МІНІСТЕРСТВО ОСВІТИ І НАУКИ
ДВНЗ "Ужгородський національний університет"
Факультет інформаційних технологій
Кафедра програмного забезпечення систем

В. М. Коцовський

Супровід програмних систем

Методичний посібник

Ужгород – 2016

Коцовський В.М. **Супровід програмних систем:** Методичний посібник для студентів спеціальності "Інженерія програмного забезпечення" / В. М. Коцовський. — Ужгород: Видавництво УжНУ "Говерла", 2016. — 52 с.

У методичному посібнику до курсу "Супровід програмних систем" наведено основні поняття та методи програмної інженерії, які стосуються супроводу програмних систем. Посібник може бути основою для підготовки студентів до практичних занять та модульного і залікового контролю.

Розробник: Коцовський В. М., к. т. н., доцент кафедри програмного забезпечення систем ДВНЗ "Ужгородський національний університет".

Рецензенти:
Гече Й. І., д. т. н., завідувач кафедри кібернетики і прикладної математики ДВНЗ "Ужгородський національний університет";

Міца О. В., к. т. н., завідувач кафедри інформаційних управляючих систем та технологій ДВНЗ "Ужгородський національний університет".

Рекомендовано кафедрою програмного забезпечення систем.
Протокол № 2 від 12 жовтня 2016 року.

Рекомендовано Вченою радою факультету інформаційних технологій.
Протокол № 2 від 13 жовтня 2016 року.

ЗМІСТ

ВСТУП

Нормативна навчальна дисципліна "Супровід програмних систем" вивчається студентами спеціальності "інженерія програмного забезпечення " у IX семестрі.

Актуальність вивчення дисципліни зумовлена тим, що процеси супроводу є важливим компонентом життєвого циклу програмних систем. Супровід необхідний для забезпечення вимог користувачів протягом усього періоду експлуатації програмних продуктів.

Предметом дисципліни є процеси розробки та супроводу інформаційних систем, технології забезпечення та контролю якості програмних систем.

Мета викладання дисципліни — ознайомлення з методами супроводу програмних систем та засобами підтримки їх якісної розробки.

Завдання викладання дисципліни — вироблення у студентів навичок роботи інженера з підтримки та контролю якості програмних систем, формування вмінь супроводу програмних продуктів.

За результатами вивчення курсу "Супровід програмних систем" студенти повинні

знати:

- основні визначення та моделі якості програмного забезпечення (ПЗ);
- вимоги міжнародних та національних стандартів якості та супроводу ПЗ, основи інженерії якості;
- моделі і метрики якості програмних систем;

вміти:

- забезпечувати основні функції інженера з підтримки та контролю якості програмних систем;
- реалізувати перевірку якості програмного продукту;
- забезпечити функцію супроводу та підтримки програмного продукту;
- забезпечити функцію еволюціонування програмного продукту.

1. Інфраструктура програмної інженерії

Програмна система (ПС) — група інтегрованих програмних засобів, які підтримують певний діловий процес споживача (або його частину) і використовують загальне сховище даних.

Термін *програмне забезпечення* (ПЗ) використовується до сукупності програмних засобів, які розробляються з метою експлуатації у складі системи.

Основа якісної розробки програмних систем — раціональна інфраструктура програмної інженерії як виду бізнесу.

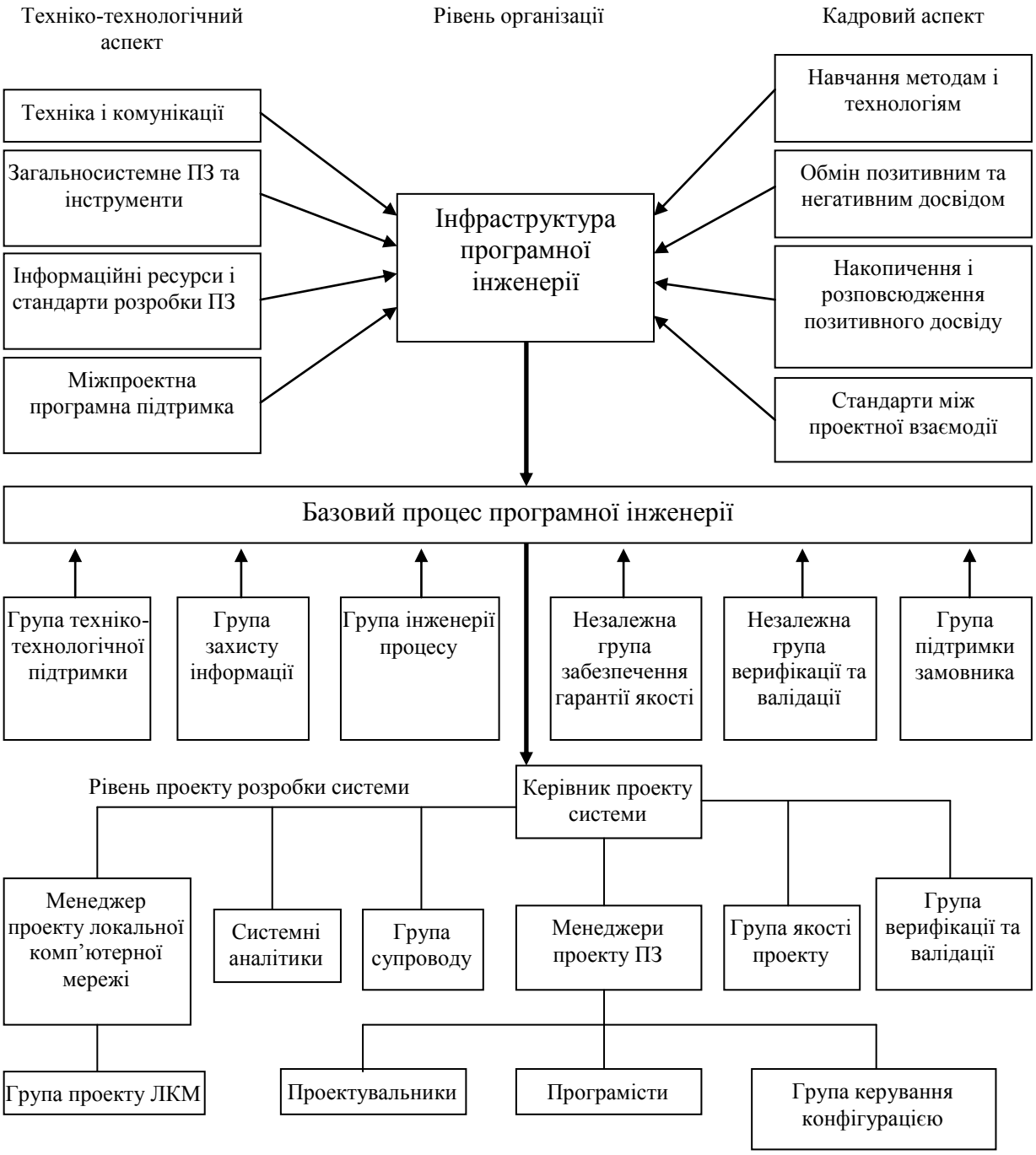

Рис 1. Діаграма інфраструктури програмної інженерії

Інфраструктура програмної інженерії — інтегрований набір загальнодоступних технічних, технологічних і методологічних ресурсів організації розробника, які роблять

можливим виконання процесу програмної інженерії колективами проектів, які відкриваються по договорах із замовниками. Структурна діаграма інфраструктури програмної інженерії наведена на рис. 1.

Проект — це обмежена часовими рамками діяльність, мета якої полягає в створенні унікального програмного продукту.

1.1. Компоненти інфраструктури розробки ПС.

1.1.1. Техніко-технологічний аспект.

1. Техніка і комунікації:
   - Комп'ютери користувачів, файлові сервери
   - Локальні комп'ютерні мережі (ЛКМ)
   - Глобальна комп'ютерна мережа (ГКМ)
   - Електронна пошта
   - Техніка для тестування
   - Офісна техніка
   - Інші складові комплексу технічних засобів
2. Загальносистемне ПЗ та інструменти:
   - Клієнт-серверні технології
   - Операційні системи
   - Офісні системи
   - Системи документообігу
   - Утиліти
   - Засоби захисту інформації (антивіруси)
   - CASE-інструменти, системи програмування
   - СУБД
   - Графічні інструменти
3. Інформаційні ресурси і стандарти розробки:
   - Методології розробки
   - Інструменти керування проектами, конфігураціями
   - Системи підтримки використання ресурсів Інтернет
   - Нормативні документи, які стосуються технічних, програмних, комунікаційних засобів, даних і захисту інформації
   - Нормативні документи оформлення матеріалів
   - Методичні матеріали, шаблони і заготовки документів
4. Міжпроектна програмна підтримка
   - Розроблені програми (модулі), визнані здатними до загального користування, документовані та поміщені під контроль конфігурації.

1.1.2. Кадровий аспект.

