,03)	Враховуючи, що AutoMapper має відкритий вихідний код і внесок від різних розробників, процес розробки, ймовірно, є гнучким. Не маючи конкретних знань про внутрішні процеси, можна зробити консервативну оцінку.
RESL (Architecture/Risk Resolution)	High (2,83)	AutoMapper є зрілим проєктом, що свідчить про те, що більшість архітектурних ризиків було враховано. Однак, не маючи конкретних даних аналізу ризиків, ми робимо консервативну оцінку.
TEAM (Team Cohesion)	Nominal (3,29)	Оскільки це проєкт з відкритим вихідним кодом, згуртованість команди може змінюватися. Учасники можуть не взаємодіяти так тісно, як команда, що знаходиться в одному приміщенні.
PMAT (Process Maturity)	Nominal (4,68)	Без конкретних знань про рівень їхньої моделі зрілості можливостей (CMM) важко надати точну оцінку. З огляду на очевидний успіх і широке використання проєкту, ми робимо консервативну оцінку.

Множники трудомісткості у табличному вигляді (попередня оцінка) наведено у табл. 28:

Таблиця 28 – Множники трудомісткості (попередня оцінка)

Множник трудомісткості	Значення	Причина
PERS (Personnel Capability)	High (0,83)	AutoMapper - це добре налагоджений проєкт з великою спільнотою учасників. Можна з упевненістю припустити, що учасники мають високу кваліфікацію.

Множник трудомісткості	Значення	Причина
PREX (Personnel Experience)	Very High (0,74)	Враховуючи, що AutoMapper є зрілим проєктом, цілком ймовірно, що учасники добре знайомі з програмою, інструментами та платформою.
RCPX (Product Reliability and Complexity)	High (1,33)	AutoMapper є широко використовуваною бібліотекою, що свідчить про її високу надійність. Крім того, враховуючи її функціональність, вона, ймовірно, має помірний рівень складності.
RUSE (Developed for Reusability)	High (1,07)	AutoMapper призначено для багаторазового використання у різних проєктах, тож цілком ймовірно, що її було розроблено з думкою про багаторазове використання.
PDIF (Platform Difficulty)	Low (0,87)	AutoMapper є бібліотекою .NET, а .NET є стабільною платформою без серйозних обмежень на пам'ять і продуктивність.
FCIL (Facilities)	High (0,87)	Враховуючи, що AutoMapper є проєктом з відкритим вихідним кодом, цілком ймовірно, що учасники використовують різноманітні інструменти і мають хороші канали зв'язку.
SCED (Required Development Schedule)	Nominal (1)	Не маючи конкретних знань про графік їхньої розробки, важко надати точну оцінку. Номінальне значення використовується як консервативна оцінка.

Множники трудомісткості (детальна оцінка) наведено у таблиці 29:

Таблиця 29 – Множники трудомісткості (детальна оцінка)

Множник трудомісткості	Значення	Причина
Analyst Capability (ACAP)	High (0,85)	Зважаючи на складність проєкту AutoMapper, цілком ймовірно, що в ньому беруть участь висококваліфіковані аналітики.
Applications Experience (AEXP)	High (0,88)	AutoMapper є зрілим проєктом, що свідчить про те, що команда має значний досвід розробки додатків.
Programmer Capability (PCAP)	High (0,88)	Якість коду та широке використання бібліотеки свідчать про високий рівень здібностей програмістів.
Personnel Continuity (PCON)	Nominal (1)	Не маючи конкретної інформації про плинність кадрів в команді, найбезпечніше припустити номінальну оцінку.
Platform Experience (PEXP)	High (0,91)	Проєкт побудований на .NET, і команда, схоже, добре володіє цією платформою.
Language and Tool Experience (LTEX)	High (0,91)	Проєкт написаний на C#, і досвід команди в цій мові видно з кодової бази.
Required Software Reliability (RELY)	Nominal (1)	Бібліотека широко використовується, що свідчить про необхідність надійної роботи, але це не та сфера, де надзвичайна надійність є критично важливою.
Database Size (DATA)	Very Low (n/a)	AutoMapper не орієнтована на роботу з базами даних; вона в першу чергу призначена для зіставлення об'єктів між собою.
Software Product Complexity (CPLX)	High (1,17)	AutoMapper вирішує складну задачу - зіставлення між об'єктами, що передбачає роботу з різними сценаріями.
Required Reusability (RUSE)	High (1,07)	Так як бібліотека призначена для використання в інших проєктах,

Множник трудомісткості	Значення	Причина
		потрібна висока придатність до повторного використання.
Documentation Match to Life Cycle Needs (DOCU)	High (1,11)	Проєкт має велику документацію.
Execution Time Constraint (TIME)	Nominal (1)	Хоча продуктивність є важливою, немає жодних доказів екстремальних обмежень часу виконання.
Main Storage Constraint (STOR)	Nominal (1)	AutoMapper не є вимогливим до обсягу пам'яті.
Platform Volatility (PVOL)	Low (0,87)	Платформа .NET є зрілою та стабільною.
Use of Software Tools (TOOL)	Nominal (1)	AutoMapper використовує стандартний набір програмних інструментів, типовий для проєктів такого типу.
Multisite Development (SITE)	Nominal (1)	Не маючи конкретної інформації про місця розробки, можна припустити номінальну оцінку.
Required Development Schedule (SCED)	Nominal (1)	Немає жодних доказів екстремального тиску на графік.

