ТЕМА 2. *Життєвий цикл програмного забезпечення*

Життєвий цикл програмного забезпечення.

Існує чимало різних моделей або представлень життєвого циклу розроблення ПЗ. Всі вони представляють собою логічно побудовану послідовність дій, починаючи з визначення потреби і закінчуючи виробництвом ПЗ. Кожна модель представляє собою процес, який структурно складається з з етапів, спрямованих на забезпечення цілісності відповідних дій. Кожний повністю відпрацьований етап знижує ступінь ризику при виконанні проекту завдяки застосуванню критеріїв входу та виходу для визначення подальших дій. Життєві цикли розроблення ПЗ є методиками, які охоплюють всі стандарти і процедури, які впливають на планування, збір і аналіз вимог, розроблення проекту, конструювання та впровадження програмної системи. З метою забезпечення ефективності життєвого циклу його слід ретельно обрати та підлаштувати відповідно до задач і цілей певного проекту. Популярні узагальнені моделі надають лише початкові схеми. Кожна така схема має недоліки та переваги, які й визначають можливість її застосування для певних типів проектів.

Найбільшого поширення набули наступні моделі ЖЦ ПЗ: каскадна (водоспадна), інкрементна, спіральна, еволюційна моделіта модель стандартизованого ЖЦ ПЗ.

Однією з перших стала застосовуватися каскадна модель ЖЦ ПЗ (waterfall model), в якій кожна робота виконується один разі в тому порядку, як це представлено у моделі (рис.1.1).

Відповідно до цієї моделі ЖЦ, роботи та завдання процесу розроблення зазвичай виконуються послідовно, як це представлено у схемі. Однак допоміжні та організаційні процеси (контроль вимог, управління якістю) зазвичай виконуються паралельно з процесом розроблення. У даній моделі повернення до початкового процесу передбачається після супроводу та виправлення помилок.

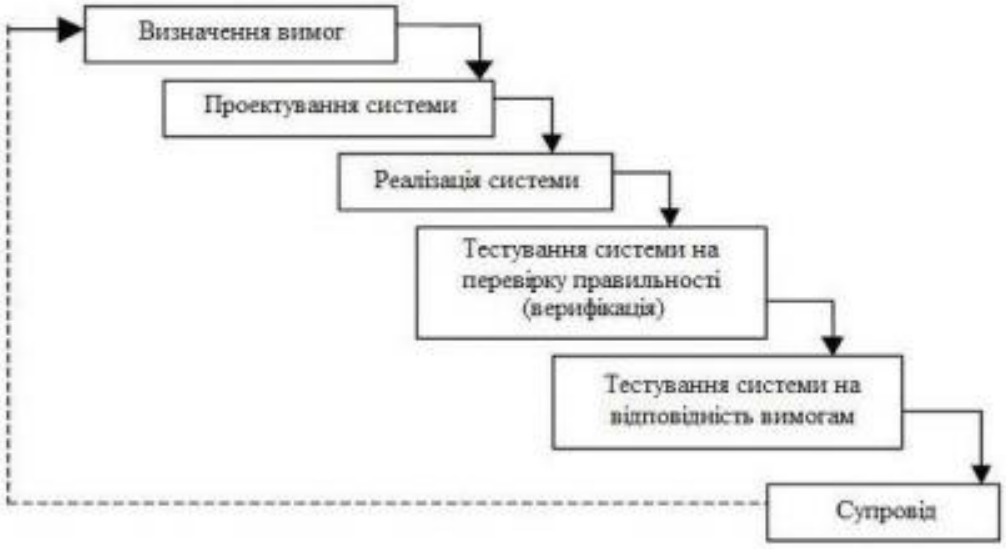


Рис.1.1 - Каскадна модель ЖЦ програмного забезпечення

Переваги реалізації ПЗ за допомогою каскадної моделі наступні:
- всі завдання підсистем та ПЗ реалізуються одночасно, що сприяє встановленню стабільних зв'язків між ними;
- повністю розроблене ПЗ з документацією на нього легше супроводжувати,

тестувати, фіксувати помилки і вносити зміни не безладно, а цілеспрямовано, починаючи з вимог і повторити процес.