1. Навчання методам і технологіям:
   - Можливості організації по навчанню спеціалістів методам та прийомам розробки ПЗ
   - Можливості вивчення спеціалістами техніко-технологічних компонент інфраструктури
2. Обмін позитивним та негативним досвідом:

- Культура «відкритого» сприйняття/передачі набутого досвіду, знань, характерних помилок. Сприяння розповсюдженню позитивного досвіду. Не приховування власних помилок і не перекладання відповідальності за них. Бажання навчатись/навчати

3. Накопичення і закріплення позитивного досвіду:
   - Визначення форматів і засобів накопичення і зберігання здобутого досвіду (опитування, семінари тощо)
   - Створення бібліотек активів організації за принципом «кращий об'єкт». Включення їх у сферу керування конфігурацією. Забезпечення доступності.
4. Стандарти міжпроектної взаємодії:
   - Визначення стандартів (меж компетенції, знань) по процесам життєвого циклу (ЖЦ) ПС. Уніфікація та стандартизація прийомів роботи з метою побудови і підтримки базового процесу програмної інженерії
   - Профілювання знань для забезпечення замінюваності спеціалістів в проекті. Дотримання принципу «глибокі знання у вузькій сфері»

1.2. Ролі спеціалістів в організаційній структурі розробки

1.2.1. Ролі на рівні організації

1. Група техніко-технологічної підтримки:
   - Вивчення ринку послуг і попиту в організації відносно техніки та загальносистемного ПЗ.
   - Придбання/встановлення/підтримка техніки.
   - Придбання/встановлення/підтримка загальносистемного ПЗ.
   - Навчання/консультаційні послуги співробітникам.
   - Рекомендації по застосуванню техніки і технологій в проектах.
2. Група захисту інформації:
   - Вивчення стану справ в області захисту інформації і накопичення досвіду.
   - Забезпечення захисту інформації в організації.
   - Перевірка захисту інформації в організації.
   - Підтримка проектів в питаннях захисту інформації.
3. Група інженерії процесу
   - Визначення, супровід та вдосконалення базового процесу програмної інженерії. Забезпечення нормативно-методичної підтримки виконання процесів ЖЦ. Організація та поповнення сховища (бібліотеки) активів організації.
   - Допомога менеджерам проектів в адаптації базового процесу до потреб проектів. Підбір або виготовлення форм (шаблонів) документів для інженерії проектів.
   - Підтримка процесу документування в проектах, зокрема виконання важких графічних робіт, оформлення документів згідно стандартів оформлення. Нормоконтроль та друк документів.
   - Міжпроектна координація в частині накопичення досвіду і організації навчання.
   - Підтримка керування конфігурацією в проектах.
4. Незалежна група якості (SQA-група):

- Планування та виконання дій по контролю і гарантії дотримання дисципліни створення програмної продукції в проектах (організація перевірок робіт в контрольних точках проектів, визначених календарними планами).
- Контроль документів і продуктів ПЗ в контрольних точках проектів на предмет дотримання діючих стандартів та інших нормативних документів, встановлених у вимогах замовника.
- Звітність безпосередньо перед керівником організації

5. Незалежна група верифікації та валідації (V&V-група):
   - Виконання функції верифікації (по домовленості з групою SQA).
   - Планування і проведення незалежного кваліфікаційного тестування інтегрованих компонент ПЗ або програмних продуктів з метою визначення їх відповідності потребам замовника.
   - Координація планів робіт з менеджерами проектів відносно вимог до тестового середовища, строків і порядку передачі ПЗ на тестування.
   - Представлення звітів (результатів) тестування менеджерам проектів для прийняття мір по виправленню ПЗ.
   - Незалежність від менеджерів проектів в частині визначення об’ємів і методів тестування.
   - Звітність перед керівником організації за дотримання порядку тестування і стан розроблених програмних продуктів.
6. Група підтримки замовника:
   - Зв’язок із замовником з питань автоматизації ділових процесів.
   - Підтримка процесів керування вимогами, навчання користувачів, супроводу (або допомога в їх виконанні на рівні окремих проектів).

1.2.2. Ролі на рівні проекту

1. Керівник проекту системи:
   - Повна фінансова відповідальність за виконання проектних домовленостей перед замовником.
   - Керування розробкою складових створюваної продукції – проектів ПЗ, комплексу технічних засобів, засобів захисту інформації.
   - Відповідальність за дії виконавців проекту.
2. Системні аналітики:
   - Дослідження умов та потреб автоматизації діяльності організації-споживача.
   - Системний аналіз вимог споживача і формування концепції системи.
   - Контроль обґрунтованості проектних рішень, що приймаються.
3. Група якості проекту:
   - Контроль якості робочих продуктів, створених процесами ЖЦ (на відповідність стандартам, методикам тощо).
   - Звітність тільки керівнику проекту.
   - Може бути відсутньою, якщо на рівні організації діє незалежна група якості.
4. Група V&V проекту:
   - Перевірка відповідності робочих продуктів, вироблених на певному етапі ЖЦ, вимог до них, встановлених на попередньому етапі.
   - Може виконувати тестування окремих компонент ПЗ, а також системне (інтеграційне) тестування ПЗ, виробленого в проекті.

- Звітність тільки керівнику проекту.

5. Менеджер проекту ПЗ:
   - Повна відповідальність за усі проектні рішення та дії, пов’язані з розробкою ПЗ в проекті
   - Підбір і контроль ресурсів проекту, а також графіка робіт.
   - У великих або розподілених програмних проектах може бути декілька менеджерів (по підсистемам або рівням проекту ПЗ).
6. Проектувальники:
   - Прийняття і документування проектних рішень по архітектурі і функціям ПЗ. Узгодження рішень з менеджером проекту ПЗ.
   - Дотримання стандартів якості (забезпечення досягнення характеристик якості).
7. Програмісти:
   - Програмування або моделювання компонентів ПЗ по проектним специфікаціям, підготованих проектувальниками.
   - Дотримання стандартів якості при програмуванні (по зручності супроводу коду, зручності застосування програм).
   - Відладка та автономне тестування розроблених компонент.
8. Група керування конфігурацією:
   - Виконання процесу конфігураційного керування версій ПЗ і робочих продуктів проекту ПЗ.
9. *Група супроводу:*
   - Виконання процесу супроводу версій ПЗ і робочих продуктів проекту ПЗ під час дослідної експлуатації і під час встановленого періоду супроводу.
   - Навчання користувачів.
   - Виконання процесу розв’язання проблем.
   - Можуть бути членами групи підтримки замовника.
10. Група проекту ЛКМ:
    - При розробці системи «під ключ» проектування і монтаж ЛКМ для встановлення в організації споживача.
    - Закупівля і встановлення КТЗ і загальносистемного ПЗ, пуско-налагоджувальні дії.

1.3. Архітектура процесів життєвого циклу програмної системи

Процес програмної інженерії має ієрархічну структуру і включає множину процесів ЖЦ програмної системи. Вимоги до процесів ЖЦ ПС визначає міжнародний стандарт ISO/IEC 12207, а в Україні йому відповідає ДСТУ 3918-99. Процеси ЖЦ розподілені по трьом групам, які відображають функціональну направленість видів діяльності, які ці процеси регламентують:

- Основні процеси.
- Процеси підтримки.
- Організаційні процеси.

1.3.1. Основні процеси ЖЦ:

1. Придбання
   1.1.Підготовка придбання.

1.2. Вибір постачальника.
1.3. Моніторинг діяльності постачальника.
1.4. Прийом споживачем.
2. Поставка
2.1. Участь в тендері.
2.2. Укладення договору.
2.3. Випуск продукту (релізу).
2.4. Підтримка приймання продукту.
3. Розробка
3.1. Виявлення вимог.
3.2. Аналіз вимог до системи.
3.3. Проектування архітектури системи.
3.4. Аналіз вимог до ПЗ системи.
3.5. Проектування ПЗ.
3.6. Програмування ПЗ.
3.7. Інтеграція ПЗ.
3.8. Тестування ПЗ.
3.9. Системна інтеграція.
3.10. Системне тестування.
3.11. Інсталяція ПЗ.
4. Експлуатація
4.1. Функціональне використання.
4.2. Підтримка споживача.
5. *Супровід.*

1.3.2. Процеси підтримки

Процеси підтримки інтегруються з будь-якими іншими процесами, розв'язуючи задачі, допоміжні по відношенню до задач цих процесів, і забезпечують якість їх розв'язання в конкретних проектах.

1. Документування.
2. Керування конфігурацією.
3. Забезпечення гарантії якості.
4. Верифікація.
5. Валідація.
6. Сумісний перегляд.
7. Аудит.
8. Керування розв'язанням проблем.
9. Керування запитами на зміну.
10. Забезпечення застосовуваності продукту (підтримка користувача).
11. Оцінювання проекту.

1.3.3. Організаційні процеси ЖЦ

1. Керування
1.1. Організаційне будівництво (формування корпоративної політики, цілей, процесів, стандартів, етики).
1.2. Управління організацією.
1.3. Управління проектом.

1.4. Управління якістю.
1.5. Управління ризиком.
1.6. Вимірювання.
2. Підтримка інфраструктури.
3. Вдосконалення
3.1. Установлення процесів.
3.2. Оцінювання процесів.
3.3. Покращення процесів.
4. Забезпечення трудовими ресурсами
4.1. Управління кадрами.
4.2. Навчання.
4.3. Управління знаннями (розповсюдження знань).
5. Управління активами організації.
6. Управління програмою повторного використання.
7. Доменна інженерія.

1.4. Базовий процес організації

Сучасною концепцією процесу програмної інженерії є побудова базового процесу організації (БПО), який розробляється, супроводжується, оцінюється і покращується подібно до того як розробляються програмні продукти.

БПО є основою для визначення процесів усіх програмних проектів. Процеси на рівні проектів розробляються шляхом адаптації БПО до характеристик конкретного проекту. Визначення БПО – це формалізований опис складу процесів ЖЦ, з яких мають побудуватись процеси розробки в програмних проектах, а також взаємозв'язків між елементами цих процесів.