Розрахунки для попередньої оцінки:

$$E = B + 0{,}01 * \sum_{j=1}^{5} SF_j = 1{,}0507$$

$$EAF = \prod_{k=1}^{7} EM_k = 0{,}6616$$

$$PM = EAF * A * (SIZE)^E = 0{,}6616 * 2{,}94 * 51{,}979^{1{,}0507} = 123{,}53$$

$$TM = SCED * C * (PM_{NS})^{D+0{,}2*(E-B)} = 1 * 3{,}67 * (123{,}53)^{0{,}28+0{,}2*(1{,}0507-0{,}91)} = 16{,}189$$

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 (KSLOC);

Розрахунки для детальної оцінки:

$$E = B + 0{,}01 * \sum_{j=1}^{5} SF_j = 1{,}0507$$

$$EAF = \prod_{k=1}^{17} EM_k = 0{,}659$$

$$PM = EAF * A * (SIZE)^E = 0{,}659 * 2{,}45 * 51{,}979^{1{,}0507} = 102{,}535$$

$$TM = SCED * C * (PM_{NS})^{D+0{,}2*(E-B)} = 1 * 3{,}67 * (102{,}535)^{0{,}28+0{,}2*(1{,}0507-0{,}91)} = 15{,}286$$

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 (KSLOC);

Дослідити вплив розміру програмного коду (SIZE) на трудомісткість (PM) та час розробки проєкту (TM) для різних моделей COCOMO II:

Оскільки досліджуваний проєкт мав трохи більше 50 000 рядків програмного коду, зробимо розрахунки для розмірів 25000, 75000 та 100 000:

Для цих розрахунків:

$$E = B + 0{,}01 * \sum_{j=1}^{5} SF_j = 1{,}0507$$

Для попередньої оцінки:

$$EAF = \prod_{k=1}^{7} EM_k = 0{,}6616$$

Для детальної оцінки:

$$EAF = \prod_{k=1}^{17} EM_k = 0{,}659$$

Розмір 25 000 рядків програмного коду:

Попередня оцінка:

$$PM = EAF * A * (SIZE)^E = 0{,}6616 * 2{,}94 * 25^{1{,}0507} = 57{,}248$$

$$TM = SCED * C * (PM_{NS})^{D+0{,}2*(E-B)} = 1 * 3{,}67 * (57{,}248)^{0{,}28+0{,}2*(1{,}0507-0{,}91)} = 16{,}189$$

Детальна оцінка:

$$PM = EAF * A * (SIZE)^E = 0{,}659 * 2{,}45 * 25^{1{,}0507} = 47{,}519$$

$$TM = SCED * C * (PM_{NS})^{D+0{,}2*(E-B)} = 1 * 3{,}67 * (47{,}519)^{0{,}28+0{,}2*(1{,}0507-0{,}91)} = 15{,}286$$

Розмір 75 000 рядків програмного коду:

Попередня оцінка:

$$PM = EAF * A * (SIZE)^E = 0{,}6616 * 2{,}94 * 75^{1{,}0507} = 181{,}581$$

$$TM = SCED * C * (PM_{NS})^{D+0{,}2*(E-B)} = 1 * 3{,}67 * (181{,}581)^{0{,}28+0{,}2*(1{,}0507-0{,}91)} = 18{,}23$$

Детальна оцінка:

$$PM = EAF * A * (SIZE)^E = 0{,}659 * 2{,}45 * 75^{1{,}0507} = 150{,}726$$

$$TM = SCED * C * (PM_{NS})^{D+0{,}2*(E-B)} = 1 * 3{,}67 * (150{,}726)^{0{,}28+0{,}2*(1{,}0507-0{,}91)} = 17{,}213$$

Розмір 100 тисяч:

Попередня оцінка:

$$PM = EAF * A * (SIZE)^E = 0{,}6616 * 2{,}94 * 100^{1{,}0507} = 245{,}665$$

$$TM = SCED * C * (PM_{NS})^{D+0{,}2*(E-B)} = 1 * 3{,}67 * (245{,}665)^{0{,}28+0{,}2*(1{,}0507-0{,}91)} = 20{,}01$$

Детальна оцінка:

$$PM = EAF * A * (SIZE)^E = 0{,}659 * 2{,}45 * 100^{1{,}0507} = 203{,}916$$

$$TM = SCED * C * (PM_{NS})^{D+0{,}2*(E-B)} = 1 * 3{,}67 * (203{,}916)^{0{,}28+0{,}2*(1{,}0507-0{,}91)} = 18{,}893$$

Отримати значення PM та TM по всім моделям для одного й того ж значення параметра SIZE, обравши номінальний (середній) рівень складності проєкту, що має високу ступінь новизни.

Проміжна COCOMO, розмір – 100 000 рядків програмного коду (рис. 26):

Product Attributes	
Required Reliability	1.15 (H)
Database Size	1.00 (N)
Product Complexity	1.00 (N)
Computer Attributes	
Execution Time Constraint	1.00 (N)
Main Storage Constraint	1.00 (N)
Platform Volatility	1.00 (N)
Computer Turnaround Time	1.00 (N)
Personnel Attributes	
Analyst Capability	0.86 (H)
Applications Experience	0.91 (H)
Programmer Capability	0.86 (H)
Platform Experience	0.90 (H)
Programming Language and Tool Experience	0.95 (H)
Project Attributes	
Modern Programming Practices	1.10 (L)
Use of Software Tools	1.10 (L)
Required Development Schedule	1.00 (N)

Рис. 26. Параметри проєкту

Результат виконаних розрахунків представлено на рисунку 27:

COCOMO RESULTS for Semi-detached 100								
MODE	"A" variable	"B" variable	"C" variable	"D" variable	KLOC	EFFORT, (in person-months)	DURATION, (in months)	STAFFING, (recommended)
semi-detached	2.4021984086100003	1.12	2.5	0.35	100.000	417.454	20.661	20.205

Рисунок 27 – Результат виконаних розрахунків

COCOMO II, розмір – 100 000 рядків програмного коду.