Недоліки та фактори ризику каскадної моделі ЖЦ ПЗ відображені на рис.1.2. Каскадну модель можна розглядати як модель ЖЦ, придатну для створення першої версії ПЗ з метою перевірки реалізованих в ній функцій. При супроводі та експлуатації можуть бути виявлені різного роду помилки, виправлення яких потребують повторного виконання всіх процесів, починаючи з уточнення вимог.

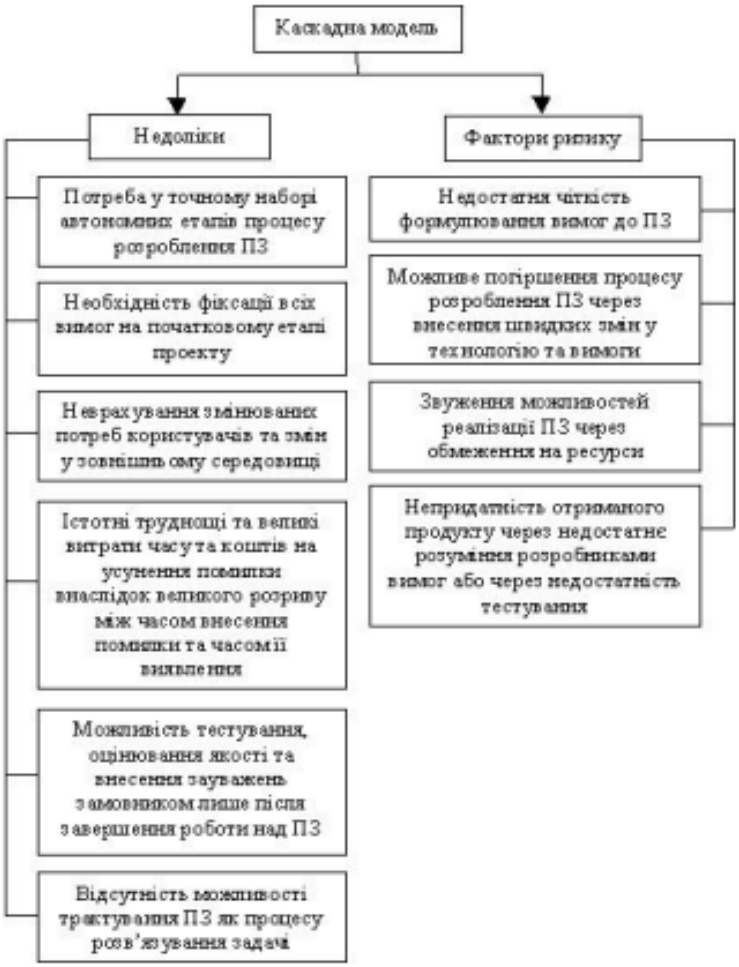


Рис.1.2 - Недоліки та фактори ризику каскадної моделі

В основу *інкрементної моделі ЖЦ ПЗ* (рис.1.3) покладено наступну концепцію: перша створювана проміжна версія ПЗ (випуск 1) реалізує частину вимог, в наступну версію (випуск 2) додають додаткові вимоги і так доти, доки не будуть остаточно виконані всі вимоги та вирішені завдання розроблення ПЗ. Для кожної проміжної версії на етапах ЖЦ виконуються необхідні процеси, роботи та завдання, в тому числі аналіз вимог та створення нової архітектури, які можуть бути виконані одночасно.

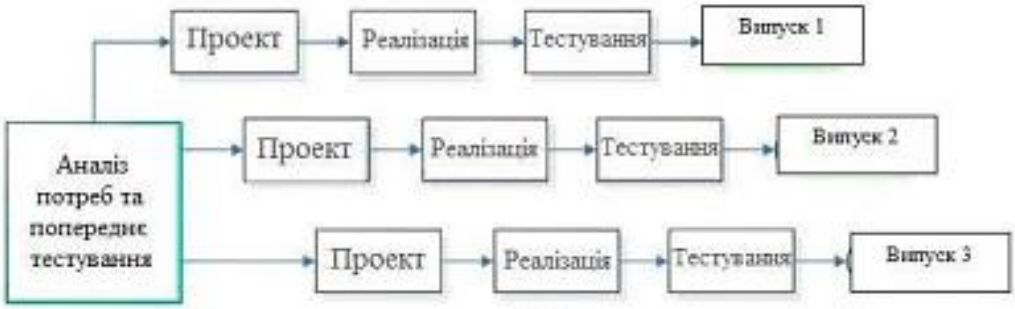


Рис.1.3 - Інкрементна модель ЖЦ програмного забезпечення

Фактори ризику, які можуть мати місце при застосуванні інкрементної моделі ЖЦ ПЗ наведені на схемі 1.