З кожним процесом розробки пов'язуються:

- *Вимоги* до процесу, які вказують, «що» собою являє процес (що він буде робити).
- *Архітектура* і проект процесу, які описують, «як» процес буде визначений (які будуть елементи процесу і як будуть пов'язані).
- *Реалізація* опису процесу в рамках організації програмного проекту (створення елементі процесу і встановл. інтерфейсу).
- *Перевірка* і затвердження визначення процесу.
- *Впровадження* процесу в середовище конкретного проекту.

При побудові БПО керівники організації-розробника ПС спочатку визначають склад основних процесів із числа рекомендованих, а потім підтримуючих і організаційних процесів. В своїх рішеннях керуються доцільністю включення певного процесу БПО з позицій необхідності виконання регламентованих процесом дій та їх достатності для досягнення цілей розробки якісного програмного продукту.

В мінімальній конфігурації процесів ЖЦ, крім процесу управління проектом, в БПО обов'язково має знайтись місце для процесу перевірки, а також процесу управління конфігурацією.

Побудований БПО має підтримувати використання моделей ЖЦ, застосування яких допускається в проектах організації.

2. Ядро знань із програмної інженерії (SWEBOK)

Для створення ядра знань в програмній інженерії в 1993 році сумісними зусиллями ACM (Association for Computing Machinery) та IEEE був створений спеціальний комітет SWECC (Software engineering coordination committee). В рамках цього комітету були організовані групи по наступним напрямам досліджень:

1. Визначення необхідного ядра знань і рекомендованих практичних засобів діяльності в програмній інженерії.
2. Визначення норм професійної етики і стандартів з програмної інженерії.
3. Визначення програм навчання студентів ВНЗ із спеціальності.

2.1. Ядро SWEBOK

Ядро SWEBOK складають знання з десяти різних областей знань:

1. Програмні вимоги.
2. Проектування (дизайн) ПЗ.
3. Конструювання ПЗ.
4. Тестування ПЗ.
5. *Супровід ПЗ.*
6. Керування конфігурацією ПЗ.
7. Керування інженерією ПЗ.
8. Процес інженерії ПЗ.
9. Інструменти та методи інженерії ПЗ.
10. Якість ПЗ.

Кожній області знань присвячений окремий розділ, структурований по розділам та рубрикам. Розділи завершуються об’ємними списками літератури по предмету, яка по суті і являє матеріал, що утворює ядро знань.

2.1.1. Програмні вимоги

- Основи вимог.
- Процес інженерії.
- Витяг вимог.
- Аналіз вимог.
- Специфікація вимог.
- Перевірка вимог.
- Практичні міркування.

2.1.2. Проектування ПЗ

- Основи проектування.
- Ключові питання.
- Структура і архітектура.
- Аналіз і оцінка якості проекту.
- Нотації дизайну.
- Стратегії і методи проектування.

2.1.3. Конструювання ПЗ

- Основи конструювання.
- Управління конструюванням.

- Практичні міркування.

Тестування ПЗ

- Основи тестування.
- Рівні тестування.
- Засоби тестування.
- Метрики тестування.
- Процес тестування.

Супровід ПЗ

- Основи супроводу.
- Ключові питання.
- Процес супроводу.
- Практичні засоби.

Управління конфігурацією

- Управління процесом.
- Ідентифікація конфігурації.
- Контроль конфігурації.
- Облік стану конфігурації.
- Аудит конфігурації.
- Управління випуском та поставкою.

Управління інженерією

- Ініціювання та визначення рамок проекту.
- Планування проекту.
- Огляд і оцінка проекту.
- Закриття проекту.
- Вимірювання в інженерії ПЗ.

Процес програмної інженерії

- Реалізація і зміна процесу.
- Визначення процесу.
- Оцінювання процесу.
- Вимірювання процесу і продукту

Інструменти і методи програмної інженерії

- Інструменти розробки:
    - Управління вимогами.
    - Проектування.
    - Конструювання.
    - Тестування.
    - Супровід.
    - Управління конфігурацією.
    - Управління інженерією.
    - Підтримка процесів.
    - Забезпечення якості.
    - Інші інструменти.
- Методи інженерії ПЗ:

- Евристичні методи.
- Формальні методи.
- Методи прототипування.

Якість ПЗ

- Основи якості.
- Процеси керування якістю.
- Практичні міркування.

2.2. Класифікація інструментів по SWEBOK

Інструменти роботи з вимогами:

- Засоби моделювання.
- Засоби трасування.

Інструменти проектування:

- UML.
- Бізнес-проектування.
- Проектування БД.

Інструменти конструювання:

- Редактори програм.
- Компілятори і генератори коду.
- Інтерпретатори.
- Дебаггери.

Інструменти тестування:

- Генератори тестів.
- Засоби виконання тестів.
- Інструменти оцінки тестів.
- Засоби керування тестами.
- Інструменти аналізу продуктивності.

Інструменти супроводу:

- Засоби візуалізації.
- Інструменти ре інженерії.

Інструменти управління конфігурацією:

- Інструменти відслідковування дефектів і проблем.
- Інструменти управління версіями.
- Інструменти зборки та випуску.

Управління інженерією:

- Інструменти планування та відстежування , прогнозування вартості.
- Інструменти керування ризиками.
- Засоби кількісної оцінки.

Інструменти підтримки процесів:

- Інструменти моделювання процесів.
- Засоби керування процесами.
- Інтегровані CASE-середовища і рольові платформи розробки.
- Процес-орієнтовані середовища розробки.

Інструменти забезпечення якості:

- Інструменти інспекції, підтримка оглядів та аудитів.

- Інструменти статичного аналізу.

Додаткові аспекти:

- Засоби інтеграції інструментів: програмні платформи (Java, .NET Framework), платформи розподілених обчислень (CORBA, WebServices).
- Мета інструменти: засоби генерації інших інструментів, компілятор компіляторів тощо.
- Засоби оцінки інструментів.

2.3. Ядро знань по керуванню проектами (PMBOK)

PMBOK визначає 39 процесів ЖЦ проекту, об'єднаних в 5 базових груп процесів і 9 ключових областей знань, типових практично для будь-яких проектів.

Групи процесів:

1. Ініціація
2. Планування
3. Виконання
4. Моніторинг і керування
5. Завершення

Ключові області знань:

1. Управління інтеграцією проекту
2. Управління змістом проекту
3. Управління тривалістю (строками) проекту
4. Управління вартістю проекту
5. Управління якістю проекту
6. Управління людськими ресурсами
7. Управління комунікаціями проекту
8. Управління ризиками проекту
9. Управління закупівлями (поставками) проекту.

Схема взаємодії груп процесів ЖЦ, визначених згідно до PMBOK наведена на рис. 2.

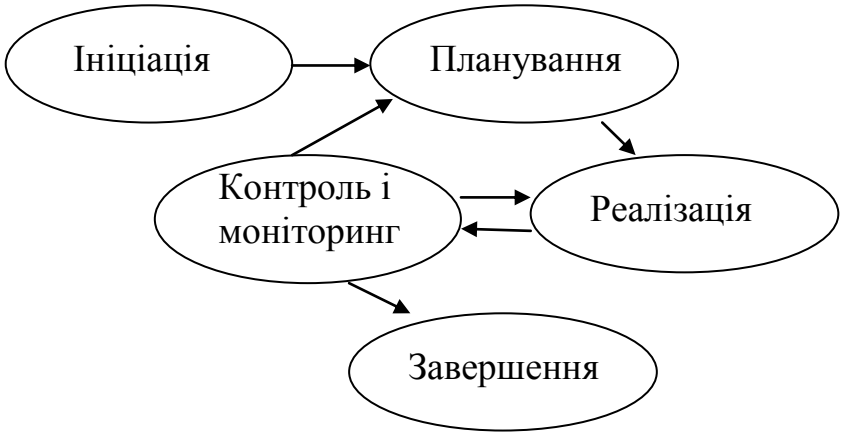

Рис. 2. Схема взаємодії груп процесів

3. Якість програмного забезпечення

Основною проблемою в управлінні якістю є той факт, що визначення якості неясне і неоднозначне. Це викликано тим, що зазвичай термін "якість" розуміють неправильно. Причини:

1. Якість — це не окрема ідея або поняття, а багатомірна і різнопланова концепція.
2. Для будь-якого поняття і визначення існує декілька рівнів абстракції, наприклад, коли люди говорять про якість, одна частина розуміє під цим занадто широкий і розмитий смисл, а інша може вказувати на конкретне визначення.
3. Термін "якість" є невід'ємною частиною нашого повсякденного спілкування, але загальноприйняте і професійне використання може сильно відрізнятись.

3.1. Популярний погляд на якість

Загальноприйнята думка про якість – це дещо нематеріальне і «неосяжне» - про нього можуть вестись суперечки та дискусії, його можна критикувати і вихваляти, але зважити і виміряти неможливо. Вирази типу "хороша якість" і "погана якість" служать наглядним прикладом. Люди так кажуть про щось невизначене для них, що вони не можуть чітко охарактеризувати і визначити. Отже, люди сприймають і інтерпретують якість по різному. Якість не може бути контрольованою і керованою. Якість не може бути кількісно виміряна. Такий погляд різко контрастує з професійним підходом до управління якістю – якість чітко визначена величина, яку можна виміряти і проконтролювати, вона піддається управлінню і покращенню.