Параметри представлено на рисунку 28:

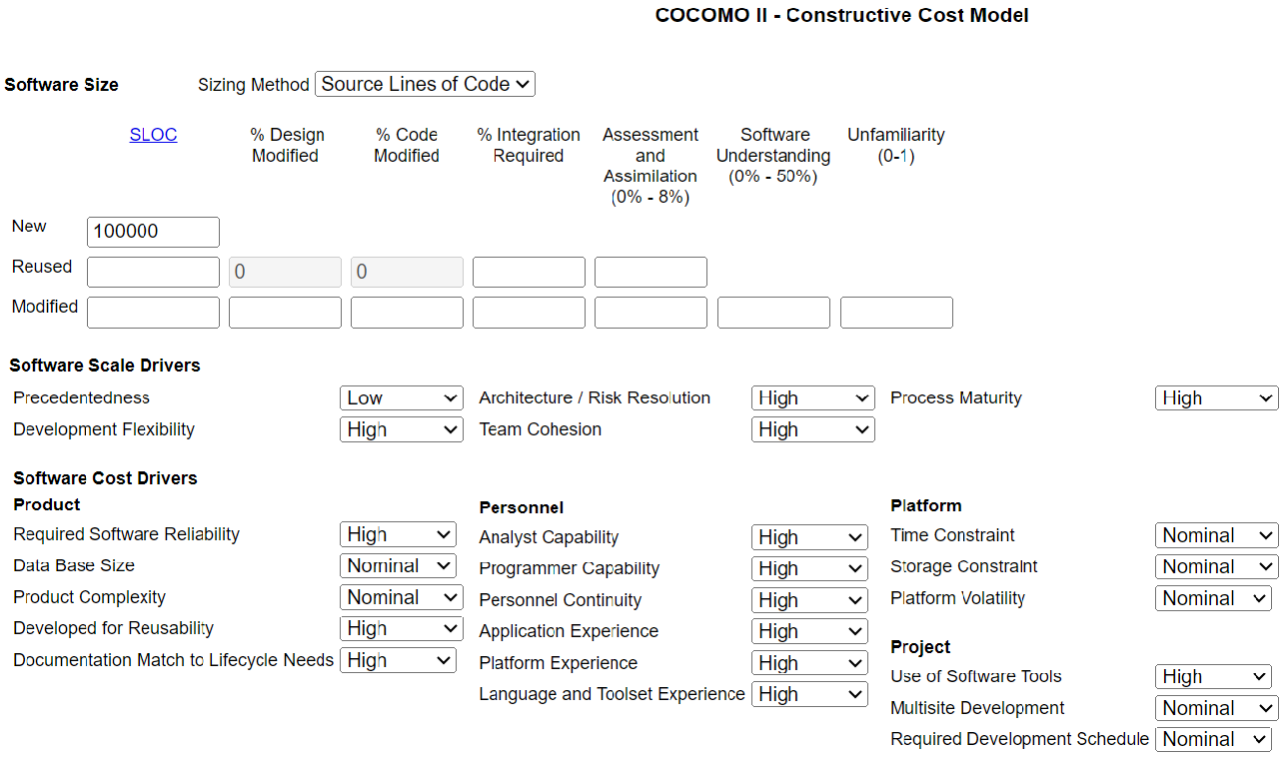


Рисунок 28 – Параметри проєкту, що містить більше 100 000 рядків програмного коду

Результат проведених розрахунків:

Effort = 224.9 Person-months
Schedule = 19.7 Months

Таким чином, було проведено дослідження трудомісткості розробки програмного продукту за допомогою моделей COCOMO та COCOMO II. Було встановлено, що розмір програмного коду (SIZE) має значний вплив на трудомісткість (PM) та час розробки проєкту (TM) для різних моделей COCOMO II [39].

ЗАВДАННЯ

1. Розрахувати трудомісткість розробки програмного застосунку використовуючи за базовою та проміжною моделями COCOMO. Для виконання роботи брати проєкти, що містять більше 25000 рядків коду.

2. Проаналізувати програмний застосунок на основі моделі COCOMO II (попередня та детальна оцінка).

3. Дослідити вплив розміру програмного коду (SIZE) на трудомісткість (PM) та час розробки проєкту (TM) для різних моделей COCOMO II.

4. Отримати значення PM та TM по всім моделям для одного й того ж значення параметра SIZE, обравши номінальний (середній) рівень складності проєкту, що має високу ступінь новизни.

5. Обов'язково навести проведені розрахунки з поясненням вибору всіх параметрів. Якщо параметр не використовувався (або дорівнює нулю) – вказати причину невикористання.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте та порівняйте моделі COCOMO та COCOMO II.
2. Які існують рівні створення продукту моделі COCOMO?
3. Які характеристики мають organic projects моделі COCOMO?
4. Які характеристики мають semidetached projects моделі COCOMO?
5. Які характеристики мають embedded projects моделі COCOMO?
6. Назвіть рівняння базового рівня моделі COCOMO.
7. Назвіть характеристики продуктів проміжного рівня моделі COCOMO?
8. Які характеристики персоналу проміжного рівня моделі COCOMO?
9. Якою є роль Effort Adjustment Factor?
10. Які існують стадії оцінки проєкту у моделі COCOMO II.
11. Дайте визначення Scale Drivers, Effort Multipliers.
12. Чи відрізняються фактори масштабу для різних стадій оцінки проєкту?
13. Які множники трудомісткості необхідно оцінити для стадії попередньої оцінки проєкту?
14. Які множники трудомісткості необхідно оцінити для стадії детальної оцінки проєкту?

