Тема 10. Методи оцінки програмних систем

Зміст

1. Вступ

Оцінка розміру та трудовитрат є одним з найскладніших і водночас найважливіших аспектів управління програмними проектами. Точні оцінки необхідні для планування ресурсів, складання графіків, бюджетування та прийняття стратегічних рішень щодо розробки програмного забезпечення.

Існує відомий парадокс оцінки: ми повинні оцінити проект до його початку, але найбільш точну оцінку можна зробити лише після завершення проекту, коли ця оцінка вже не потрібна. Саме тому були розроблені різноманітні методи та підходи, які допомагають зробити оцінки більш надійними та обґрунтованими.

У цій лекції ми розглянемо різні методи оцінки програмних систем, зосереджуючись на методах вимірювання функціональності та розміру програмного забезпечення. Ми вивчимо як традиційні, так і сучасні підходи, їх сильні та слабкі сторони, а також контекст,

у якому вони найбільш ефективні. Розуміння цих методів допоможе вам обирати найбільш підходящі інструменти для оцінки в конкретних проектних ситуаціях.

2. Оцінка розміру проекту і трудовитрат розробки програмних систем

2.1. Важливість оцінки в управлінні програмними проектами

Оцінка розміру проекту та необхідних трудовитрат є фундаментальним елементом управління програмними проектами з кількох причин:

1. **Планування ресурсів** — визначення кількості персоналу, обладнання та інших ресурсів, необхідних для успішного завершення проекту.
2. **Складання графіку робіт** — встановлення реалістичних термінів і контрольних точок проекту на основі обсягу роботи та наявних ресурсів.
3. **Бюджетування** — розрахунок бюджету проекту на основі оцінених трудовитрат та інших витрат.
4. **Управління очікуваннями** — забезпечення реалістичних очікувань серед зацікавлених сторін щодо термінів, вартості та обсягу проекту.
5. **Прийняття рішень** — надання ключової інформації для прийняття стратегічних рішень, таких як приймати чи відхиляти проект, аутсорсити чи розробляти власними силами.
6. **Контроль проекту** — створення базових показників для відстеження прогресу та виявлення відхилень.
7. **Управління ризиками** — виявлення потенційних ризиків, пов'язаних з обсягом робіт та ресурсами.

2.2. Фактори, що впливають на точність оцінки

Оцінка програмних проектів є нетривіальним завданням через низку факторів, що впливають на її точність:

1. **Нематеріальна природа програмного забезпечення** — на відміну від фізичних продуктів, програмне забезпечення не має фізичних вимірів, що ускладнює оцінку його розміру та складності.
2. **Унікальність кожного проекту** — програмні проекти часто мають унікальні вимоги та умови, що ускладнює використання історичних даних для оцінки.
3. **Мінливість вимог** — вимоги до програмного забезпечення часто змінюються протягом проекту, що впливає на початкові оцінки.
4. **Людський фактор** — продуктивність розробників може суттєво відрізнятися в залежності від їх досвіду, навичок і мотивації.
5. **Технологічні зміни** — нові технології та інструменти можуть як підвищити, так і знизити продуктивність розробки.
6. **Складність програмного забезпечення** — складність алгоритмів, архітектури та взаємодій компонентів системи може бути важко оцінити наперед.
7. **Оптимістичний ухил** — тенденція недооцінювати складність і трудомісткість завдань (синдром "ще трохи і готово").
8. **Тиск з боку керівництва або клієнтів** — зовнішній тиск може призвести до занижених оцінок, що не відображають реальних трудовитрат.
9. **Відсутність стандартизованих процесів оцінки** — різні підходи до оцінки можуть давати різні результати.
10. **Недостатній аналіз ризиків** — нерозпізнані ризики можуть значно вплинути на фактичні трудовитрати.

Розуміння цих факторів допомагає обрати відповідні методи оцінки та інтерпретувати їх результати з необхідною обережністю. Жоден метод не гарантує абсолютно точних оцінок, але систематичний підхід до оцінювання підвищує ймовірність отримання надійних прогнозів.

3. Основні одиниці виміру розміру програмних систем

Для оцінки розміру програмних систем використовуються різні одиниці виміру, кожна з яких має свої переваги та обмеження. Розуміння цих метрик є необхідним для вибору відповідного методу оцінки.

3.1. Рядки коду (LOC/SLOC)

Lines of Code (LOC) або Source Lines of Code (SLOC) — це традиційна метрика, що вимірює розмір програмного забезпечення за кількістю рядків вихідного коду.

Типи LOC:

- **Фізичні рядки коду (PLOC)** — підрахунок усіх рядків у файлах вихідного коду, включаючи коментарі та порожні рядки.
- **Логічні рядки коду (LLOC)** — підрахунок окремих операторів або інструкцій, незалежно від форматування.
- **Чисті рядки коду (NLOC)** — фізичні рядки без коментарів і порожніх рядків.

Переваги LOC:

- Легко зрозуміти та виміряти
- Можливість автоматизації підрахунку
- Наявність великої кількості історичних даних

Обмеження LOC:

- Залежність від мови програмування (одна й та сама функціональність може вимагати різної кількості рядків у різних мовах)
- Не враховує складність коду
- Не вимірює безпосередньо цінність або функціональність для користувача
- Може стимулювати неефективні практики кодування для досягнення цільових показників

Типові значення продуктивності:

- Простий бізнес-додаток: 10-20 рядків коду на людино-день
- Складна система: 2-10 рядків коду на людино-день
- Критична система: 1-5 рядків коду на людино-день

3.2. Функціональні метрики

Функціональні метрики вимірюють розмір програмного забезпечення з точки зору його функціональності для користувача, а не з точки зору кількості рядків коду.

Основні типи функціональних метрик:

- **Function Points (FP)** — вимірює функціональність з точки зору взаємодії з користувачем та даними
- **Feature Points** — розширення FP, що враховує алгоритмічну складність
- **Use Case Points** — базується на аналізі варіантів використання (use cases)
- **Story Points** — оцінка відносної складності користувацьких історій в Agile-методологіях

Переваги функціональних метрик:

- Незалежність від мови програмування та технології реалізації
- Вимірювання функціональності з точки зору користувача
- Можливість оцінки на ранніх етапах проекту до початку кодування
- Кращий зв'язок з бізнес-вимогами

Обмеження функціональних метрик:

- Більш суб'єктивна оцінка порівняно з LOC
- Вимагає досвіду та експертних знань для точної оцінки
- Деякі типи функціональності (наприклад, алгоритмічна складність) можуть бути недостатньо враховані
- Процес оцінки може бути трудомістким

3.3. Об'єктно-орієнтовані метрики

Об'єктно-орієнтовані метрики розроблені спеціально для вимірювання розміру та складності систем, створених з використанням об'єктно-орієнтованої парадигми.