Інша популярна думка – якість нерозривно пов'язана з розкішшю, першим сортом і тонким смаком. Дорогий, досконально продуманий і технічно більш складний продукт розглядається як гарантія вищої якості ніж більш дешеві аналоги. За такою логікою Каділлак – якісна машина, а Шевроле – ні, незважаючи на надійність і кількість поломок. Якщо слідувати такому підходу, то якість обмежена визначеним класом дорогих продуктів. Недорогі продукти ж важко віднести до якісних.

3.2. Професійний підхід до якості

Існують чіткі та зручні для роботи визначення.

В 1979 році Ф. Кросбі визначив якість як "відповідність вимогам" ("conformance to requirements").

В 1970 році Юран і Грін визначили якість як "придатність до використання" ("fitness for use"). Ці два визначення тісно пов'язані і добре узгоджуються.

На практиці також використовують наступні визначення якості:

Досягнення відмінного рівня придатності до використання (Хемпфрі);

Компанія IBM — якість, яка управляється ринковими потребами ("market-driven quality").

Ступінь відповідності наявних характеристик вимогам (визначення системи менеджменту якості ISO 9001).

"Відповідність вимогам" припускає, що вимоги мають бути настільки чітко визначені, що вони не можуть бути зрозумілі і інтерпретовані некоректно. Пізніше, на етапі розробки, проводяться регулярні вимірювання розробленого продукту для визначення відповідності вимогам. Будь-які невідповідності розглядаються як дефекти – невідповідність якості. Наприклад, специфікація на деяку модель радіостанції може вимагати можливість приймати визначену частоту радіохвиль на відстані більше ніж за

30 км від джерела віщання. У випадку, якщо радіостанція не може виконати дану вимогу, вона не відповідає вимогам до якості і має бути визнаною негідною і неякісною.

"Придатність до використання" приймає до уваги вимоги і очікування кінцевого користувача, який очікує, що продукт або надаваний сервіс буде зручним для нього. Однак різні користувачі можуть використовувати продукт по різному і це означає, що продукт має володіти максимально різноманітними варіантами використання. Згідно визначенню Юран кожен варіант використання — це характеристика якості і усі вони можуть бути класифіковані по категоріям в якості параметрів придатності до використання.

Ці два визначення якості по суті однакові. Різниця в тому, що варіант "придатність до використання" вказує на важливу роль вимог і очікувань замовника. Роль замовника, пов’язана з якістю, не може бути переоцінена. З точки зору замовника, якість продукту, який він придбав, складається з багатьох факторів: ціна, продуктивність, надійність і т.д. *Тільки замовник може розказати про якість, оскільки це єдине, що він дійсно купляє. Замовник не купляє продукт. Він купляє гарантії того, що усі його очікування до продукту будуть реалізовані.*

Отже, визначення якості з точки зору замовника або користувача продукту:
Якість – це придатність до використання. Чи робить даний продукт те, що мені потрібно? Чи спрощує роботу? Чи можу я його використати так, як мені потрібно?

Точка зору розробника:

Якість – це відповідність специфікованим і забраним вимогам. Чи робить даний продукт усе те, що вказано у вимогах?
Проблема в тому, що специфіковані і забрані вимоги – це зазвичай частина реальних вимог і очікувань замовника.

Отже, якість – це відповідність реальним вимогам, явним і неявним. Дуже часто неявні вимоги настільки очевидні для замовника або користувача, що він навіть не припускає, що вони невідомі розробникам. На прикладі автомобілів: замовник може детально описати вимоги до дизайну, параметрам двигуна, оформленню салону, кольору кузова, але ніде не вказати, що шини мають бути круглими, а лобове скло – прозорим.
Замовник буде повністю задоволеним тільки тоді, коли куплений продукт буде повністю відповідати його реальним і життєвим вимогам – специфікованим і ні.

Існує дві професії пов’язані з якістю програмного забезпечення: тестер (Quality Control) і QA manager.

Контроль якості (Quality Control) це вимірювання якості продукту.

Забезпечення якості (Quality Assurance) – це вимірювання і управління якістю процесу, який використовується для створення якості продукту (або якісного продукту).

Іншими словами:

Quality Assurance — попередження дефектів шляхом перевірки і тестування процесу.

Quality Control — виявлення дефектів шляхом перевірки і тестування продукту.

3.3. Якість ПЗ за МакКолом

Першою широко відомою моделлю якості ПЗ стала запропонована в 1977 МакКолом та іншими модель. В ній характеристики якості розділені на три групи:

Фактори, які описують ПЗ з позицій користувача. Задаються вимогами.

Критерії, які описують ПЗ з позицій розробника. Задаються як цілі.

Метрики, які використовуються для кількісного описання і вимірювання якості.

Фактори якості, яких було виділено 11, групуються в три групи по різним способам роботи людей з ПЗ. Отримана структура відображується у вигляді трикутника МакКола.

Критерії якості – це числові рівні факторів, поставлених як цілей при розробці.

Об'єктивно оцінити або виміряти фактори якості безпосередньо важко. Тому МакКол ввів *метрики якості*, які з його точки зору легше виміряти і оцінити. Оцінки в його шкалі приймають значення від 0 до 10.

Метрики:

- Зручність перевірки на відповідність стандартам (auditability)
- Точність керування і обчислень (accuracy)
- Ступінь стандартності інтерфейсів (communication commonality)
- Функціональна повнота (completeness)
- Однорідність використовуваних правил проектування і документації (consistency)
- Ступінь стандартності форматів даних (data commonality)
- Стійкість до помилок (error tolerance)
- Ефективність роботи (execution efficiency)
- Розширюваність (expandability)
- Широта області потенційного використання (generality)
- Незалежність від апаратної платформи (hardware independence)
- Повнота протоколювання помилок та інших подій (instrumentation)
- Модульність (modularity)
- Зручність роботи (operability)
- Захищеність (security)
- Самодокументованість (selfdocumentation)
- Простота роботи (simplicity)
- Незалежність від програмної платформи (software system independence)
- Можливість співвідношення проекту з вимогами (traceability)
- Зручність навчання (training).

Кожна метрика впливає на оцінку деяких факторів якості. Числовий вираз фактору представляє собою лінійну комбінацію значень впливаючих на нього метрик. Коефіцієнти цього виразу визначаються по різному для різних організацій, команд розробки, видів ПЗ та інш.

3.4. Якість ПЗ за Боемом

У 1978 Боем запропонував свою модель, яка по суті є розширенням моделі МакКола.

Атрибути якості поділяються за способом використання ПЗ (primary use).

Визначено 19 проміжних атрибутів (intermidiate construct), які включають усі 11 факторів якості за МакКолом. Проміжні атрибути розділяються на примітивні, які в свою чергу, можуть бути оцінені за допомогою метрик.

До факторів МакКола Боем додав наступні атрибути:

- ясність (clarity)
- зручність внесення змін (modifiability)
- документованість (documentation)
- здатність до відновлення функцій (resilience)
- зрозумілість (understandability)

- адекватність (validity)
- функціональність (functionality)
- універсальність (generality)
- економічна ефективність (economy).

4. Міжнародний стандарт якості програмних засобів ISO 9126

ISO 9126 ("Інформаційна технологія. Оцінка програмного продукту. Характеристики якості та керівництво по їх застосуванню") — це міжнародний стандарт, який визначає оцінні якості програмного забезпечення. Стандарт поділяється на 4 частини, які описують наступні питання: модель якості, зовнішні метрики якості, внутрішні метрики якості, метрики якості у використанні.

Модель якості, встановлена у першій частині стандарту 9126-1, класифікує якість ПЗ в шести структурних наборах характеристик, які в свою чергу деталізовані підхарактеристиками (субхарактеристиками). Характеристики:

- Функціональність (functionality).
- Надійність (reliability).
- Практичність (зручність застосування) (usability).
- Ефективність (efficiency).
- Супроводжуваність **(**maintainability**).**
- Мобільність (portability).

Розглянемо більш детально кожну з характеристик:

4.1. Функціональність

Функціональність — набір атрибутів, який характеризує відповідність функціональних можливостей ПЗ набору потрібної користувачу функціональності. Деталізується *субхарактеристиками*:

- Придатність до застосування (suitability) – здатність ПС надавати належну множину функцій для розв’язання специфікованих задач і досягнення цілей користувача;
- Коректність (правильність, точність) (accuracy) – здатність ПС забезпечувати правильні або погоджені результати або впливи з необхідним ступенем точності;
- Здатність до взаємодії (в тому числі мережевої) (interoperability) – здатність взаємодіяти з однією чи більше специфікованими системами;
- Захищеність (security) — здатність забезпечувати захист інформації від несанкціонованого доступу осіб або систем і її доступність особам та системам з необхідними правами доступу.

4.2. Надійність

Надійність (reliability) – набір атрибутів, які відносяться до здатності ПЗ зберігати свій рівень якості функціонування в встановлених умовах за визначений період часу. Субхарактеристики:

- Рівень завершеності (відсутність помилок) (maturity) – здатність уникати відмови із-за внутрішніх дефектів;
- Стійкість до дефектів (fault tolerance) – здатність підтримувати встановлений рівень функціонування в умовах прояви дефектів в ПС, помилок даних або порушень специфікованого інтерфейсу;
- Відновлюваність (recoverability) – здатність відновлювати функціонування на заданому рівні і відновлювати пошкоджені програми та дані;
- Доступність;
- Готовність.