ПРАКТИЧНА РОБОТА №7

ОЦІНКА РОЗМІРУ (ЕКОНОМІЧНИХ ПОКАЗНИКІВ) В AGILE МЕТОДАХ (НА ПРИКЛАДІ SCRUM)

Мета: ознайомитися та навчитися використовувати методи для розрахунку економічних показників розробки програмного забезпечення при застосуванні Agile методів.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Методологію Agile характеризує маніфест гнучких принципів. Принципи манифесту AGILE:

1. Найвищим пріоритетом є задоволення клієнта шляхом ранньої та постійної доставки програмного забезпечення.

2. Зміни вимог, навіть на пізній стадії розробки. Agile процеси обробляють зміни, щоб забезпечити конкурентну перевагу замовника.

3. Поставляти робоче програмне забезпечення часто, від декількох тижнів до кількох місяців, з перевагою щодо скорочення часу.

4. Замовник та розробники повинні щодня працювати разом протягом усього проєкту.

5. Будуйте проєкти навколо мотивованих людей. Забезпечте їм необхідне середовище та підтримку та довіряйте їм виконувати роботу.

6. Найефективнішим методом передачі інформації до команди розробників та в рамках команди розробників є особиста розмова.

7. Робоче програмне забезпечення є основним показником прогресу.

8. Agile процеси сприяють сталому розвитку проєкту. Спонсори, розробники та користувачі повинні підтримувати постійний темп.

9. Постійна увага до технічної досконалості та гарного дизайну підвищує гнучкість.

10. Простота - мистецтво максимізувати обсяг роботи, яку не потрібно виконувати - має важливе значення.

11. Найкращі архітектури, вимоги та проєкти притаманні командам, що самоорганізовані.

12. Через рівні проміжки часу команда розмірковує над тим, як стати більш ефективною, а потім відповідно налаштовує та коригує свою поведінку.

Agile методологія містить декілька методів. До таких методів належать, наприклад, наступні: Dynamic System Development Method, Adaptive Software Development, Crystal Clear, Scrum, XP, Feature-Driven Development, Agile Unified Process. Серед найбільш розповсюджених є Scrum. На рис. 29 представлено Scrum Process.

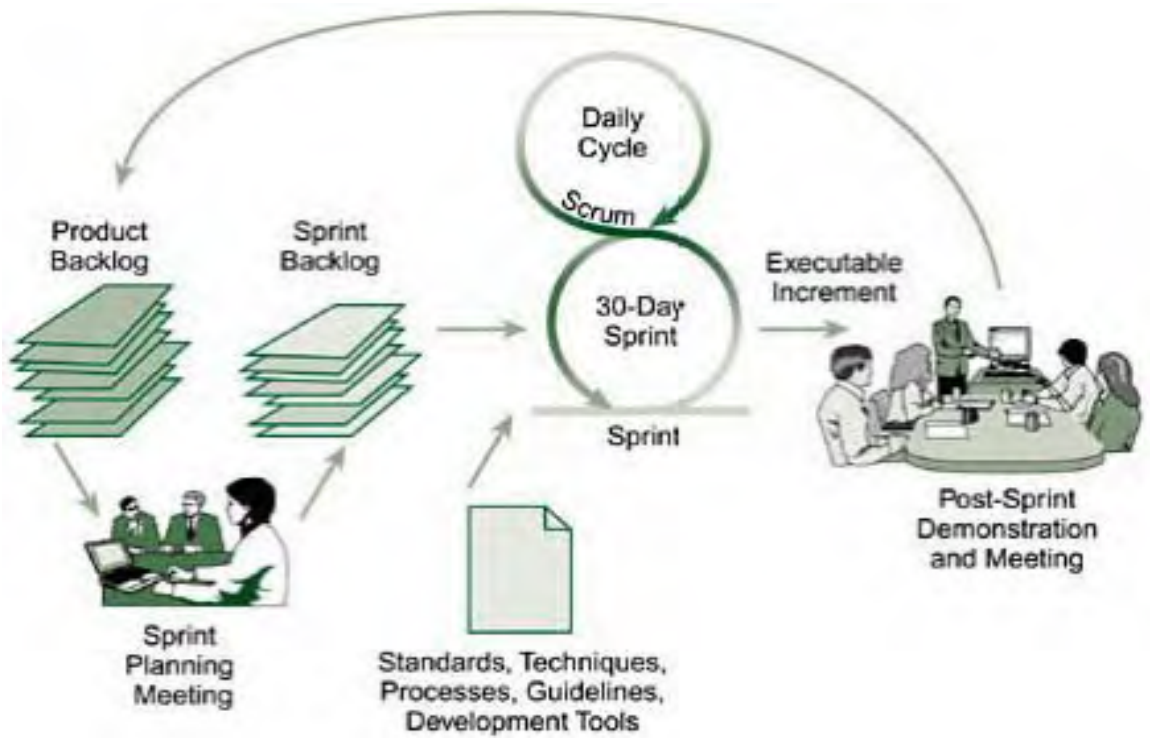


Рисунок 29 – Процеси відповідно до SCRUM

Термінологія Scrum

Команда Scrum. Типова команда Scrum налічує від п'яти до дев'яти людей, але проєкти Scrum можуть легко масштабуватися до сотень. Однак Scrum може бути легко використаний командами з однієї особи. Команда не включає жодної з традиційних ролей, таких як програміст, тестувальник або архітектор. Кожен, хто бере участь у проєкті, працює разом, щоб виконати ту роботу, яку вони колективно взяли. Команди Scrum формують глибоку форму товариськості та відчуття, що «ми всі в цьому разом».