Основні типи об'єктно-орієнтованих метрик:

- **Object Points** — вимірює розмір системи на основі кількості та складності об'єктів
- **Class Points** — вимірює розмір на основі класів та їх атрибутів
- **Кількість класів та інтерфейсів**
- **Метрики Чидамбера та Кемерера** — набір метрик для оцінки різних аспектів ОО-дизайну:
    - Зваженні методи на клас (WMC)
    - Глибина дерева успадкування (DIT)
    - Кількість нащадків (NOC)
    - Зв'язність між класами (CBO)
    - Відгук для класу (RFC)
    - Відсутність зв'язності методів (LCOM)

Переваги об'єктно-орієнтованих метрик:

- Краще відображають структуру ОО-систем
- Враховують специфічні характеристики ОО-парадигми, такі як наслідування та поліморфізм
- Допомагають оцінити якість дизайну, а не лише розмір

Обмеження об'єктно-орієнтованих метрик:

- Можуть бути неефективними для не-ОО систем
- Для деяких метрик існує менше історичних даних
- Можуть вимагати спеціальних інструментів для збору та аналізу

4. Метод функціональних точок

4.1. Історія та основні концепції

Метод функціональних точок (Function Point Analysis, FPA) був розроблений Аланом Альбрехтом з IBM у 1979 році як спосіб вимірювання функціональності програмного забезпечення з точки зору користувача. Метод був створений для подолання обмежень, пов'язаних з використанням рядків коду як міри розміру програмного забезпечення.

Основна концепція методу полягає в оцінці програмного забезпечення на основі функціональності, яку воно надає користувачеві, а не на основі його технічної реалізації. Це дозволяє оцінювати програмне забезпечення незалежно від використовуваних технологій, мов програмування та інструментів розробки.

З моменту створення методу він отримав широке визнання та став стандартом індустрії. У 1986 році була створена Міжнародна асоціація користувачів функціональних точок (IFPUG), яка стандартизувала та розвиває методологію. На сьогодні існує кілька варіацій методу, включаючи IFPUG FPA, COSMIC FP, Mark II FPA та Netherlands Software Metrics Association (NESMA).

4.2. Процес оцінки з використанням функціональних точок

Процес оцінки з використанням методу функціональних точок включає наступні основні кроки:

1. Визначення границь системи

Перш ніж почати оцінку, необхідно чітко визначити, які частини програмного забезпечення будуть включені в оцінку, а які будуть вважатися зовнішніми по відношенню до системи.

2. Ідентифікація користувацьких функцій

Користувацькі функції поділяються на дві категорії:

- **Транзакції** — функції, що обробляють дані
    - Зовнішні входи (External Inputs, EI) — процеси, при яких дані надходять ззовні системи
    - Зовнішні виходи (External Outputs, EO) — процеси, при яких дані виходять за межі системи

    - Зовнішні запити (External Inquiries, EQ) — процеси, що забезпечують отримання даних без їх зміни
- **Дані** — логічні групи даних, що використовуються системою
    - Внутрішні логічні файли (Internal Logical Files, ILF) — логічні групи даних, що підтримуються всередині системи
    - Зовнішні інтерфейсні файли (External Interface Files, EIF) — логічні групи даних, що підтримуються за межами системи, але використовуються нею

3. Класифікація складності кожної функції

Кожна ідентифікована функція класифікується за рівнем складності (низька, середня, висока) на основі:

- Для транзакцій: кількості типів даних і файлів, з якими вона взаємодіє
- Для логічних файлів: кількості типів записів і полів даних

4. Визначення невідкоригованої кількості функціональних точок

Кожній функції присвоюється певна кількість функціональних точок залежно від її типу та складності. Наприклад:

- Зовнішній вхід (EI): 3 (низька), 4 (середня), 6 (висока) точок
- Зовнішній вихід (EO): 4 (низька), 5 (середня), 7 (висока) точок
- Зовнішній запит (EQ): 3 (низька), 4 (середня), 6 (висока) точок
- Внутрішній логічний файл (ILF): 7 (низька), 10 (середня), 15 (висока) точок
- Зовнішній інтерфейсний файл (EIF): 5 (низька), 7 (середня), 10 (висока) точок

Сума всіх цих значень дає невідкориговану кількість функціональних точок (Unadjusted Function Points, UFP).

5. Оцінка факторів складності системи

Оцінюються 14 загальних системних характеристик (General System Characteristics, GSC), кожна за шкалою від 0 (відсутній вплив) до 5 (сильний вплив):

1. Комунікація даних
2. Розподілена обробка даних
3. Продуктивність
4. Інтенсивне використання конфігурації
5. Частота транзакцій
6. Інтерактивний введення даних
7. Ефективність для користувача
8. Інтерактивне оновлення
9. Складність обробки
10. Повторне використання
11. Легкість встановлення
12. Легкість експлуатації
13. Множинність сайтів
14. Сприяння змінам

6. Розрахунок коефіцієнта коригування

На основі оцінок GSC розраховується коефіцієнт коригування (Value Adjustment Factor, VAF): VAF = 0.65 + (Сума всіх GSC × 0.01)

VAF зазвичай знаходиться в діапазоні від 0.65 до 1.35.

7. Розрахунок відкоригованої кількості функціональних точок

Відкоригована кількість функціональних точок (Adjusted Function Points, AFP) розраховується як: AFP = UFP × VAF

4.3. Переваги та обмеження методу

Переваги методу функціональних точок:

- **Незалежність від технології** — оцінка не залежить від мови програмування, платформи або технології реалізації.

- **Ранні оцінки** — може бути застосований на ранніх етапах проекту, коли доступні тільки вимоги.
- **Орієнтація на користувача** — вимірює функціональність з точки зору користувача, що краще відображає бізнес-цінність.
- **Стандартизація** — метод має міжнародний стандарт та підтримується професійними організаціями.
- **Накопичення історичних даних** — існує велика база накопичених даних для порівняння та калібрування.
- **Порівнянність** — дозволяє порівнювати продуктивність різних проектів і команд.

Обмеження методу функціональних точок:

- **Суб'єктивність** — класифікація функцій може бути суб'єктивною та залежати від досвіду оцінювача.
- **Трудомісткість** — процес оцінки вимагає значних зусиль, особливо для великих систем.
- **Необхідність навчання** — для ефективного використання методу потрібне спеціальне навчання та сертифікація.
- **Обмежене врахування алгоритмічної складності** — метод не завжди добре враховує складні алгоритми та обчислення.
- **Складність оцінки для деяких типів систем** — може бути менш ефективним для систем реального часу, вбудованих систем, систем штучного інтелекту тощо.
- **Витрати на підтримку** — вимагає постійного перерахунку при зміні вимог.