4.3. Практичність

Практичність (зручність застосування) (usability) — набір атрибутів, які відносяться до об'єму робіт, які необхідні для виконання та індивідуальної оцінки такого виконання визначеним або передбачуваним колом користувачів. Субхарактеристики:

- Зрозумілість (understandability) – властивості ПС, які забезпечують користувачу можливості зрозуміти, чи дійсно вона може задовільнити його потреби, як вона може бути використана для розв'язання визначених задач і які умови її використання;
- Простота використання (learnability) – властивості ПС, які забезпечують користувачу можливість засвоїти прийоми її застосування;
- Керованість (operability) властивості ПС, які забезпечують користувачу можливість керувати або контролювати її дії;
- Привабливість (attractiveness).

4.4. Ефективність

Ефективність (efficiency) — набір атрибутів, які відносяться до співвідношення між рівнем якості функціонування ПЗ і об'ємом використаних ресурсів при встановлених умовах. Субхарактеристики:

- Часова ефективність (time behavior) - властивості ПС, які спричиняють її здатність забезпечувати належний час відклику (відповіді) і обробки завдань, а також рівень пропускної здатності при виконанні функцій у встановлених умовах застосування;
- Використовуваність ресурсів (resource utilization) властивості ПС, які спричиняють її здатність використовувати ресурси в необхідні періоди часу при виконанні своїх функцій у встановлених умовах застосування.

4.5. Супроводжуваність

Супроводжуваність **(**maintainability) — набір атрибутів, які відносяться до об'єму робіт, які необхідні для проведення конкретних змін (модифікацій). Субхарактеристики:

- Зручність для аналізу (analyzability) властивості ПС, які спричиняють можливість діагностування її недоліків або причин відмов, а також ідентифікації частин, які мають модифікуватись;
- Змінюваність (changeability) — властивості ПС, які спричиняють можливість виконання встановлених видів модифікації;
- Стабільність (stability) — властивості ПС, які спричиняють її здатність мінімізувати неочікувані ефекти модифікацій;
- Зручність для тестування (testability) — властивості ПС, які спричиняють її здатність сприяти перевірці модифікованого ПЗ.

4.6. Мобільність

Мобільність (portability) — набір атрибутів, які відносяться до здатності ПЗ бути перенесеним з одного оточення в інше. Підхарактеристики:

- Здатність до адаптації (adaptability) — властивості ПС, які спричиняють її здатність адаптуватись для застосування в різноманітних специфікованих середовищах без використання дій або засобів відмінних від тих, що спеціально призначені для цих цілей;
- Простота встановлення (installability).

- Співвіснування (відповідність) (co-existence) властивості ПС, які спричиняють її здатність співіснувати з іншими незалежними ПС в спільному середовищі, розділяючи спільні ресурси.
- Здатність до заміщення (replaceability) — властивості ПС, які спричиняють її здатність використовуватись замість інших специфікованих ПС в середовищі їх застосування.

Підхарактеристика "відповідність" не приведена і списку, але вона належить усім характеристикам. Ця характеристика має відображати відсутність протиріч з іншими стандартами або характеристиками. Наприклад, відповідність надійності і практичності.
Кожна під характеристика якості далі розбивається на атрибути. Атрибут – це сутність, яка може бути перевірена або виміряна в програмному продукті. Атрибути не визначені в стандарті із-за їх різноманітності в різних програмних продуктах.

4.7. Модель характеристик експлуатаційної якості

В стандарті виділена модель характеристик експлуатаційної якості. Основні характеристики:

- Результативність — ступінь в якій користувачами досягаються задані цілі по точності і повноті розв’язування задач у встановленому контексті використання ПС
- Продуктивність — продуктивність при розв’язанні основних задач програмної системи, яка досягається при реально обмежених ресурсах в конкретному зовнішньому середовищі застосування.
- Безпека — надійність функціонування комплексу програм і можливий ризик від його застосування для людей, бізнесу і зовнішнього середовища.
- Задоволення потреб і витрат користувачів у відповідності з цілями застосування ПЗ.

Друга частина стандарту ISO 9126-2 присвячена формалізації зовнішніх метрик. Зовнішні метрики відносяться до атрибутів, які визначають поведінку ПЗ у зовнішньому середовищі і взаємодію з іншими програмами.

Третя частина стандарту ISO 9126-3 присвячена формалізації внутрішніх метрик. Такі метрики вимірюють атрибути внутрішніх характеристик ПЗ.

Метрика — це комбінація конкретного методу вимірювання (способу отримання значень) атрибуту сутності і шкали вимірювання (засобу, який використовується для структурування отриманих значень).

Викладені загальні рекомендації по використанню відповідних метрик і взаємозв’язку між типами метрик.

Четверта частина стандарту ISO 9126-4 призначена для покупців, постачальників, розробників, супроводжуючих, користувачів і менеджерів якості ПЗ. В ній повторена концепція трьох типів метрик, а також анотовані рекомендовані види вимірювань характеристик ПЗ.

5. Моделі життєвого циклу програмних систем

Життєвий цикл ПС визначається як «весь період існування системи від початку розробки до завершення її використання» (ДСТУ 2941-94. Розробка систем. Терміни і визначення)

ЖЦ поділяється на впорядковані стадії, основні з яких:

- Визначення потреб
- Аналіз вимог і оформлення концепції
- Розробка
- Виробництво
- Впровадження/продаж
- Експлуатація
- Супровід і підтримка
- Вилучення з експлуатації

Всередині кожної з цих стадій відбувається подальша деталізація по більш дрібним стадіям.

Моделі ЖЦ описують взаємозв'язки стадій.

Далі будемо розглядати стадії ЖЦ, які пов'язані з процесом розробки ПС, основні з яких:

- Аналіз вимог
- Проектування (попереднє і детальне)
- Реалізація
- Тестування

Найбільш відомі типи моделей ЖЦ: послідовні та ітераційні. Ці моделі на практиці можуть змішуватись, утворюючи змішані моделі ЖЦ.

Призначення моделей розробки

Моделі ЖЦ можуть використовуватись для:

- Організації, планування. Розподілення ресурсів (затрат праці і часу) і керування проектом розробки
- Організації взаємодії з замовниками і визначення складу документів (робочих продуктів), які розроблюються на кожній стадії
- Аналізу і оцінювання розподілу ресурсів і затрат на протязі ЖЦ
- Наглядного опису або в якості основи для проведення фінансових розрахунків з замовниками
- Проведення емпіричних досліджень з метою визначення впливу моделей на ефективність розробки і загальну якість програмного продукту

Рекомендації по можливому відображенню процесів ЖЦ на основні моделі розробки приведені в Керівництві по застосуванню стандарту ISO/IEC 12207 (Guide for ISO/IEC 12207 — Software life cycle processes).

5.1. Моделі послідовного виконання стадій

5.1.1. Каскадна модель

Каскадна (Waterfall) або стандартна модель — найбільш відома модель розробки, яка пропонується стандартами як базова. Ця модель характеризується стадіями, які виконуються послідовно. Кожна стадія має бути завершена до переходу до наступної. Створені в ній робочі продукти після їх верифікації та валідації мають бути

«заморожені» і передані на наступну стадію в якості еталону. Користувач бачить працюючий програмний продукт в самому кінці розробки. Найбільш жорстке обмеження цієї моделі – необхідність «заморозки» вимог. При цьому, для того, щоб мінімізувати ризик, допускаються лише невеликі зміни.

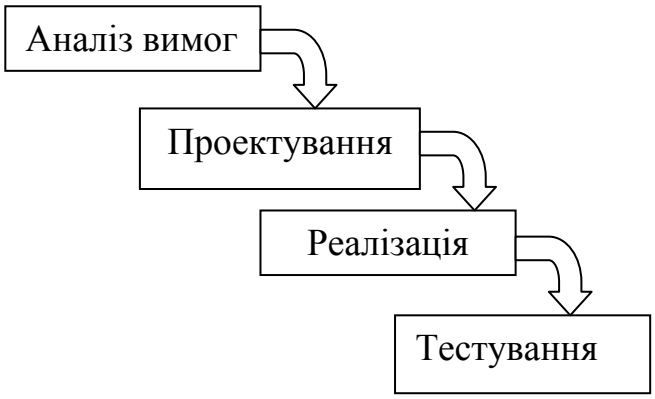

Рис 3. Каскадна модель

З точки зору якості ПС в цій моделі вартість виправлення дефектів на стадії тестування найбільша (у зрівнянні з іншими моделями), оскільки тестування відбувається в самому кінці розробки. Із-за нестачі часу на переробки і тестування існує значний ризик випуску ПС з серйозними дефектами.

5.1.2. Каскадна модель із зворотнім зв'язком

Ця модель розширює стандартну модель включенням в неї циклів зворотного зв'язку для повернення на попередню стадію при зміні вимог, проекту і по результатам інспекцій або дій по V&V

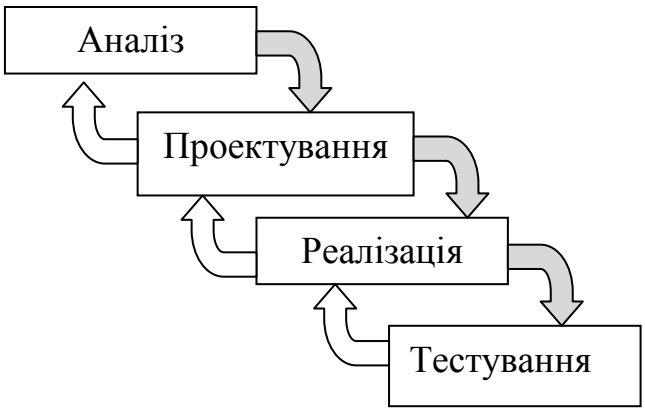

Рис 4. Каскадна із зворотнім зв'язком

Процеси V&V, які виконуються після завершення кожної стадії розробки, грають в цій моделі важливу роль.