Власник продукту (Product owner). Ключова зацікавлена сторона проєкту і представляє користувачів, клієнтів та інших учасників процесу. Власником

продукту часто є хтось із керівництва продуктом або маркетингу, ключова зацікавлена сторона або ключовий користувач.

Scrum Master. Відповідає за те, щоб команда була максимально продуктивною. Майстер Scrum робить це, допомагаючи команді використовувати процес Scrum, усуваючи перешкоди для прогресу, захищаючи команду ззовні тощо.

Product Backlog. Список пріоритетних функцій, що містить усі бажані функції або зміни продукту. Примітка. Термін "відставання" може заплутати, оскільки він використовується для двох різних речей. Для уточнення, відставання товару - це список бажаних функцій продукту.

Sprint Backlog. Список завдань, які потрібно виконати в спринті. На початку кожного спринту команда вибирає певний обсяг роботи із Product Backlog та зобов’язується завершити цю роботу під час спринту. Частиною з’ясування того, які вони можуть взяти на себе зобов’язання, є створення Sprint backlog, тобто перелік завдань (та оцінка того, скільки часу триватиме кожне з них), необхідних для доставки вибраного набору елементів Product Backlog, які потрібно виконати у спринті.

В кінці кожного спринту команда робить потенційно можливий приріст функціональності продукту - тобто працююче високоякісне програмне забезпечення. Щодня під час спринту члени команди збираються, щоб обговорити свій прогрес та будь-які перешкоди для завершення роботи для цього спринту. Це відомо як Daily Scrum.

Планувальна нарада (Planning Meeting). На початку кожного спринту проводиться зустріч з планування спринту, під час якої власник продукту (owner) представляє команді пріоритетні елементи з продукту (Product Backlog). Команда Scrum обирає роботу, яку може виконати під час наступного спринту. Потім ця робота переноситься з Product Backlog до Sprint Backlog, що є переліком завдань, які команда зобов'язується виконати у спринті (рис. 30).

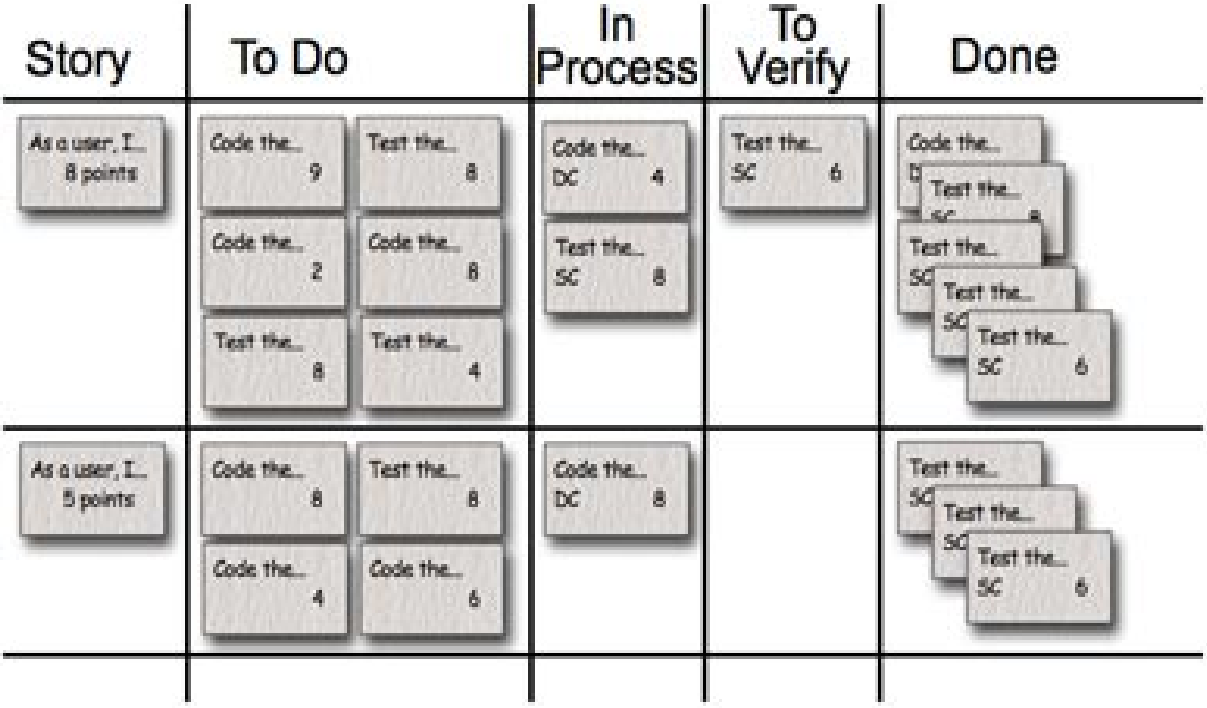


Рисунок 30 – Дошка SCRUM

Щоденна Scrum (Daily Scrum). Кожного дня під час спринту проводиться коротка зустріч, яка називається Daily Scrum. Ця зустріч допомагає встановити контекст роботи кожного дня та допомагає команді залишатися на шляху. Усі члени команди повинні відвідувати Daily Scrum.

Зустріч з огляду спринту (Sprint review meeting). Наприкінці кожного спринту команда демонструє завершену функціональність на оглядовому засіданні спринту, під час якого команда показує, що вони досягли під час спринту. Як правило, це приймає форму демонстрації нових можливостей, але в неформальній формі.