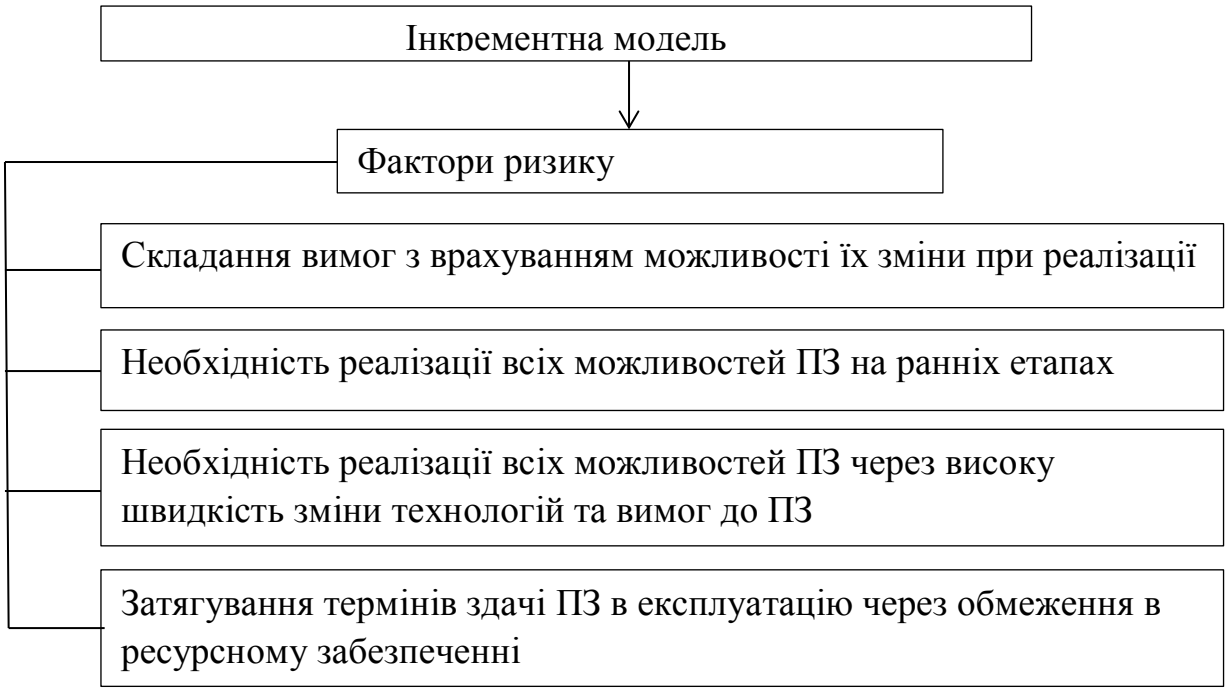


Схема 1. Фактори ризику інкрементної моделі

Наразі інкрементна модель частіше розглядається в контексті поступового нарощування функціональностістворюваного продукту.

Інкрементну модель ЖЦ доцільно використовувати у випадках, коли:
- бажано реалізувати деякі можливості ПЗ швидко зарахунок створення проміжної версії продукту;

- система розбивається на окремі складові частини, якіможна реалізовувати як деякі самостійні проміжні або готові продукти;

- можливе збільшення фінансування на розроблення окремих частин ПЗ. Виходячи з можливості внесення змін, як до процесу, так і в створюваний проміжний продукт була створена *спіральна модель ЖЦ ПЗ* (spiral model) - рис.1.5. Спіральна модель розроблена Б.Боемом у середині 80-х років XX століття. Внесення змін у спіральній моделі орієнтоване на задоволення потреби користувачів одразу, як тільки буде встановлено, що створені артефакти або елементи документації (опис вимог проекту, коментарі різного виду і т.і.) не відповідають дійсному стану розроблення.

Дана модель ЖЦ припускає аналіз продукту на витку розроблення, його перевірку, оцінювання правильності та прийняття рішення переходити на наступний виток або спуститися на попередній виток для доопрацювання на ньому проміжного продукту.