5. Метод Function Points (FP)

5.1. Складові елементи Function Points

Метод Function Points (FP), також відомий як метод IFPUG FPA (International Function Point Users Group Function Point Analysis), є найбільш широко використовуваною реалізацією аналізу функціональних точок. Розглянемо детальніше складові елементи цього методу.

Типи функцій даних:

1. **Внутрішні логічні файли (Internal Logical Files, ILF)**
    - Це логічні групи пов'язаних даних, які підтримуються всередині системи
    - Наприклад: таблиці бази даних, файли, які створюються і підтримуються системою
    - Користувач може створювати, змінювати та видаляти дані в ILF через функціональні можливості системи
    - Вага: 7, 10 або 15 FP залежно від складності
2. **Зовнішні інтерфейсні файли (External Interface Files, EIF)**
    - Це логічні групи пов'язаних даних, які використовуються системою, але підтримуються зовні
    - Наприклад: довідкові дані, дані, що надходять з інших систем
    - Система може читати, але не може змінювати дані в EIF
    - Вага: 5, 7 або 10 FP залежно від складності

Типи функцій транзакцій:

1. **Зовнішні входи (External Inputs, EI)**
    - Процеси, при яких дані переходять через межу системи ззовні всередину
    - Дані можуть оновлювати ILF
    - Наприклад: екрани введення даних, операції імпорту даних
    - Вага: 3, 4 або 6 FP залежно від складності
2. **Зовнішні виходи (External Outputs, EO)**
    - Процеси, при яких оброблені дані переходять через межу системи зсередини назовні
    - Включають додаткову обробку, крім простого отримання даних
    - Наприклад: звіти, графіки, повідомлення

- Вага: 4, 5 або 7 FP залежно від складності

3. **Зовнішні запити (External Inquiries, EQ)**
   - Процеси, при яких система надає дані користувачеві без обробки
   - Операція "запит-відповідь" без зміни даних
   - Наприклад: пошукові форми, прості запити
   - Вага: 3, 4 або 6 FP залежно від складності

5.2. Процес підрахунку Function Points

Процес підрахунку Function Points включає наступні кроки:

1. Визначення типу підрахунку

Перед початком підрахунку необхідно визначити його тип:

- **Проект розробки (Development Project)** — оцінка нової системи, що розробляється
- **Проект покращення (Enhancement Project)** — оцінка змін у існуючій системі
- **Прикладна функціональність (Application)** — оцінка існуючої системи

2. Визначення меж системи

Необхідно чітко визначити, що знаходиться всередині системи, а що поза її межами. Межа системи встановлює границю між оцінюваним програмним забезпеченням та зовнішнім середовищем, включаючи користувачів та інші системи.

3. Ідентифікація та класифікація функцій даних (ILF та EIF)

Для кожного ILF та EIF необхідно визначити:

- **Record Element Types (RET)** — підгрупи елементів даних (наприклад, підтаблиці або логічні групи полів)
- **Data Element Types (DET)** — унікальні, користувацькі розпізнавані поля

Складність функцій даних визначається за допомогою наступної матриці:

Кількість DET	1-19 (RET)	20-50 (RET)	51+ (RET)
1	Низька	Низька	Середня
2-5	Низька	Середня	Висока
6+	Середня	Висока	Висока

4. Ідентифікація та класифікація функцій транзакцій (EI, EO, EQ)

Для кожної транзакції необхідно визначити:

- **File Type Referenced (FTR)** — кількість типів файлів (ILF або EIF), з якими взаємодіє транзакція
- **Data Element Types (DET)** — кількість унікальних полів даних, що використовуються в транзакції

Складність функцій транзакцій визначається за допомогою наступних матриць:

Для EI:

Кількість DET	1-4 (FTR)	5-15 (FTR)	16+ (FTR)
1-15	Низька	Низька	Середня
16-25	Низька	Середня	Висока
26+	Середня	Висока	Висока

Для EO та EQ:

Кількість DET	1-5 (FTR)	6-19 (FTR)	20+ (FTR)
1-19	Низька	Низька	Середня
20-50	Низька	Середня	Висока
51+	Середня	Висока	Висока

5. Підрахунок невідкоригованих функціональних точок (UFP)

Підсумовуються ваги всіх ідентифікованих функцій на основі їх типу та складності:

Тип функції	Низька складність	Середня складність	Висока складність
ILF	7	10	15
EIF	5	7	10
EI	3	4	6
EO	4	5	7
EQ	3	4	6

5.3. Коригування складності та розрахунок підсумкової оцінки

Оцінка загальних системних характеристик (GSC)

Оцінюються 14 загальних системних характеристик за шкалою від 0 до 5:

1. **Комунікація даних** — наскільки система використовує комунікаційні засоби для обміну даними
2. **Розподілена обробка даних** — як розподілені дані та обробка
3. **Продуктивність** — вимоги до часу відгуку та продуктивності
4. **Інтенсивне використання конфігурації** — навантаження на наявну технічну інфраструктуру
5. **Частота транзакцій** — як часто виконуються транзакції
6. **Інтерактивний введення даних** — відсоток операцій з інтерактивним введенням
7. **Ефективність для користувача** — фокус на зручності використання
8. **Інтерактивне оновлення** — наскільки інтерактивно оновлюються ILF
9. **Складність обробки** — складність внутрішньої логіки
10. **Повторне використання** — наскільки код розроблено для повторного використання
11. **Легкість встановлення** — складність перетворення та встановлення
12. **Легкість експлуатації** — наскільки ефективні процеси запуску, резервного копіювання та відновлення
13. **Множинність сайтів** — чи розроблена система для встановлення на різних сайтах
14. **Сприяння змінам** — наскільки система спроектована для полегшення змін

Розрахунок коефіцієнта коригування (VAF)

Після оцінки всіх GSC, розраховується загальний рівень впливу (Total Degree of Influence, TDI) як сума всіх 14 оцінок.

Потім розраховується коефіцієнт коригування (Value Adjustment Factor, VAF):

VAF = 0.65 + (TDI × 0.01)

VAF завжди знаходиться в діапазоні від 0.65 до 1.35.