Ретроспектива спринту (Sprint retrospective). В кінці кожного спринту команда проводить ретроспективу спринту, це зустріч, під час якої команда (включаючи свого керівника Scrum та власника продукту) розмірковує про те, наскільки добре Scrum працює для них і які зміни вони можуть побажати щоб це працювало ще краще.

Методика розв’язання типових задач

(SCRUM Estimation Methods)

1. Planning Poker. Planning Poker є широко використовуваним методом. JIRA має для керування проєктами Agile Poker. Планування Покер — це гнучка техніка оцінки, яка встановлює відносне визначення розміру в story points і гральних карток із позначеннями 0,1,2,3,5,8,13,21,34 і 55. Ці значення гральних карт представляють story points для певного item/story, за якими вимірюється команда на scrum meeting команди. Числова послідовність зазвичай послідовність Фібоначчі. Product owner або Scrum Master читає story користувача, пояснюючи всі властивості та вимоги і обговорюючи з командою усі технічні та нетехнічні вимоги для оцінок (рис. 31).

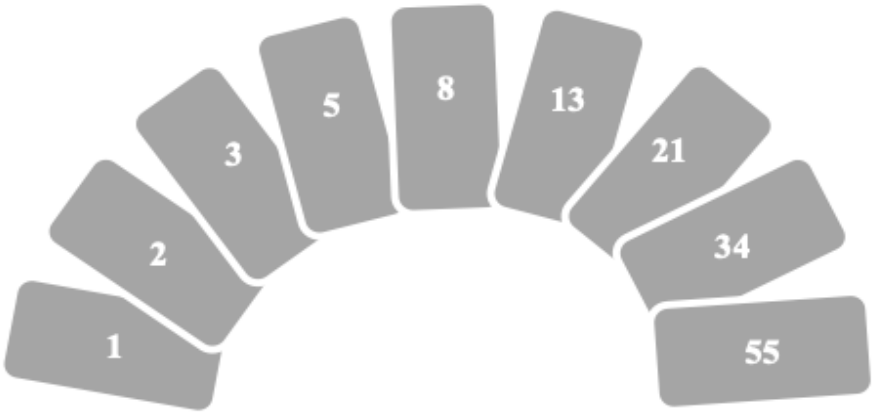


Рисунок 31 – Planning Poker

Після детального аналізу та обговорення item/story, всіх оцінювачів просять вибрати одну картку, щоб оцінити story користувача. все оцінювачі вибирає еквівалентне значення з карт, що стає остаточною оцінкою. Якщо значення є різні, то модератори просять пояснення щодо оцінювачів, чому вони надали такі значення (враховуючи високі та низькі значення). Обговорюють вимогу до досягнення консенсусу в команді.

2. T-Shirt Size. В моделі для для приблизних оцінок використовуються розміри XS (Extra Small), S (Small), M (середній), L (великий), XL (дуже великий). Модель спрямовано зробити грубу оцінку проєкту дуже швидко. Встановити розмір (переважно середній) можливо після спільного обговорення в команді і узгоджуючи значення, присвоєне вимозі відповідно до середнього розміру (рис. 32).

Рисунок 32 – Модель приблизних оцінок

3. Bucket System. Bucket System (рис. 33), використовується для більшої кількості елементів, які потрібно оцінити а більша кількість команд. Є Bucket (множина переліків для того, що потрібно робить).

Рисунок 33 – Bucket System

Кожен перелік (Bucket) різного відносного розміру, наприклад, з розмірами 1, 2, 3, 4, 5, 8, 13, 20, 30, 50, 100, 200. Оцінювач підбирає task/story з backlog та помістити його в будь-яке відповідне поле Bucket, зазначене вище. При цьому оцінювач пояснює, чому це вміщується в цей Bucket. Дія повторюється докі backlog не буде завершено. Scrum Master перевіряє, що ніхто

не переміщує task/story, поки оцінки не будет зроблено. Bucket System є одним із методів спільної оцінки техніки і швидко.

4. Dot Voting. Метод Dot Voting (рис. 34) - це рейтинг між історією найвищого пріоритету та історіями найнижчого пріоритету з найбільш основних завдань/історій Product Backlog, які повинні розпочатися першими. Scrum board використовується для наклейки всіх історій користувачів з жовтими наліпками та їх описом. Голосування проводиться між зацікавленими сторонами на Scrum Meeting для кожної з вимог для визначення пріоритетних вимог. Кожен учасник надає свій голос за вимогу (позитивна/негативна). Власник продукту переставляє елементи Product Backlog відповідно до кількості голосів, отриманих історією. Після завершення пріоритизації Product Backlog поділяється на 3 групи, які називаються групами високого, середнього та низького пріоритету. Дії повторюються для кожної групи до досягнення узгодження. Цей метод простий і швидкий і ефективно оцінює невелику кількість історій (до 8-10).

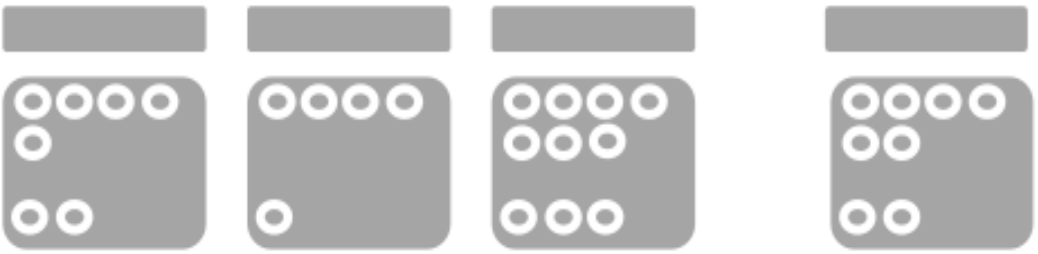


Рисунок 34 – Dot Voting

5. Large / Uncertain / Small. Існує лише три заздалегідь визначені розміри. Команда або оцінювачі оцінюють item/story у будь-яку категорію під назвою «Великий», «Невизначений» і Маленький. Цей метод оцінки використовується, коли є подібні item/story у backlog продукту. Цей спосіб схожий на Проста / Середня / Складна модель оцінок. Для кожної категорії, призначається кількість Story points, яка пов’язується з item/story.