Недоліком спіральної моделі є важкість визначення моменту переходу розроблення на наступний етап.

Серед переваг спіральної моделі слід відмітити спрощення внесення змін в проект при зміні вимог замовника, оскільки елементи ПЗ інтегруються поступово. Спіральна модель життєвого циклу дає можливість визначати якість вже на етапі проектування.

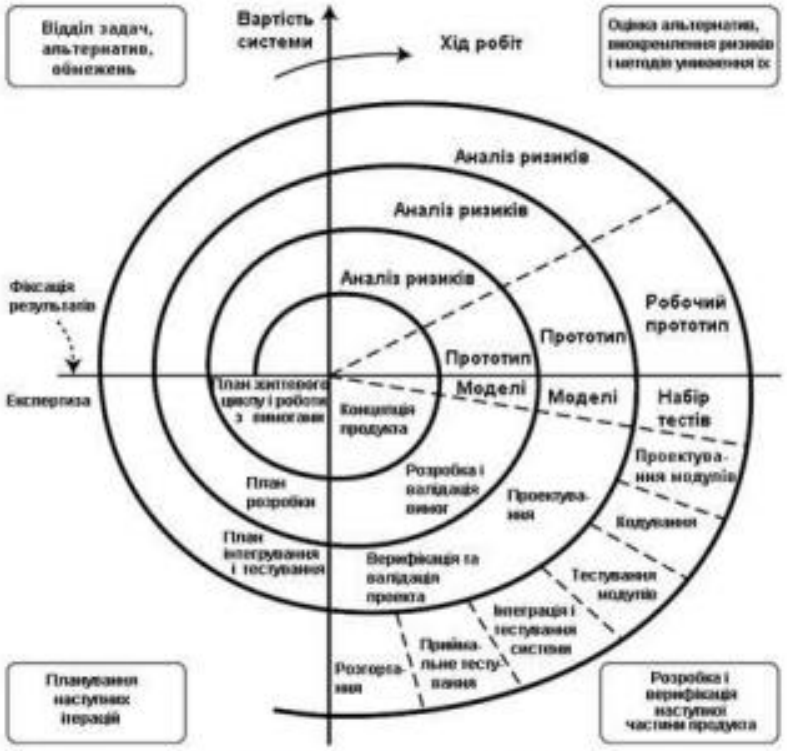


Рис.1.5 - Спіральна модель ЖЦ програмного забезпечення

У випадку *еволюційної моделі ЖЦ* програмне забезпечення розробляється у вигляді послідовності блоків структур (конструкцій). На відміну від інкрементної моделі ЖЦ, вимоги встановлюються частково і уточнюються в кожному наступному проміжному блоці структури ПЗ. Використання еволюційної моделі припускає проведення дослідження предметної області для вивчення потреб замовника проекту та аналізу можливості застосування цієї моделі для реалізації. Модель застосовується для розробки нескладних і не критичних систем, для яких головною вимогою є реалізація функцій системи. При цьому вимоги не можуть бути визначені одразу і повністю.

Розвитком цієї моделі є *модель еволюційного прототипування* (рис.1.6). В літературі вона часто називається моделлю швидкого розроблення додатків *RAD* (Rapid Application Development). У даній моделі наведені дії, пов'язані з аналізом її застосування для конкретного виду ПЗ, а також спостереження про замовника для визначення потреб користувача з метою розроблення плану створення прототипу. У моделі є дві головні ітерації розроблення функціонального прототипу, проектування та реалізації ПЗ. Перевіряється, чи задовольняє ПЗ всім функціональним і не функціональним вимогам. Основною ідеєю цієї моделі є моделювання окремих функцій ПЗ в прототипі і поступове еволюційне його доопрацювання для виконання всіх заданих функціональних вимог.