Розрахунок відкоригованих функціональних точок (AFP)

Відкоригована кількість функціональних точок розраховується множенням невідкоригованих функціональних точок на коефіцієнт коригування:

AFP = UFP × VAF

Альтернативні підходи до коригування

У сучасній практиці часто використовуються також спрощені підходи:

- **Відмова від коригування** — використання тільки UFP без VAF
- **Застосування фіксованого VAF** — використання стандартного значення VAF для всіх проектів організації
- **Агрегатне коригування** — застосування коригувальних коефіцієнтів на рівні проекту, а не окремих функцій

5.4. Конвертація Function Points в оцінку трудовитрат

Функціональні точки самі по собі є мірою розміру системи, але для оцінки трудовитрат необхідно конвертувати їх у людино-години або людино-місяці. Для цього використовуються наступні підходи:

1. Використання історичних даних організації

Найбільш точний підхід — використання власних історичних даних організації про продуктивність розробки:

- Збір даних про фактичні трудовитрати та розмір у функціональних точках для завершених проектів
- Розрахунок показника продуктивності: годин на функціональну точку (Hours per Function Point, HPFP)
- Врахування типу проекту, технології, команди та інших факторів

2. Використання галузевих еталонів

Якщо історичних даних недостатньо, можна використовувати галузеві еталони:

- За даними ISBSG (International Software Benchmarking Standards Group), середня продуктивність становить 8-12 людино-годин на функціональну точку для бізнес-додатків
- Для різних мов програмування та платформ можуть бути різні показники

3. Конвертація в рядки коду

Інший підхід — конвертація функціональних точок у рядки коду, а потім використання моделей оцінки на основі рядків коду:

- Існують таблиці конверсії FP у LOC для різних мов програмування
- Наприклад, 1 FP відповідає приблизно:
    - 130 рядкам коду на C
    - 105 рядкам коду на C++
    - 55 рядкам коду на Java
    - 30 рядкам коду на Visual Basic
    - 20 рядкам коду на SQL

4. Використання моделей оцінки

Можна використовувати функціональні точки як вхідні дані для моделей оцінки трудовитрат:

- COCOMO II безпосередньо використовує функціональні точки як вхідні дані
- Інші моделі оцінки також можуть бути адаптовані для роботи з функціональними точками

5. Формули оцінки трудовитрат

Базова формула для оцінки трудовитрат: Трудовитрати = AFP × Коефіцієнт продуктивності
Коефіцієнт продуктивності може залежати від багатьох факторів:

- Складність проекту
- Досвід команди
- Технологічне середовище
- Методологія розробки
- Вимоги до якості

6. Метод Early Function Points (EFP)

6.1. Особливості оцінки на ранніх етапах проекту

Оцінка розміру та трудовитрат на ранніх етапах проекту є особливо складною через обмежену інформацію та високий рівень невизначеності. Однак саме на цих етапах необхідно приймати критичні рішення щодо бюджету, графіку та доцільності проекту.
Основні виклики ранньої оцінки:

- Неповні або нечіткі вимоги
- Відсутність детальної інформації про дані та функції
- Невизначеність щодо технічної архітектури
- Можливі зміни обсягу проекту
- Обмежений час на проведення оцінки

Метод Early Function Points (EFP) був розроблений спеціально для вирішення цих проблем та забезпечення можливості оцінки на ранніх етапах проекту, коли доступна лише високорівнева інформація про вимоги.

6.2. Різниця між EFP та стандартними Function Points

Метод Early Function Points є адаптацією стандартного методу Function Points для ранніх етапів проекту. Основні відмінності включають:

1. Спрощення процесу ідентифікації та класифікації

- **Стандартні FP**: Вимагають детальної ідентифікації всіх функцій даних та транзакцій, включаючи конкретні поля (DET) та логічні групи даних (RET)
- **EFP**: Використовує високорівневий підхід, зосереджуючись на основних бізнес-функціях та сутностях без необхідності детальної деталізації

2. Агрегація функціональності

- **Стандартні FP**: Кожен екран, звіт, файл оцінюється окремо
- **EFP**: Функціональність може бути агрегована та оцінена на рівні бізнес-процесів або функціональних областей

3. Підхід до визначення складності

- **Стандартні FP**: Складність визначається кількістю DET і FTR/RET за допомогою детальних матриць
- **EFP**: Складність часто визначається експертним судженням або спрощеними критеріями

4. Коригування та адаптація

- **Стандартні FP**: Використовує 14 GSC для коригування
- **EFP**: Може використовувати спрощені фактори коригування або специфічні для раннього етапу фактори ризику

5. Точність та мета

- **Стандартні FP**: Орієнтовані на високу точність оцінки
- **EFP**: Орієнтовані на отримання прийнятної початкової оцінки з визнанням більшої невизначеності

6.3. Процес оцінки з використанням EFP

Процес оцінки з використанням методу Early Function Points включає наступні кроки:

1. Аналіз доступної інформації

- Збір та аналіз високорівневих вимог та бізнес-процесів
- Ідентифікація основних сутностей даних (потенційні ILF та EIF)
- Визначення ключових функціональних областей
- Розуміння потрібних інтеграцій з іншими системами

2. Ідентифікація макро-функцій

Замість детальної ідентифікації окремих функцій, EFP працює з макро-функціями, такими як:

- Основні бізнес-процеси
- Функціональні підсистеми
- Модулі системи
- Групи взаємопов'язаних сутностей та операцій

3. Оцінка кожної макро-функції

Для кожної макро-функції:

- Оцінка приблизної кількості сутностей даних (ILF/EIF) та їх складності
- Оцінка приблизної кількості транзакцій (EI/EO/EQ) та їх складності
- Використання спрощених матриць складності або експертного судження

4. Розрахунок невідкоригованих EFP

Підсумовуються оцінки всіх макро-функцій для отримання загальної кількості невідкоригованих EFP.

5. Застосування коригувальних факторів

На цьому етапі можуть бути застосовані різні коригувальні фактори:

- Спрощена оцінка GSC
- Фактори ризику ранньої оцінки

- Діапазони невизначеності

6. Розрахунок діапазону оцінки

Оскільки рання оцінка має вищий рівень невизначеності, результат часто виражається у вигляді діапазону:

- Оптимістична оцінка
- Найбільш ймовірна оцінка
- Песимістична оцінка

7. Уточнення оцінки з появою нової інформації

Важливою особливістю методу EFP є його ітеративна природа:

- Оцінка регулярно переглядається з появою нової інформації
- При деталізації вимог здійснюється перехід до більш точних методів оцінки
- Відстежуються відхилення від початкової оцінки для поліпшення майбутніх оцінок

8. Конвертація EFP в трудовитрати

Конвертація EFP в трудовитрати здійснюється аналогічно стандартним FP, але з врахуванням більшої невизначеності:

- Використання історичних даних з аналогічних проектів
- Застосування діапазонів продуктивності
- Додаткові резерви для врахування ризиків ранньої оцінки

Практичні рекомендації щодо використання EFP

- Документуйте всі припущення, зроблені під час оцінки
- Визначте фактори, що мають найбільший вплив на точність оцінки
- Використовуйте множинні методи оцінки для перехресної перевірки
- Регулярно переглядайте оцінку з появою нової інформації
- Встановлюйте реалістичні очікування щодо точності ранньої оцінки

7. Метод точок властивостей (Feature Points)

7.1. Розширення методу функціональних точок

Метод точок властивостей (Feature Points) був розроблений Калей-Джонс (Capers Jones) у 1986 році як розширення методу функціональних точок для подолання його обмежень при оцінці систем з високою алгоритмічною складністю. В той час як стандартний метод функціональних точок добре працює для бізнес-додатків, він не завжди адекватно відображає складність систем, які містять складні алгоритми та обчислення.