Характеристики каскадної моделі:

- Послідовне впорядкування стадій
- Формальні перевірки по завершенні кожної стадій (інспекції, технічні огляди)

- Наявність документованих вимог і проекту

Переваги каскадної моделі:

- Застосування формальних перевірок дозволяє вчасно виявляти дефекти
- Чіткі критерії початку і завершення стадій
- Чіткі вимоги і цілі проекту

5.1.3. V-подібна модель

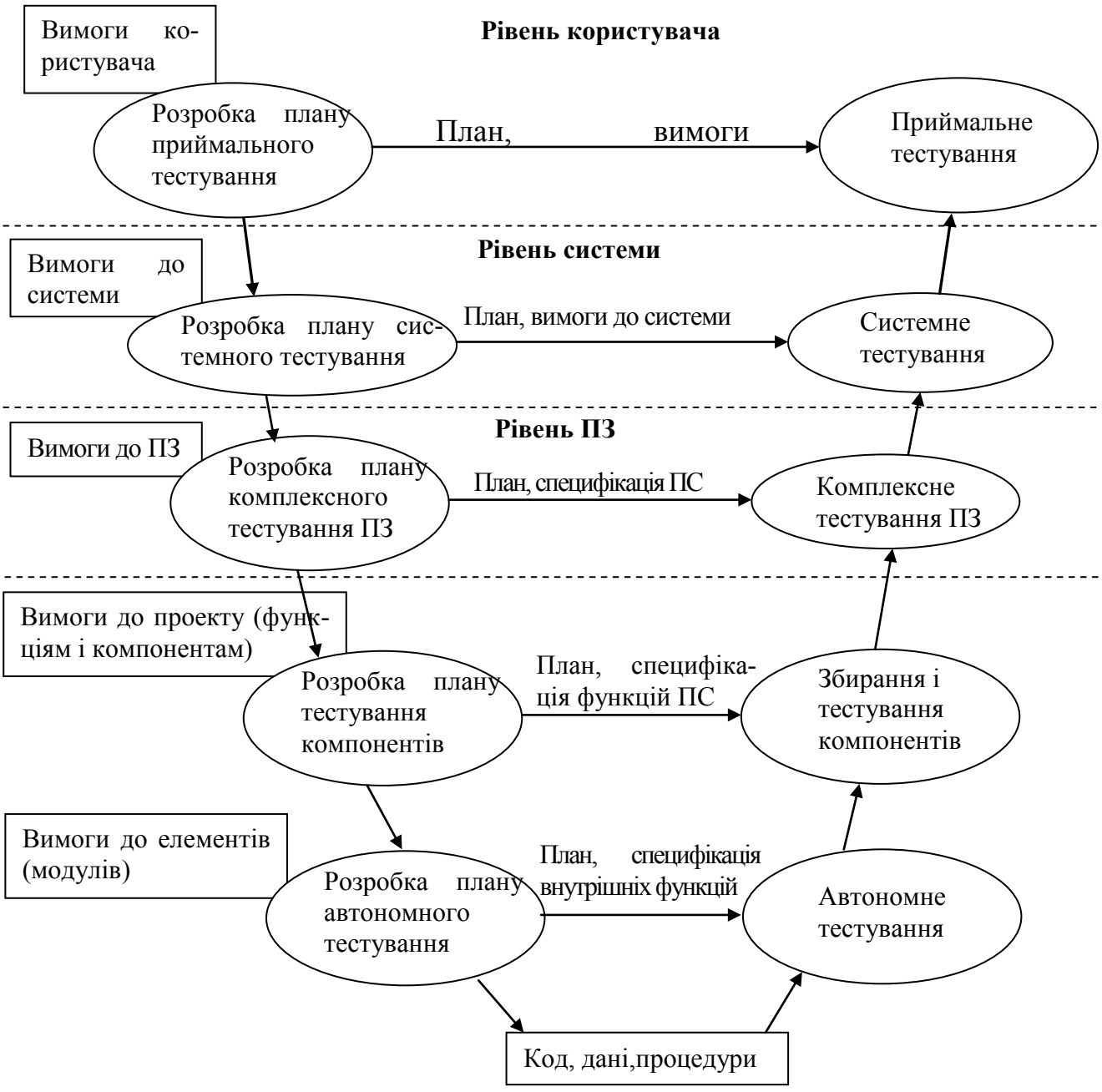

Рис. 5. V-подібна модель

V-подібна (V-shape) модель розширює каскадну модель і включає до неї дії по ранньому плануванню тестування.

В цій моделі тестування розглядається як неперервний процес, інтегрований в процес розробки ПС. Він включає два взаємопов'язаних під процесів – планування тестування в рамках процесів розробки системи (ліва гілка) і проведення тестування відповідних об'єктів (права гілка). На практиці задачі тестування ПЗ і системного тестування часто об'єднуються в один процес, однак для складних ПС тестування технічних характеристик має виконуватись окремо.

Характеристики V-подібної моделі:

- Перевірка і оцінка тестопридатності вимог на ранніх стадіях розробки (з допомогою аналізу, який виконується під час тестування).
- Наявність документованих тестових вимог.

Переваги V-подібної моделі:

- Забезпечує зворотний зв'язок з користувачем на ранніх стадіях ЖЦ.
- Покращує планування і розподіл затрат на тестування.
- Чіткі документовані цілі тестування.

5.1.4. Каскадна модель з прототипуванням

Модель є модифікацією V-подібної моделі з включенням в неї прототипів для моделювання вимог і проекту

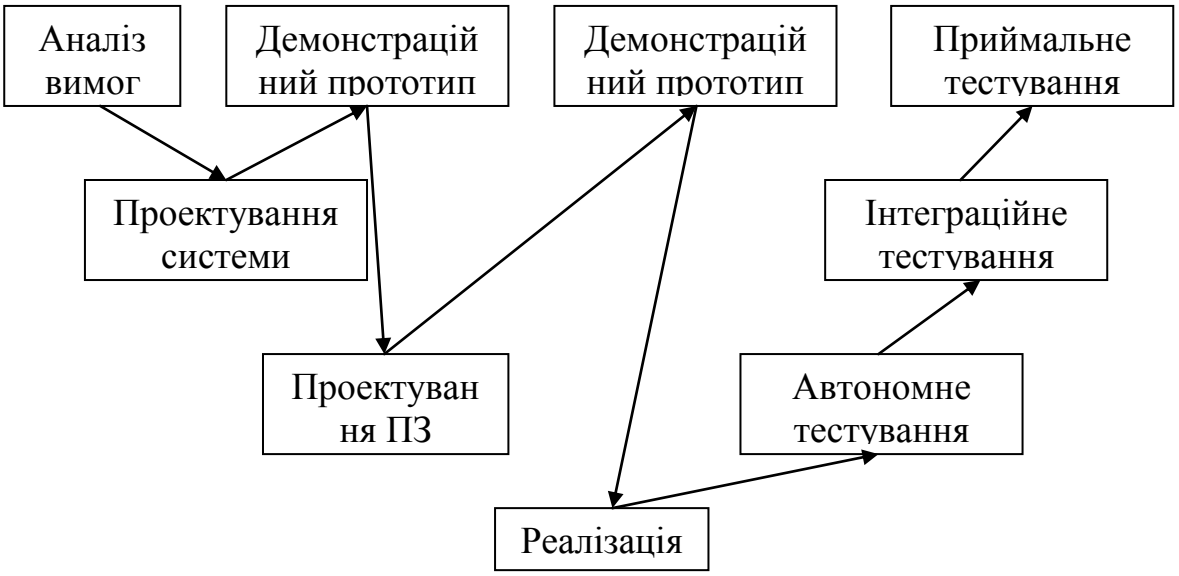

Рис. 6. Каскадна (пилоподібна) модель

Прототипи слугують для демонстрації і після розробки проекту їх викидають, а реалізація проекту може виконуватись в іншому середовищі.

Характеристика: для аналізу і моделювання проектних рішень застосовуються прототипи

Переваги: усуває проблеми, пов'язані з неповнотою і нечіткістю вимог

Ризики застосування послідовних моделей:

- Вимоги не повністю зрозумілі.
- Система занадто велика, щоб бути реалізованою одразу.
- Швидкі зміні в технологіях.
- Часта зміна вимог.

- Користувач не може використовувати проміжні результати.

Коли краще застосувати послідовну модель:

- Вимоги зрозумілі і не будуть суттєво мінятись.
- Система має невеликий розмір і складність.
- Усі можливості мають бути реалізовані одразу.
- Нова система розробляється на заміну старої і необхідно повністю замінити стару систему.

5.2. Ітераційні моделі

Ітераційні моделі загалом можна розділити на два класи: моделі з приростом (Incremental) і еволюційні (Evolutionary)

У відповідності з цими моделями програмний продукт розроблюється ітераціями, і кожна ітерація закінчується випуском працездатної версії програмного продукту. Основна відмінність між моделями — підхід до визначення вимог.