6. Ordering Method. Цей метод оцінки підходить, коли є велика кількість елементів у Product Backlog і невелика кількість ресурсів. Випадковим чином визначаються та розміщуються порядкові шкали на діапазонах пріоритетів від низького до високого. Кожен член команди або оцінювач переміщує item з однієї шкали в іншу. Розміщенні та пріоретизовані items/stories, переміщуються вгору або донизу на одне місце, поки всі учасники команди не будуть задоволені з розміщенням у Backlog. Таким чином забезпечується пріоритетний порядок item у product backlog

7. Divide Until Maximum Size or Less. Команда Scrum визначає базове значення як максимальний розмір item для оцінки (наприклад, 8 годин зусиль) у проєкті. Кожне item у Product Backlog обговорюється щоб визначити, чи є кожен item більшим або меншим відносно оціненої базової вартості. Якщо item перевищує максимальний визначений розмір, він розбивається на кілька item, що відповідають оцінці розміру базового значення. Діяльність триває до тих пір, поки не буде розглянуто весь Product Backlog, а items знаходяться в визначеному базовому значенні.

ЗАВДАННЯ

1. Створити команди та обрати один з методів оцінки. Підстави для обрання методу.

2. Створити (обрати) зміст Product Backlog.

3. Виконати оцінку розміру за обраним методом.

4. Обчислити економічні показники Product Backlog (навести опис застосування методу на конкретному Product Backlog).

5. Скласти звіт відповідно до встановлених вимог оформлення та захистити роботи..

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які існують переваги та недоліки методології Agile?

2. Які методи оцінювання програмного забезпечення існують

3. Що таке підхід Scrum та які його принципи?

4. Які існують особливості оцінки розміру проєкту при використанні підходу Scrum?

5. На основі яких показників можна здійснювати порівняння методів оцінки розміру програмного забезпечення у контексті Scrum методу?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Basilli V.R. Viewing Maintenance as Reuse-Oriented Software Development // IEEE Software. 1990.– June. P. 19–25.

2. Boehm B.W. Improving Software Productivity // Computer.– 1987. – Vol. 20, n.9.– P. 43–57.

3. Boehm B.W. Software Engineering Economics.– Englewood Cliffs, N.J.: Prentice-Hall.–1981. – 257 p.

4. Boehm B.W. Spiral Model of software Development and Enhancement // Computer. – 1988.– May.– P. 71–73.

5. Bosch J. Design and use of software architectures.– Addison Wesley, 2000.–325 p.

6. Black, R. Critical Testing Processes.– Addison–Wesley, 2003.

7. Blum B. A. Taxonomy of Software Development Methods // Commun. Of the ACM. – 1994. – Vol. 37, n. 11.– P. 82–94.

8. Budgen D. Software design: Reading.– Addison–Wesley, 1994.

9. ISO/IEC 19501:2005. Unified Modeling Language Specification. Version 1.4.2.

10. ISO/IEC 19501:2005. Unified Modeling Language Specification. Version 1.4.2.

11. International Function Point Users Group (IFPUG) Function Point Counting Practices Manual

12. Fenton K. Norman E. Software Metrics: A Rigorous Approach.–London.–Chapman & Hall, 1991.

13. Glass R.L. Extrime programming the good, the bad, and the bottom line//– IEEE. Software. 2001– Vol.18, n.7. – P.11–112.

14. Georgiadou E. Software Process and Product Improvement: A Historical Perspective // Кибернетика и системный анализ .– 2003. – № 1.– С. 147–177.

15. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process.– Addison-Wesley, 1999.– 310 p.

16. Jonsson P. Software Reuse. Architecture, Process and Organization for Business Success. Person Education Asia. – 2002. – 497.

17. Martin J. Rapid Application Development.–Macmillan, 1991. – 250 p.

18. Perry D., Kaiser G. Models of Software Development Environments // IEEE Trans. On Soft. Engin.– 1991. – Vol. 17, n. 3. –
P. 283–295.

19. Pricto-Diaz R. Domain Analysis: An Introduction // Software engineering notes.– 1990. – Vol. 15, №. 2.– H. 47–54.

20. Railich V., Wilde N. et. al. Software cultures and evolution// Computer. 2001. – Sept. – P.25–28.

21. Rombach H.D. Software specifications: a framework.–Carnegie mullon Univ.–SET.–1990.–30p.

22. Rajlich W., Bennett K. A stage Model for the Software Life Cycle // Computer. – 2000.– July.– P. 77 – 70.

23. Seminar on Software Cost Estimation, WS 2002 / 2003, Requirements Engineering Research Group Department of Computer Science University of Zurich, Switzerland.

24. Silverman B. Software Cost and Productivity Improvements: An Analogical view // Compute r.– 1985. – May.– P. 87 – 95.

25. Spillner A., Linz T., Schafer H. Software Testing Foundations.– dpunkt. Verlag.– 2007.– 277 p.

26. Sidorov N. Software Stylistics // Proceedings of NAU.– 2005. – 2(24).– P. 98–103.

27. Sidorov N. Software Engineering. –K.: NAU, 2007. – 130 p.

28. Sidorov N. INTRODUCTION TO SOFTWARE ENGINEERING. – K.: НАУ, 2008. – 140 p.

29. Story points changes in agile iterative development An empirical study and a prediction approach Jirat Pasuksmit Empirical Software Engineering (2022) 27:156 https://doi.org/10.1007/s10664-022-10192-9.