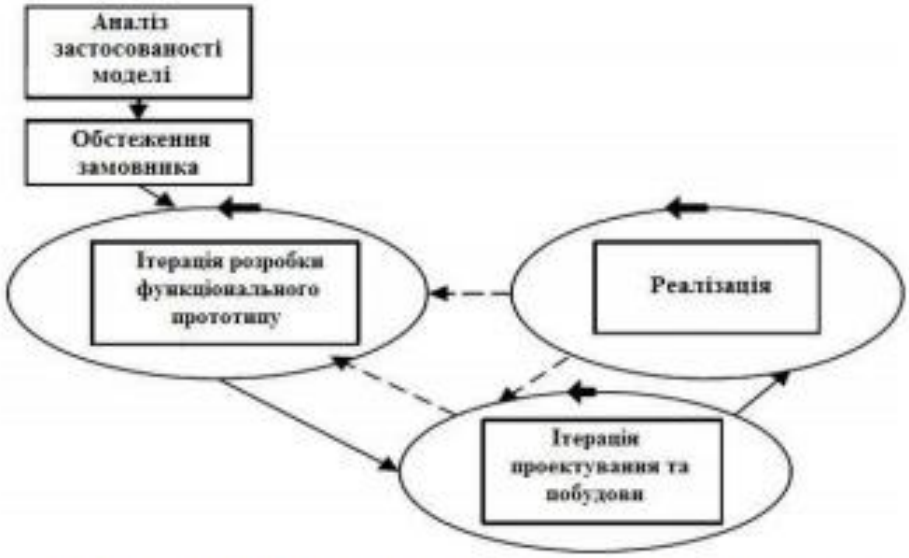


Рис.1.6 - Модель еволюційного прототипування

При використанні еволюційної моделі слід враховувати фактори ризику, наведені на рис.1.7.

Рис.1.7 - Фактори ризику еволюційної моделі

Переваги застосування даної моделі ЖЦ наступні:
- швидка реалізація деяких функціональних можливостей ПЗ та перевірка їх роботоздатності;
- використання проміжного продукту в наступному прототипі;
- виділення окремих функціональних частин для реалізації їх у вигляді прототипу;
- зворотній зв'язок із замовником для уточнення функціональних вимог;
- спрощення внесення змін в зв'язку з заміною окремої функції;
- дозволяє знизити ступінь невизначеності із завершенням кожної ітерації циклу;
- тестування продукту можна починати вже на ранніх стадіях життєвого циклу.

Вибір прийнятної моделі життєвого циклу розроблення ПЗ для проекту може здійснюватись за наступним алгоритмом:

1) Аналіз таких відмінних категорій проекту (таблиці 1-4), як: вимоги, колектив розробників, колектив користувачів, тип проекту та ризики;

2) Відповіді на питання кожної категорії;

3) Розташування категорій та питань за ступенем важливості відносно проекту, для якого обирається модель ЖЦ.

Категорія *вимог* (таблиця 1.1) складається з питань відносно вимог, які висуває користувач до проекту, тобто відносно властивостей програмної системи, яка підтримується даним проектом.

Таблиця 1.1 – Вибір моделі ЖЦ на основі вимог

Питання	**Каскадна**	**Спіральна**	**RAD**	**Інкрементна**
Чи є вимоги добре відомими або такими, що легко визначаються?	Так	Ні	Так	Ні
Чи можуть вимоги визначатись у циклі?	Так	Ні	Так	Так
Чи часто змінюватимуться вимоги в циклі?	Ні	Так	Ні	Ні
Чи потрібно демонструвати вимоги?	Ні	Так	Так	Ні
Чи потрібна перевірка концепції для демонстрації можливостей?	Ні	Так	Так	Ні
Чи будуть вимоги відображати складність програмної системи?	Ні	Так	Ні	Так
Чи володіє вимога функціональними властивостями на ранньому етапі?	Ні	Так	Так	Так

При можливості у склад *колективу розробників* краще відібрати персонал ще до того, як буде обиратись модель ЖЦ.

Характеристики такого колективу (таблиця 1.2) відіграють важливу роль у процесі вибору, оскільки саме цей колектив несе відповідальність за вдале виконання циклу.

Таблиця 1.2 – Вибір моделі ЖЦ на основі характеристик учасників колективу розробників

Питання	**Каскадна**	**Спіральна**	**RAD**	**Інкрементна**
Чи є проблеми предметної галузі проекту новими для більшості розробників?	Ні	Так	Ні	Ні
Чи є технологія предметної галузі проекту новою для більшості розробників?	Так	Так	Ні	Так
Чи є інструменти проекту новими для більшості розробників?	Так	Так	Ні	Ні
Чи змінюються ролі учасників проекту?	Ні	Так	Ні	Так
Чи можуть розробники проекту навчатись?	Ні	Ні	Так	Так
Чи є структура більш значущою для розробників, ніж гнучкість?	Так	Ні	Ні	Так
Чи буде менеджер проекту суворо відстежувати прогрес команди?	Так	Так	Ні	Так
Чи важлива легкість розподілу ресурсів?	Так	Ні	Так	Так
Чи сприймає колектив рівноправні огляди?	Так	Так	Ні	Так