5.2.1. Ітераційні моделі з приростом

Програмний продукт розробляється ітераціями – з додаванням на кожній функціональних можливостей. При цьому спочатку визначаються усі вимоги до ПС, і можливо розроблюється попередній проект. Подальша розробка ПС розбивається на ітерації. В першій ітерації реалізується набір основних вимог, які забезпечують базову функціональність. Інші ітерації реалізуються в порядку критичності вимог для кінцевого користувача. При появі в середині ітерації нового набору вимог. вони відкладаються до реалізації наступної версії. В реальному житті це допущення може порушуватись і допускається перегляд вимог.

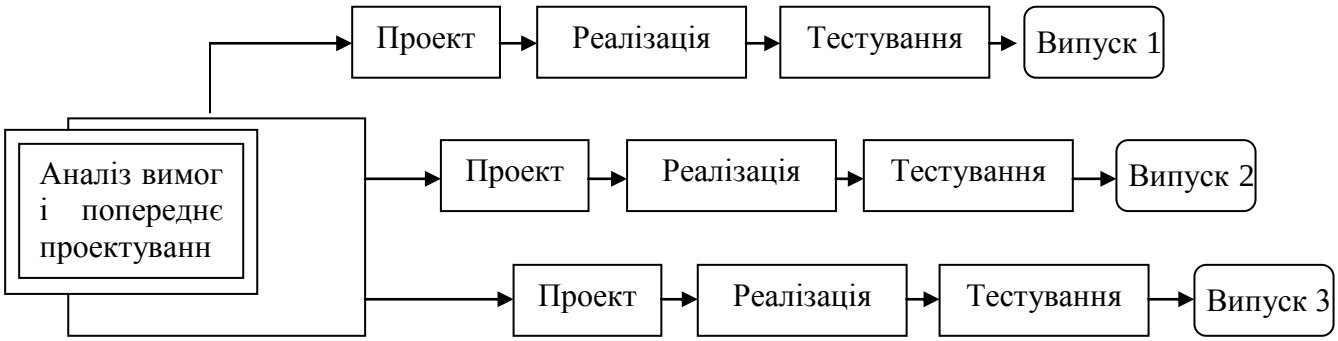

Рис 7. Ітераційна модель з перекриттям ітерацій

В різних моделях цієї групи ітерації можуть виконуватись послідовно або з перекриттям (нова ітерація починається до завершення попередньої).

Ітераційні моделі з приростом широко використовуються для розробки комерційних програмних продуктів, які розвиваються на протязі довгого періоду часу або для яких зовнішні вимоги змінюються слабо.

Характеристики ітераційних моделей з приростом:

- Аналіз і проектування виконуються для усієї системи.
- Базові функціональні вимоги реалізуються першими.
- Інші вимоги реалізуються в наступних версіях.

- Проміжні версії придатні для використання.

Переваги ітераційних моделей з приростом:

- Критичні функції реалізуються в першу чергу.
- Критичні функції тестуються більш ретельно.
- Найменш критичні задачі реалізуються останніми, що мінімізує наслідки відмови із-за дефектів.
- Завершення першої версії остаточно затверджує вимоги і проект.
- Раннє планування виконання тестування.
- Раннє виявлення дефектів користувачами.

Ризики, пов'язані з вибором моделі:

- Вимоги не повністю зрозумілі.
- Вимоги не стабільні.
- Усі можливості мають бути реалізовані одразу.
- Швидкі зміни в технології.

Коли краще застосовувати модель:

- Вимагається швидка реалізація основних можливостей.
- Якщо проект системи можна природним чином поділити на незалежні частини.

5.2.2. Еволюційні моделі

На відміну від моделей з приростом, еволюційні моделі застосовуються в тих випадках, коли усі вимоги не можуть бути визначені одразу або відомо, що вони можуть змінитись. Розробка проекту по цим моделям також виконується ітераціями. Але кожна ітерація охоплює усі стадії розробки, від аналізу вибраного набору вимог до випуску версії. На кожній ітерації виконується прототипування вимог і проекту.

До найбільш відомих еволюційних моделей відносяться спіральна модель і модель еволюційного прототипування.

1. Спіральна модель

Розроблена Боемом. Відображає керований ризиком процес еволюції проекту від аналізу до готовності продукту.

На кожному витку спіралі (стадії) виконуються наступні дії:

1. Визначаються цілі стадії. Розглядаються альтернативні рішення для досягнення цих цілей.
2. Проводиться оцінювання цих рішень. Ідентифікуються ризики завершення стадії і виконується їх аналіз. Приймаються рішення про продовження або завершення стадії.
3. Розробляються робочі продукти стадії та план для наступної стадії.
4. Останній виток спіралі може мати структуру каскадної моделі.

Види розглядуваних ризиків – ризики, які стосуються технічних аспектів розробки, фінансові ризики та ризики експлуатації.

Спіральна модель застосовується для складних проектів або в тих випадках, коли проблеми проекту недостатньо зрозумілі.

Характеристики спіральної моделі:

- Перший прототип моделює концепцію. Результатом є план вимог. Перед переходом до розробки наступного прототипу виконується аналіз ризику.
- Другий прототип моделює вимоги до ПЗ. Результатом є план розробки. Виконується аналіз ризику.

- Третій прототип моделює проект. В результаті створюється інтегрований і протестований прототип. Виконується аналіз ризику.
- Останній прототип (робочій) використовується як основа для детального проектування, кодування та тестування.

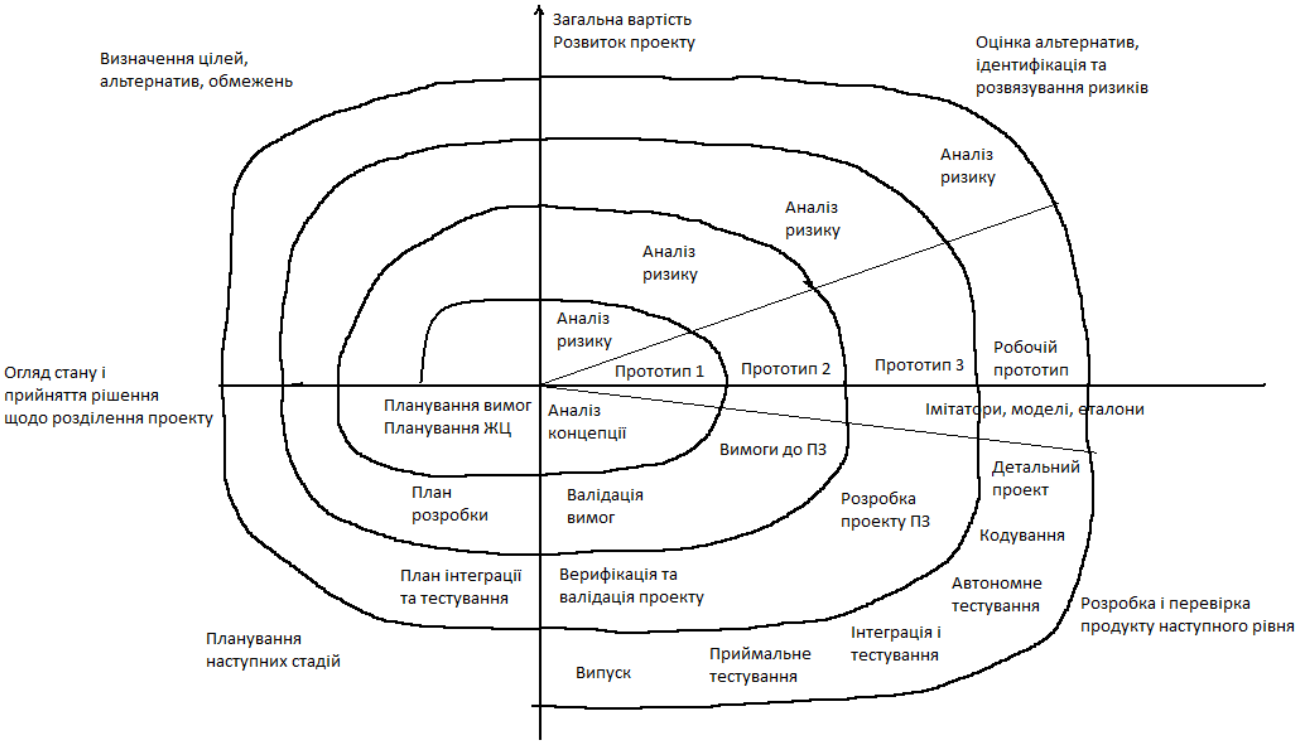

Рис 8. Спіральна модель

Ризики пов'язані з вибором моделі:

- Усі можливості мають бути реалізовані одразу.
- Проект неможна природним чином розділити на незалежні частини.

Коли краще застосовувати модель:

- Проект крупний, складний і вимоги не можуть бути визначені одразу.
- Нова технологія і вимагається її вивчення.
- Користувачі не можуть чітко сформулювати вимоги.
- Вимагається рання демонстрація можливостей.

Подальшим розвитком цієї моделі є Win-Win Spiral Model, яка основана на залученні до розробки різних категорій учасників проекту і визначенні умови успіху системи, які обговорюються на кожній ітерації.

2. Модель еволюційного прототипування.

Ця модель основана на застосуванні еволюційного прототипування в рамках усього ЖЦ розробки (а не тільки моделювання вимог). В літературі вона часто називається моделлю швидкої розробки програм (RAD — Rapid application development). Моделювання включає наступні етапи, наведені у таблиці 1.