30. Study on Agile Story Point Estimation Techniques and Challenges. Ravi Kiran Mallidi, International Journal of Computer Applications (0975 – 8887) Volume 174 – No. 13, January 2021.

31. SWEBOK Guide V3.0б 2014, IEEE Society.

32. Toriik, Matsumoto K. Gingerz: An Environment for Computer – Aided Empirical Software Engineering // IEEE Trans. On Software Eng.– 1999. – Vol. 25, №. 4 – P. 474–485.

33. Roy K. Clemmons, Project Estimation With Use Case Points, The Journal of Defense Software Engineering? February 2006

Van Veendabl E. Standard glossary of term used in Software testing. –ISTQB. – 2007. Vol. 1,2.– June. – 30 p.

34. Wiegers K. Creting a software engineering culture//Dorset House Publishing New York, 2003. – 358 p.

35. Y. Wang, Software engineering foundations: a software science perspective.- Auerbach Publications Taylor & Francis Group 6000 Broken Sound Parkway NW, Suite 300 Boca Raton, FL 33487-2742

Вельбицкий И.В. Технология программирования.–К.: Техніка, 1984.–274 с.

36. Сидоров М.О. Повторне використання, переробка та відновлення програмного забезпечення. – УсиМ. – 2000.–№ 3, 4. – С. 27–37.

37. Сидоров М.О. Вступ до інженерії програмного забезпечення. – К.: НАУ, 2008. – 65с.

38. Сидоров М.О, Баценко Д.В., Василенко Ю.Н., Щебетін Ю.В. Моделі, методи ті засоби оцінки вартості програмного забезпечення.

39. Хрущ Л.З. Економіка програмного забезпечення : навчальний посібник. Івано-Франківськ : ЛІК, 2018. 103 с.

40. Function Points Analysis Training Course. (2004). Available at: http://www.thimoty.it/softeng/functionpoints_course.pdf.

ДОДАТОК А

ТИТУЛЬНИЙ АРКУШ

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

Практична робота №1

з дисципліни

«Економіка ІТ-індустрії та підприємництво»

«Тема практичної роботи»

Виконав(ла) _____ Група ІП-01

Перевірив _____

Київ 2024

ДОДАТОК Б

QSM SLOC/FP Data				
Language	**Avg**	**Median**	**Low**	**High**
ABAP (SAP) *	28	18	16	60
ASP*	51	54	15	69
Assembler *	119	98	25	320
Brio +	14	14	13	16
C *	97	99	39	333
C++ *	50	53	25	80
C# *	54	59	29	70
COBOL *	61	55	23	297
Cognos Impromptu Scripts +	47	42	30	100
Cross System Products (CSP) +	20	18	10	38
Cool:Gen/IEF *	32	24	10	82
Datastage	71	65	31	157
Excel *	209	191	131	315
Focus *	43	45	45	45
FoxPro	36	35	34	38
HTML *	34	40	14	48
J2EE *	46	49	15	67
Java *	53	53	14	134
JavaScript *	47	53	31	63
JCL *	62	48	25	221
LINC II	29	30	22	38
Lotus Notes *	23	21	19	40
Natural *	40	34	34	53
.NET *	57	60	53	60
Oracle *	37	40	17	60
PACBASE *	35	32	22	60
Perl *	24	15	15	60

PL/I *	64	80	16	80
PL/SQL *	37	35	13	60
Powerbuilder *	26	28	7	40
REXX *	77	80	50	80
Sabretalk *	70	66	45	109
SAS *	38	37	22	55
Siebel *	59	60	51	60
SLOGAN *	75	75	74	75
SQL *	21	21	13	37
VB.NET *	52	60	26	60
Visual Basic *	42	44	20	60

ДОДАТОК В

1. Звіт з лабораторної роботи оформлюється відповідно до наступних вимог: формат А4, гарнітура «Times New Roman», розмір кегля 14 пт, міжрядковий інтервал – 1,5; поля сторінок: ліве – 20 мм, праве – 20 мм, верхнє – 20 мм, нижнє – 20 мм, абзацний відступ – 1,25 см). Ілюстрації, графіки та схеми мають бути вирівняними по центру, обтікання текстом – зверху і знизу. Графічні матеріали мають бути належної якості. Таблиці мають бути вирівняними по ширині сторінки, розмір кегля у таблицях – 12 пт. Таблиці та рисунки підписуються згідно зразка у Додатку 2 після першого згадування у документі. Вміст комірок вирівнюється по центру. У звіті не допускається підкреслення або виділення тексту та висячі рядки. До звіту з лабораторної роботи додається Титульний аркуш відповідно до зразка, що міститься у Додатку 1. Робота має бути збережена у форматі PDF. Програмний код оформлюється шрифтом Courier, має бути встановлено розмір кегля 10 пт.

2. У звіті з лабораторної роботи мають бути конкретні висновки, що розміщуються починаючи з нової сторінки та у яких відповідно до поставленої мети студентом дається оцінка виконаної роботи, порівнюються отримані результати з теоретичними положеннями та очікуваними результатами. Також у звіті має бути список використаних джерел, в якому перелічуються друковані, а потім Інтернет джерела. Список використаних джерел має бути оформлений відповідно до вимог. 3. Захист звіту про лабораторну роботу проводиться індивідуально кожним студентом по теоретичним та практичним аспектам виконаної роботи.