На початкових фазах проекту можна одержати чітке уявлення про *колектив користувачів* (таблиця 1.3) та його майбутній взаємозв’язок з колективом розробників протягом всього проекту. Таке уявлення допоможе при виборі прийнятної моделі ЖЦ, оскільки деякі моделі вимагають посиленої участі користувачів у процесі розроблення та вивчення проекту.

Таблиця 1.3 – Вибір моделі ЖЦ на основі характеристик колективу користувачів

Питання	Каскадна	Спіральна	RAD	Інкрементна
Чи є присутність користувачів обмежена?	Так	Так	Ні	Так
Чи будуть користувачі знайомі з визначенням програмної системи?	Ні	Так	Ні	Так
Чи будуть користувачі ознайомлені з проблемами предметної галузі?	Ні	Ні	Так	Так
Чи користувачі задіяні на всіх етапах ЖЦ?	Ні	Ні	Так	Ні
Чи відстежуватиме замовник хід проекту?	Ні	Так	Ні	Ні

Уточнимо тепер, що собою являють *тип проекту та ризики* (таблиця 1.4), які є елементами, визначення яких виконується на 17 етапі планування. В деяких моделях передбачений менеджмент ризиків високого степеня, в той час як в інших він не передбачений взагалі. Вибір моделі, яка робить можливим менеджмент ризиків, не означає, що не потрібно складати план дій, спрямований на мінімізацію виявлених ризиків. Така модель просто забезпечує схему, за якої можно обговорити та виконати даний план дій.

Таблиця 1.4 – Вибір моделі ЖЦ на основі характеристик типу проекту та ризиків

Питання	Каскадна	Спіральна	RAD	Інкрементна
Чи буде проект ідентифікувати новий напрямок продукту для організації?	Ні	Так	Ні	Так
Чи буде проект мати системну інтеграцію?	Ні	Так	Так	Так
Чи є проект розширенням існуючого ПЗ?	Ні	Ні	Так	Так
Чи буде фінансування проекту стабільним?	Так	Ні	Так	Ні
Чи очікується тривала експлуатація програмної системи в організації?	Так	Так	Ні	Так
Чи потрібен високий ступінь надійності?	Ні	Так	Ні	Так
Чи буде ПЗ змінюватись непередбаченими методами на етапі супроводу?	Ні	Так	Ні	Так
Чи є графік обмеженим?	Ні	Так	Так	Так
Чи є «прозорими» інтерфейсні модулі?	Так	Ні	Ні	Так
Є повторно використовувані компоненти?	Ні	Так	Так	Ні
Чи є достатніми ресурси?	Ні	Так	Ні	Ні

Розподіл фінансових і часових витрат на реалізацію кожного з етапів розробки програмного забезпечення.

При оцінюванні вартості проекту та кількості часу, потрібного для його виконання, слід виконати два основних етапи:

1) оцінювання розміру проекту;
2) визначення загального обсягу;
3) трудовитрат та, відповідно, вартості робіт.

Одержання оцінки можливого *розміру ПЗ* значно спрощується в процесі виконання робіт в рамках програмного проекту. Якщо є потреба у визначенні розміру на початковому етапі життєвого циклу, коли для оперування доступним є лише невеликий обсяг відомостей, то ця оцінка буде мати прогнозний характер. Але саме на базі такої ранньої оцінки робиться загальний висновок, чи варто займатись даним

проектом надалі, на яких умовах слід заключати контракт на його виконання. Саме завдяки ранній оцінці можливого розміру ПЗ можна говорити про розмір планованих трудовитрат.

В якості одиниць вимірювання *розмірів програмного проекту* можуть виступати різні величини, вибір яких залежить від конкретного проекту та потреб організації. Розмір ПЗ визначається у термінах рядків коду (LOC – Lines of Code), функціональних точок та точок властивостей. Також можна враховувати кількість «жирних» точок на діаграмі потоку даних, оцінювати кількісні характеристики специфікацій процесу/керування, кількість основних записів та/або взаємозв'язків на діаграмі взаємозв'язку сутностей (ERD).