Основна ідея методу точок властивостей полягає в додаванні нової категорії — алгоритмів — до стандартних категорій функціональних точок.

Цей метод особливо корисний для оцінки:

- Системи реального часу
- Вбудовані системи
- Інженерні та наукові додатки
- Системи управління процесами
- Телекомунікаційні системи
- Системи з комплексною бізнес-логікою
- Системи штучного інтелекту та машинного навчання

7.2. Врахування алгоритмічної складності

Основна відмінність методу точок властивостей від стандартних функціональних точок полягає в явному врахуванні алгоритмічної складності:

1. Додавання нової категорії — алгоритмів

В додаток до п'яти стандартних категорій IFPUG (ILF, EIF, EI, EO, EQ), метод точок властивостей додає шосту категорію — алгоритми:

- **Алгоритми (ALG)** — унікальні, самодостатні логічні обробки, що виконують конкретні бізнес- або технічні операції

- Приклади: складні розрахунки, оптимізаційні алгоритми, алгоритми маршрутизації, алгоритми стиснення даних, алгоритми шифрування, алгоритми обробки зображень
- Вага: 3 точки властивостей для кожного ідентифікованого алгоритму (стандартна вага)

2. Зміна ваг для інших категорій

Метод точок властивостей також змінює ваги деяких стандартних категорій:

Категорія	Стандартна вага (FP)	Вага в методі точок властивостей
ILF	7/10/15	7 (стандартна вага)
EIF	5/7/10	5 (стандартна вага)
EI	3/4/6	3 (стандартна вага)
EO	4/5/7	4 (стандартна вага)
EQ	3/4/6	3 (стандартна вага)
ALG	-	3 (нова категорія)

Важливо зазначити, що в методі точок властивостей часто використовуються стандартні (фіксовані) ваги для всіх категорій, а складність визначається кількістю елементів у кожній категорії, а не матрицями складності, як в IFPUG. Це спрощує процес оцінки, хоча і може знижувати його точність для деяких систем.

3. Критерії ідентифікації алгоритмів

Не всі логічні процеси класифікуються як алгоритми. Щоб бути зарахованим як алгоритм, процес повинен відповідати певним критеріям:

- **Унікальність** — алгоритм повинен бути унікальним для системи та вирішувати конкретну проблему
- **Складність** — алгоритм повинен включати складні обчислення або логіку
- **Самодостатність** — алгоритм повинен бути самодостатнім компонентом
- **Значимість** — алгоритм повинен вносити значний внесок у функціональність системи

4. Класифікація алгоритмічної складності

У розширених версіях методу точок властивостей алгоритми можуть класифікуватися за рівнем складності:

- **Проста алгоритмічна складність** — 3 точки
- **Середня алгоритмічна складність** — 4 точки
- **Висока алгоритмічна складність** — 6 точок

Класифікація може базуватися на різних критеріях, таких як:

- Кількість логічних умов та рішень
- Кількість математичних операцій
- Складність структур даних, що використовуються
- Обчислювальна складність (O-нотація)
- Кількість вхідних параметрів та вихідних результатів

7.3. Область застосування методу

Метод точок властивостей має специфічну область застосування, де він особливо ефективний:

1. Типи систем, для яких метод найбільш ефективний

- **Системи реального часу** — системи, що взаємодіють з фізичним середовищем в реальному часі
- **Вбудовані системи** — програмне забезпечення, вбудоване в апаратне середовище
- **Інженерні та наукові додатки** — системи, що використовують складні математичні моделі
- **Системи управління процесами** — системи автоматизації та контролю

- **Телекомунікаційні системи** — системи комунікації та обробки сигналів
- **Системи з комплексною бізнес-логікою** — системи зі складними правилами прийняття рішень
- **Системи штучного інтелекту** — системи, що використовують методи машинного навчання та обробки природної мови

2. Порівняння з іншими методами оцінки

Аспект	Function Points	Feature Points	Object Points	LOC
Алгоритмічна складність	Слабке врахування	Сильне врахування	Середнє врахування	Пряме відображення
Складність даних	Сильне врахування	Сильне врахування	Сильне врахування	Слабке врахування
Складність інтерфейсу користувача	Сильне врахування	Сильне врахування	Середнє врахування	Слабке врахування
Складність об'єктної моделі	Не враховується	Не враховується	Сильне врахування	Слабке врахування
Потрібний рівень деталізації	Високий	Середній	Середній	Дуже високий
Можливість ранньої оцінки	Середня	Середня	Висока	Низька
Стандартизація	Висока	Середня	Низька	Висока

3. Практичні аспекти використання методу точок властивостей

- **Поєднання з іншими методами** — метод точок властивостей часто використовується разом з іншими методами оцінки для отримання більш повної картини
- **Калібрування** — для підвищення точності необхідно калібрувати метод на основі історичних даних організації
- **Автоматизація** — існують інструменти, що підтримують оцінку з використанням точок властивостей
- **Навчання** — правильне використання методу вимагає навчання та досвіду, особливо в ідентифікації та класифікації алгоритмів

4. Приклад застосування

Розглянемо приклад системи управління запасами з елементами оптимізації:

Категорія	Кількість	Вага	Точки властивостей
ILF	8	7	56
EIF	3	5	15
EI	12	3	36
EO	10	4	40
EQ	15	3	45
ALG	4	3	12
Загалом			204

Алгоритми включають:

1. Алгоритм прогнозування попиту
2. Алгоритм оптимізації рівнів запасів
3. Алгоритм маршрутизації для оптимізації доставки
4. Алгоритм визначення ціноутворення на основі множинних факторів

Оцінка трудовитрат:

- Припустимо, що історичні дані організації показують продуктивність 10 годин на точку властивостей
- Очікувані трудовитрати: 204 × 10 = 2040 людино-годин

8. Метод об'єктних точок (Object Points)

8.1. Оцінка в контексті об'єктно-орієнтованої розробки

Метод об'єктних точок (Object Points) був розроблений для оцінки розміру та трудовитрат програмних систем, створених з використанням об'єктно-орієнтованої парадигми. Цей метод враховує особливості об'єктно-орієнтованої розробки, такі як класи, об'єкти, успадкування та інкапсуляція.