Для знаходження *очікуваної LOC-оцінки* - за кожною функцією експерти надають краще, гірше та імовірне значення LOC-оцінки, тоді очікувана LOC-оцінка функції обчислюється за формулою: $LOCочі = (LOCкращ_i + LOCгірш_i + 4 \times LOCімов_i)/6$.

Відповідно, очікувана LOC-оцінка всієї програми буде сумою очікуваних LOC-оцінок всіх її функцій. Показник LOC залежить від мови програмування.

Це оціночна і необов'язкова метрика. Вона може призвести до оптимізації кількості рядків коду, а не проекту в цілому. Дозволяє спрогнозувати трудовитрати, час і вартість розробки, а також необхідну кількість програмістів для виконання проекту. Є вільно поширювані інструменти очікуваної LOC- оцінки [http://www.construx.com]. LOC-оцінка впливає на оцінку величини змін обсягу коду в часі.

На основі розміру проекту (в основному, на основі LOC-оцінок) можна обчислити вартість, трудовитрати та тривалість програмного проекту.

Очікувана вартість розробки кожної функції: $ВАРТІСТЬ_i = LOC_{оч_i} \times ВАРТІСТЬ_{рядка_i}$; вартість рядка є константою і не змінюється від реалізації до реалізації. Відповідно вартість ПЗ є сумою вартостей розробки всіх його функцій.

Прогнозовані витрати на розробку кожної функції: $ВИТРАТИ_i = (LOC_{оч_i} / ПРОДУКТ_i)$. І відповідно витрати на ПЗ є сумою витрат на розроблення всіх його функцій.

Прогнозована оцінка трудовитрат та тривалості проекту за моделлю Боема - трудовитрати на розробку програмних додатків зростають швидше, ніж розмір додатків. Для представлення даного співвідношення використовується експоненційна функція.

Тривалість проекту за моделлю Боема зростає експоненційно разом з докладеними до проекту зусиллями.

$$Трудовитрати = a \times KLOC^b;$$

$$Тривалість = c \times Трудовитрати^d = c \times (a \times KLOC^b)^d = c \times a^d \times KLOC^{b \times d},$$

де *KLOC* - кількість тисяч рядків коду, *a*,*b*,*c*,*d* – коефіцієнти COCOMO.

Для органічного (самостійного) програмного проекту: $a = 2.4, b = 1.05, c = 2.5, d = 0.38$.

Для вбудованих програмних проектів (інтеграція апаратного та програмного забезпечення): $a = 3.6, b = 1.20, c = 2.5, d = 0.32$.

Для проміжних програмних проектів (не органічні, але й не жорстко вбудовані):

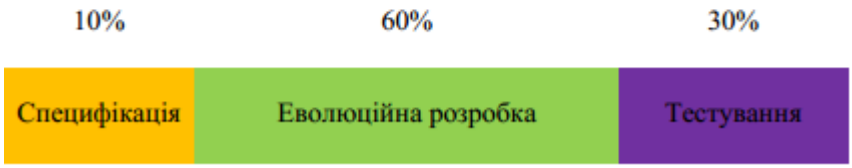
$a = 3.0, b = 1.12, c = 2.5, d = 0.35$.

На початкових етапах життєвого циклу для оперування доступним є лише невеликий обсяг відомостей, тому оцінки розміру, вартості, тривалості, трудовитрат програмного проекту мають прогнозний характер. Але саме на основі таких ранніх оцінок робиться загальний висновок, чи варто займатись даним проектом надалі.

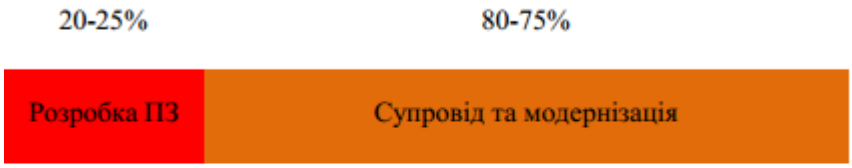
Розподіл часу та витрат на різні етапи життєвого циклу ПЗ представлений нижче.

Окремий розрахунок витрат на кожен етап виробництва ПЗ:

Структура витрат при використанні еволюційного ЖЦ



Врахування вартості супроводу контрактного ПЗ



ПЗ для ПК