Метод об'єктних точок виник як відповідь на обмеження традиційних методів оцінки, таких як функціональні точки, при застосуванні до об'єктно-орієнтованих систем. Він також добре підходить для систем, розроблених за допомогою інструментів швидкої розробки додатків (RAD) та систем, заснованих на компонентах.

Основні концепції методу об'єктних точок:

1. **Фокус на об'єктах, а не на функціях** — метод розглядає систему як набір взаємодіючих об'єктів, а не як набір функцій та даних.
2. **Врахування повторного використання** — метод явно враховує можливість повторного використання компонентів, що є важливим аспектом об'єктно-орієнтованої розробки.
3. **Врахування об'єктно-орієнтованих характеристик** — метод враховує такі аспекти, як успадкування, поліморфізм та інкапсуляція, які впливають на складність системи.
4. **Придатність для ранньої оцінки** — метод може бути застосований на ранніх етапах проекту, коли доступна тільки високорівнева інформація про об'єкти системи.

8.2. Процес підрахунку об'єктних точок

Існує кілька варіантів методу об'єктних точок, але найбільш поширений підхід включає наступні кроки:

1. Ідентифікація екранів, звітів та компонентів 3GL

На першому етапі ідентифікуються три типи об'єктів:

- **Екрани (Screens)** — інтерфейси користувача, форми введення даних
- **Звіти (Reports)** — вихідні формати, звіти, графіки
- **Компоненти 3GL (3GL Components)** — компоненти, розроблені з використанням мов 3-го покоління (Java, C++, тощо)

2. Класифікація складності кожного об'єкта

Кожен об'єкт класифікується за рівнем складності:

Для екранів:

- **Проста складність**: Екран з невеликою кількістю полів даних, мінімальною валідацією, без складних операцій
- **Середня складність**: Екран з помірною кількістю полів, деякою валідацією, простими операціями
- **Висока складність**: Екран з великою кількістю полів, складною валідацією, складними операціями

Для звітів:

- **Проста складність**: Простий список або таблиця з мінімальною кількістю обчислень
- **Середня складність**: Звіт з деякими обчисленнями, групуванням та сортуванням
- **Висока складність**: Складний звіт з багатьма обчисленнями, групуваннями, сортуваннями, графіками

Для компонентів 3GL:

- **Проста складність**: Компонент з простою логікою та невеликою кількістю методів

- **Середня складність**: Компонент з помірною логікою, декількома методами, деяким успадкуванням
- **Висока складність**: Компонент зі складною логікою, багатьма методами, глибоким успадкуванням

3. Призначення ваг кожному об'єкту

Кожному об'єкту призначається вага в залежності від його типу та складності:

Тип об'єкта	Проста складність	Середня складність	Висока складність
Екран	1	2	3
Звіт	2	5	8
Компонент 3GL	10	15	30

4. Розрахунок невідкоригованих об'єктних точок (UOP)

Підсумовуються ваги всіх ідентифікованих об'єктів для отримання загальної кількості невідкоригованих об'єктних точок (Unadjusted Object Points, UOP).

5. Оцінка рівня повторного використання

Оцінюється очікуваний рівень повторного використання об'єктів (Reuse Percentage, RP) — відсоток об'єктів, які будуть повторно використані з існуючих бібліотек або попередніх проектів.

6. Розрахунок нових об'єктних точок (NOP)

Розраховується кількість нових об'єктних точок (New Object Points, NOP), які потрібно розробити:

NOP = UOP × (100% - RP)

7. Оцінка продуктивності на основі досвіду та середовища розробки

Оцінюється продуктивність розробки на основі досвіду розробників та характеристик середовища розробки:

Досвід розробників / Середовище	Дуже низька	Низька	Номінальна	Висока	Дуже висока
PROD	4	7	13	25	50

Де PROD — кількість об'єктних точок, які можуть бути розроблені за людино-місяць.

8. Розрахунок трудовитрат

Трудовитрати розраховуються як:

Трудовитрати (людино-місяці) = NOP / PROD

8.3. Використання методу при розробці на основі компонентів

Метод об'єктних точок особливо ефективний при оцінці систем, що розробляються з використанням компонентного підходу, таких як:

- Веб-додатки на основі компонентів
- Настільні додатки, розроблені з використанням RAD-інструментів
- Бізнес-додатки, створені на базі фреймворків
- Мобільні додатки, розроблені з використанням готових компонентів

Особливості застосування методу для компонентної розробки:

1. **Розширена класифікація компонентів**

При компонентній розробці класифікація компонентів може бути розширена та деталізована:

- **Базові компоненти інтерфейсу** (кнопки, текстові поля, випадаючі списки)
- **Складні компоненти інтерфейсу** (таблиці даних, деревовидні структури, графічні редактори)
- **Компоненти бізнес-логіки** (валідатори, калькулятори, конвертери)
- **Сервісні компоненти** (компоненти доступу до даних, комунікаційні компоненти)
- **Компоненти інтеграції** (адаптери, шлюзи, перетворювачі)

2. **Врахування конфігурації компонентів**

Метод об'єктних точок для компонентної розробки може враховувати складність конфігурації готових компонентів:

- **Проста конфігурація**: Мінімальні зміни стандартних налаштувань
- **Середня конфігурація**: Значна кількість налаштувань, деяка кастомізація
- **Складна конфігурація**: Глибока кастомізація, розширення функціональності компонента

3. **Розширений підхід до повторного використання**

Враховуються різні аспекти повторного використання:

- **Використання готових компонентів** без змін
- **Адаптація існуючих компонентів** з мінімальними змінами
- **Значна модифікація існуючих компонентів**
- **Створення нових компонентів** з нуля, але з потенціалом для повторного використання

4. **Інтеграція з COCOMO II**

Метод об'єктних точок часто інтегрується з моделлю COCOMO II для уточнення оцінки трудовитрат:

- Об'єктні точки використовуються для оцінки розміру системи
- COCOMO II використовується для конвертації розміру в трудовитрати з урахуванням факторів вартості

5. **Практичні рекомендації щодо використання методу при компонентній розробці**

- **Каталогізація компонентів**: Створення та підтримка каталогу доступних компонентів з оцінкою їх складності
- **Стандартизація**: Використання стандартних підходів до конфігурації та кастомізації компонентів
- **Збір метрик**: Відстеження фактичних трудовитрат на розробку та конфігурацію компонентів для покращення майбутніх оцінок
- **Врахування інтеграційних витрат**: Окрема оцінка трудовитрат на інтеграцію компонентів, які часто недооцінюються

9. Порівняльний аналіз методів оцінки

Розглянуті методи оцінки програмних систем мають свої сильні та слабкі сторони, а також специфічні області застосування. Нижче наведено порівняльний аналіз цих методів за різними аспектами.

9.1. Порівняння за типом метрики та підходом

Метод оцінки	Тип метрики	Підхід до оцінки	Основний фокус
Lines of Code (LOC)	Технічна метрика	Вимірювання обсягу коду	Технічна реалізація
Function Points (FP)	Функціональна метрика	Оцінка функціональності системи	Функціональність з точки зору користувача
Early Function Points	Функціональна метрика	Рання оцінка функціональності	Рання оцінка на основі високорівневих вимог
Feature Points	Розширена функціональна метрика	Оцінка функціональності та алгоритмів	Функціональність та алгоритмічна складність
Object Points	Об'єктна метрика	Оцінка на основі об'єктів та компонентів	Об'єкти та повторне використання

9.2. Порівняння за застосовністю до різних типів систем

Тип системи	LOC	Function Points	Early Function Points	Feature Points	Object Points
Бізнес-додатки	✓	✓✓✓	✓✓	✓✓	✓✓
Веб-додатки	✓	✓✓	✓✓	✓	✓✓✓
Мобільні додатки	✓	✓✓	✓✓	✓	✓✓✓
Наукові/інженерні додатки	✓✓	✓	✓	✓✓✓	✓
Системи реального часу	✓✓	✓	✓	✓✓✓	✓
Вбудовані системи	✓✓✓	✓	✓	✓✓✓	✓
ОО-системи	✓	✓	✓	✓	✓✓✓
Компонентні системи	✓	✓	✓	✓	✓✓✓

9.3. Порівняння за етапом життєвого циклу

Етап життєвого циклу	LOC	Function Points	Early Function Points	Feature Points	Object Points
Концепція проекту	✗	✓	✓✓✓	✓	✓✓
Аналіз вимог	✗	✓✓	✓✓✓	✓✓	✓✓
Проектування	✓	✓✓✓	✓✓	✓✓✓	✓✓✓
Реалізація (кодування)	✓✓✓	✓✓	✓	✓✓	✓✓
Тестування	✓✓✓	✓✓	✓	✓✓	✓✓
Супровід	✓✓✓	✓✓	✓	✓✓	✓✓

9.4. Порівняння за точністю, трудомісткістю та суб'єктивністю

Метод оцінки	Точність оцінки	Трудомісткість оцінки	Суб'єктивність оцінки	Потреба в експертизі
Lines of Code (LOC)	Висока (після кодування)	Низька	Низька	Низька
Function Points (FP)	Висока	Висока	Середня	Висока
Early Function Points	Середня	Середня	Висока	Висока
Feature Points	Висока для алгоритмічних систем	Висока	Середня-висока	Висока
Object Points	Висока для ОО-систем	Середня	Середня	Середня-висока

9.5. Порівняння за перевагами та обмеженнями

Метод оцінки	Основні переваги	Основні обмеження
Lines of Code (LOC)	Простота, об'єктивність, легкість автоматизації	Залежність від мови, неможливість ранньої оцінки, не відображає функціональну цінність
Function Points (FP)	Незалежність від технології, зв'язок з бізнес-вимогами, можливість ранньої оцінки	Трудомісткість, суб'єктивність, недостатнє врахування алгоритмічної складності

Метод оцінки	Основні переваги	Основні обмеження
Early Function Points	Можливість дуже ранньої оцінки, швидкість, простота	Нижча точність, висока суб'єктивність, потреба в експертизі
Feature Points	Врахування алгоритмічної складності, придатність для технічних систем	Менша стандартизація, суб'єктивність в оцінці алгоритмів, складність навчання
Object Points	Придатність для ОО-систем, врахування повторного використання, компонентний підхід	Обмежена область застосування, менша стандартизація, обмежена документація

10. Практичні рекомендації щодо використання методів оцінки

10.1. Вибір відповідного методу оцінки

При виборі методу оцінки програмних систем слід враховувати наступні фактори:

1. **Тип проекту та його характеристики**
   - Для бізнес-додатків: Function Points або Object Points
   - Для систем з високою алгоритмічною складністю: Feature Points
   - Для об'єктно-орієнтованих систем: Object Points
   - Для компонентних систем: Object Points
   - Для вбудованих систем: Feature Points або LOC
2. **Етап життєвого циклу**
   - На ранніх етапах: Early Function Points
   - На етапі аналізу: Function Points, Feature Points
   - На етапі проектування: Object Points, Function Points
   - На етапі реалізації: LOC, Object Points
3. **Доступність інформації**
   - Мінімальна інформація: Early Function Points
   - Високорівневі вимоги: Function Points, Feature Points
   - Детальний дизайн: Object Points
   - Готовий код: LOC
4. **Наявність історичних даних**
   - Використовуйте методи, для яких у організації є накопичені дані
   - При відсутності історичних даних починайте з стандартизованих методів (наприклад, IFPUG FP)
5. **Досвід команди**
   - Враховуйте наявність навичок та досвіду використання конкретних методів
   - При відсутності досвіду починайте з простіших методів

10.2. Підвищення точності оцінок

Для підвищення точності оцінок програмних систем рекомендується:

1. **Використання кількох методів**
   - Застосовуйте 2-3 різних методи та порівнюйте результати
   - Аналізуйте розбіжності між оцінками та їх причини
   - Розглядайте діапазон оцінок, а не точкове значення
2. **Калібрування методів**
   - Адаптуйте стандартні методи до специфіки організації
   - Використовуйте історичні дані для корекції параметрів методів
   - Регулярно перевіряйте та оновлюйте калібрування
3. **Облік факторів ризику**
   - Ідентифікуйте фактори ризику, що можуть вплинути на оцінку
   - Розробіть сценарії (оптимістичний, найбільш ймовірний, песимістичний)

    - Використовуйте PERT-формулу для зваженої оцінки: (O + 4M + P) / 6
4. **Залучення експертів**
    - Проводьте експертні оцінки з залученням фахівців різних ролей
    - Використовуйте методи досягнення консенсусу (Дельфі, Planning Poker)
    - Аналізуйте причини розбіжностей між експертними оцінками
5. **Регулярне уточнення**
    - Переглядайте оцінки з появою нової інформації
    - Розбивайте великі системи на менші компоненти для підвищення точності
    - Документуйте всі припущення та рішення, прийняті при оцінці

10.3. Інтеграція методів оцінки в процес розробки

Для ефективної інтеграції методів оцінки в процес розробки програмного забезпечення:

1. **Для традиційних (водоспадних) методологій**
    - Проводіть детальну оцінку на початку проекту
    - Використовуйте функціональні точки або об'єктні точки на етапі аналізу та проектування
    - Регулярно переглядайте оцінку при проходженні контрольних точок
2. **Для гнучких (Agile) методологій**
    - Використовуйте спрощені методи, такі як Story Points
    - Інтегруйте оцінку функціональних точок з оцінкою користувацьких історій
    - Проводіть регулярну переоцінку з кожним спринтом
3. **Для DevOps та безперервної поставки**
    - Фокусуйтеся на відносних оцінках та швидкості команди
    - Збирайте метрики автоматично в процесі розробки
    - Використовуйте історичні дані для прогнозування майбутньої продуктивності
4. **Для великих програм та портфелів**
    - Використовуйте стандартизовані методи для забезпечення порівнянності різних проектів
    - Застосовуйте ієрархічну оцінку (зверху вниз та знизу вгору)
    - Забезпечте узгодженість методів між різними командами та проектами

10.4. Автоматизація процесу оцінки

Для підвищення ефективності процесу оцінки доцільно використовувати автоматизацію:

1. **Інструменти для підрахунку функціональних точок**
    - CAST Function Point Analysis
    - SCOPE Function Point Workbench
    - FPMeter
    - QSM SLIM-Estimate
2. **Інструменти для підрахунку рядків коду**
    - CLOC (Count Lines of Code)
    - SonarQube
    - SourceMonitor
    - Understand
3. **Інтегровані рішення для оцінки та управління проектами**
    - QSM SLIM Suite
    - SEER-SEM
    - Microsoft Project with Function Point Extension
    - Jira з плагінами для оцінки
4. **Розробка власних інструментів**
    - Створення шаблонів у електронних таблицях для автоматизації розрахунків
    - Інтеграція з системами управління вимогами та проектами
    - Розробка внутрішніх баз даних проектних метрик

10.5. Документування та аналіз результатів оцінки

Для покращення процесу оцінки в довгостроковій перспективі:

1. **Документування**
   - Фіксуйте всі припущення, зроблені під час оцінки
   - Документуйте використані методи та параметри
   - Зберігайте вхідні дані та результати оцінки
2. **Порівняння прогнозу з фактичними результатами**
   - Регулярно порівнюйте оцінки з фактичними трудовитратами
   - Аналізуйте причини відхилень
   - Використовуйте результати аналізу для покращення майбутніх оцінок
3. **Накопичення бази знань**
   - Створюйте та підтримуйте базу даних проектних метрик
   - Розробляйте власні моделі оцінки на основі накопичених даних
   - Діліться знаннями та досвідом оцінки всередині організації
4. **Регулярний перегляд процесу оцінки**
   - Періодично оцінюйте ефективність використовуваних методів
   - Впроваджуйте нові методи та інструменти при необхідності
   - Оновлюйте процес оцінки з врахуванням змін у технологіях та методологіях розробки

11. Висновки

Оцінка програмних систем є складним і багатогранним процесом, який вимагає поєднання формальних методів, експертних знань та історичних даних. У цій лекції ми розглянули різні методи оцінки, їх переваги, обмеження та області застосування.

Основні висновки:

1. **Немає універсального методу оцінки.** Кожен метод має свої сильні та слабкі сторони, і вибір методу залежить від типу проекту, етапу життєвого циклу, доступної інформації та інших факторів.
2. **Методи функціональних точок (FP, EFP)** є потужними інструментами для оцінки функціональності з точки зору користувача, незалежно від технології реалізації. Вони особливо ефективні для бізнес-додатків та дозволяють проводити оцінку на ранніх етапах проекту.
3. **Метод точок властивостей (Feature Points)** розширює метод функціональних точок для врахування алгоритмічної складності, що робить його більш придатним для систем реального часу, вбудованих систем та інженерних додатків.
4. **Метод об'єктних точок (Object Points)** адаптований для оцінки об'єктно-орієнтованих та компонентних систем, враховуючи такі аспекти, як повторне використання та об'єктна структура.
5. **Поєднання різних методів** часто дає найбільш надійні результати, дозволяючи використовувати сильні сторони кожного методу та компенсувати їх обмеження.
6. **Важливість калібрування та адаптації** методів до специфіки організації та проекту не можна недооцінювати. Стандартні параметри методів часто потребують корекції на основі історичних даних.
7. **Оцінка — це безперервний процес**, а не одноразовий захід. З появою нової інформації оцінки повинні переглядатися та уточнюватися.
8. **Автоматизація та інструменти** можуть значно підвищити ефективність та точність процесу оцінки, але вони не замінюють експертного судження та досвіду.
9. **Документування та аналіз результатів** оцінки є критично важливими для навчання та вдосконалення процесу в майбутньому.
10. **Інтеграція методів оцінки в процес розробки** забезпечує безперервне уточнення оцінок та їх використання для прийняття управлінських рішень.

Ефективна оцінка програмних систем вимагає балансу між формальними методами та практичним досвідом, між деталізацією та швидкістю, між точністю та корисністю. Розуміння різних методів оцінки та їх правильне застосування є важливою компетенцією для менеджерів проектів, аналітиків та інших фахівців, залучених до планування та управління програмними проектами.

12. Список літератури

1. Albrecht, A. J. (1979). "Measuring Application Development Productivity." Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, pp. 83-92.
2. Boehm, B. W. (2000). "Software Cost Estimation with COCOMO II." Prentice Hall.
3. Capers Jones. (1996). "Applied Software Measurement: Assuring Productivity and Quality." McGraw-Hill.
4. Garmus, D., & Herron, D. (2001). "Function Point Analysis: Measurement Practices for Successful Software Projects." Addison-Wesley Professional.
5. IFPUG. (2010). "Function Point Counting Practices Manual, Release 4.3." International Function Point Users Group.
6. Karner, G. (1993). "Resource Estimation for Objectory Projects." Objective Systems SF AB.
7. McConnell, S. (2006). "Software Estimation: Demystifying the Black Art." Microsoft Press.
8. NESMA. (2004). "NESMA Counting Practices Manual, Version 2.2." Netherlands Software Metrics Association.
9. Pfleeger, S. L. (2001). "Software Engineering: Theory and Practice." Prentice Hall.
10. Pressman, R. S. (2014). "Software Engineering: A Practitioner's Approach." McGraw-Hill Education.
11. Sommerville, I. (2015). "Software Engineering." Pearson.
12. Symons, C. R. (1988). "Function Point Analysis: Difficulties and Improvements." IEEE Transactions on Software Engineering, 14(1), 2-11.
13. Whitmire, S. A. (1995). "An Introduction to 3D Function Points." Software Development, April.