Таблиця 1
Стадії моделі еволюційного прототипування

Крок	**Дія**
1. Аналіз застосовуваності моделі	Вивчення можливості застосування моделі для проекту
2. Обстеження замовника	Вивчення потреб користувача та розробка плану створення прототипу
3. Ітерація розробки функціонального прототипу	Створення і узгодження прототипу інтерфейсу користувача. Визначення не функціональних вимог і стратегії реалізації системи
4. Ітерація проектування і побудови	Побудова протестованої системи, яка задовольняє усім функціональним та не функціональним вимогам. На кроках 3 і 4 розробники визначаються прототипи, узгоджують строки розробки, побудову і перевірку прототипів. Ці кроки виконуються ітеративно і включають три ітерації: початкове ознайомлення, уточнення та узгодження
5. Реалізація	Встановлення системи в середовищі замовника, розробка документації і навчання

Ця модель застосовується для розробки не критичних бізнес-програм. Для яких найбільш важливими є функціональні можливості. Її застосування передбачає тісну взаємодію розробника і користувача.

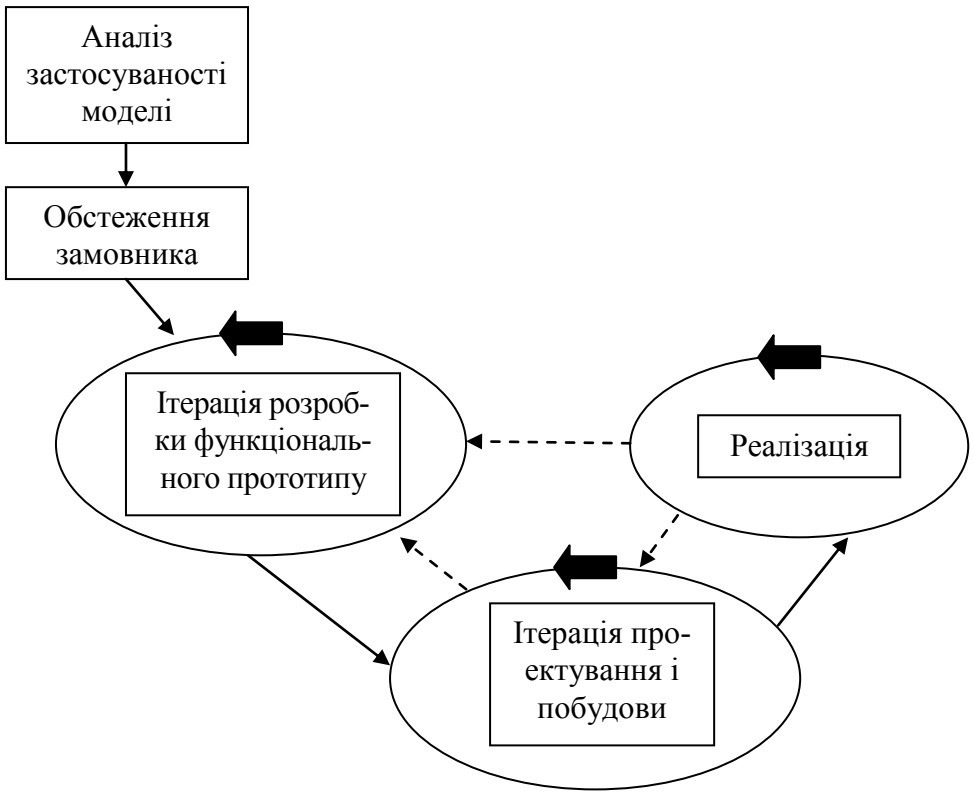

Рис 9. Модель еволюційного прототипування

Характеристики моделі:

- Гнучкість. Можливість швидко реагувати на зміни і розширення вимог.
- Пріоритети функціональних характеристик перед технічними (якості).

Переваги моделі:

- Раннє виявлення дефектів в інтерфейсі.
- Швидка демонстрація функціональних можливостей.

Ризики пов'язані з вибором моделі:

- Від розробника вимагається хороше володіння CASE-засобами та інструментами.
- Програми не моє бути критичною.
- Вимагається наявність потужних CASE-засобів.

Коли краще застосовувати модель

- Користувачі не можуть чітко сформулювати вимоги.
- Вимагається рання демонстрація можливостей.

5.3. Вибір моделі розробки

Вибір моделі суттєво залежить від двох факторів:

А) чи можна спочатку визначити практично повний набір функцій, які необхідно реалізувати в програмному продукті.

Б) чи мають усі жадані функції поставлятись замовнику одночасно.

Якщо А та Б виконуються, то вибираємо каскадні моделі.

А та не Б — вибирається ітераційна модель з прирощуваннями.

Не А та Б, а також бажана розробка прототипів для моделювання вимог – спіральна модель.

Не А та не Б — модель швидкої розробки програм, при умові, що строки розробки не будуть чітко встановлені.

6. Супровід програмного забезпечення

Результат зусиль з розробки програмного забезпечення полягає в передачі в експлуатацію програмного продукту, що задовольняє вимогам користувачів. Відповідно, в процесі експлуатації продукт буде змінюватися або еволюціонувати. Пов'язано це з виявленням при реальному використанні прихованих дефектів, змінами в операційному оточенні, необхідністю покриття нових вимог і т.п. *Фаза супроводу* в життєвому циклі, зазвичай, *починається відразу після приймання / передачі продукту і діє протягом періоду гарантії або, частіше, технічної підтримки*. Однак, сама діяльність, пов'язана з супроводом, починається набагато раніше.

Супровід програмного забезпечення (Software Maintenance) є складовою частиною життєвого циклу. На жаль, так склалося, що питанням супроводу приділяється істотно менше уваги, ніж іншим фазам життєвого циклу. В більшості організацій, розробка програмних систем явно у фаворі у порівнянні з діяльністю по супроводу. Однак, така ситуація поступово починає змінюватися (досить хоча б поглянути на частоту згадок ITIL[1] — IT Infrastructure Library, що приділяє особливу увагу питанням підтримки та супроводу інфраструктури інформаційних технологій). Як зазначає SWEBOK, це пов'язано з скороченням інвестицій організацій безпосередньо в розробку програмних систем, з метою збільшення термінів використання вже існуючого ПЗ. Звичайно, це не єдина причина. Постійна зміна бізнес-потреб, динаміка бізнесу та бажання підвищити віддачу від вже експлуатованих систем призводять до посилення ролі підтримки і супроводу ПЗ і природною інтеграції такої діяльності в бізнес-процеси підрозділів інформаційних технологій.

Проблема 2000 року вплинула на ставлення до супроводу на Заході. Розширення застосування продуктів Open Source у всьому світі і пов'язана з ним хвиля надій на отримання дешевих рішень в сфері IT призвела до того, що питання супроводу вийшли для багатьох організацій на перший план. Ситуація в багатьох IT-підрозділах показує, що такі надії виправдалися тільки частково. Використання продуктів Open Source Герасимчука необов'язково є дешевою альтернативою і, в ряді випадків, призвело навіть до великих проектних витрат саме в силу недостатньо проробленої політики експлуатації і супроводу побудованих на їх основі прикладних рішень. Це ні в якому разі не означає, що хвиля Open Source "захлинулася". Це означає лише, що ігнорування оцінки вартості супроводу призвело до перевищення бюджетів, нестачі ресурсів і, врешті-решт, частому провалу ініціатив по використанню таких продуктів в корпоративному середовищі.

Неготовність розглядати життєвий цикл як такий, що триває і після передачі системи в експлуатацію, неопрацьованість відповідних процедур коригування продукту після його випуску — основна біда і в бізнесі-середовищі, для якого програмне забезпечення лише інструмент, і в компаніях-інтеграторах, де часто "забувають" про необхідність розвинення успіху після впровадження системи у замовника, і в незалежних постачальників програмних продуктів, які, випускаючи нову версію кращого у своєму класі продукту, починають працювати над новою версією, приділяючи недостатню увагу підтримці та оновленню вже існуючих версій.

---

[1] ITIL, зокрема, визначає три аспекти управління життєвим циклом додатків – визначення вимог, проектування і розробку, і, нарешті, супровід. Все це, в контексті програмного забезпечення, відноситься до діяльності з управління додатками — Application Management в ITIL ICT Infrastructure Management (ICT — Information and Communications Technology).

Супровід програмного забезпечення в SWEBOK визначається як вся сукупність діяльності, необхідної для забезпечення ефективної (з точки зору витрат) підтримки програмних систем. Ці роботи виконуються як перед введенням системи в експлуатацію, так і після цього. Попередні роботи включають планування діяльності по супроводу системи, а також організацію переходу до її повнофункціонального використання. Якщо нова система повинна замінити стару систему, призначену для вирішення тих же завдань, просто на новому рівні ефективності, вартості використання, нових функціональних можливостей, в цьому випадку важливо забезпечити плавний перехід від старої системи на нову, максимально природний для користувачів. З цим пов’язано не тільки планування, наприклад, перенесення інформації, збереженої у відповідних базах даних, а й навчання користувачів, підготовка, настройка і перевірка "бойовий" конфігурації, визначення послідовності операцій, організація та навчання служби підтримки і т. п.

Галузь знань "*Супровід програмного забезпечення*" пов’язана з іншими аспектами *програмної інженерії*. По-суті, опис цієї галузі знань безпосередньо перетинається з усіма іншими дисциплінами. Структура області "Супровід програмного забезпечення" наведена на рис. 10.

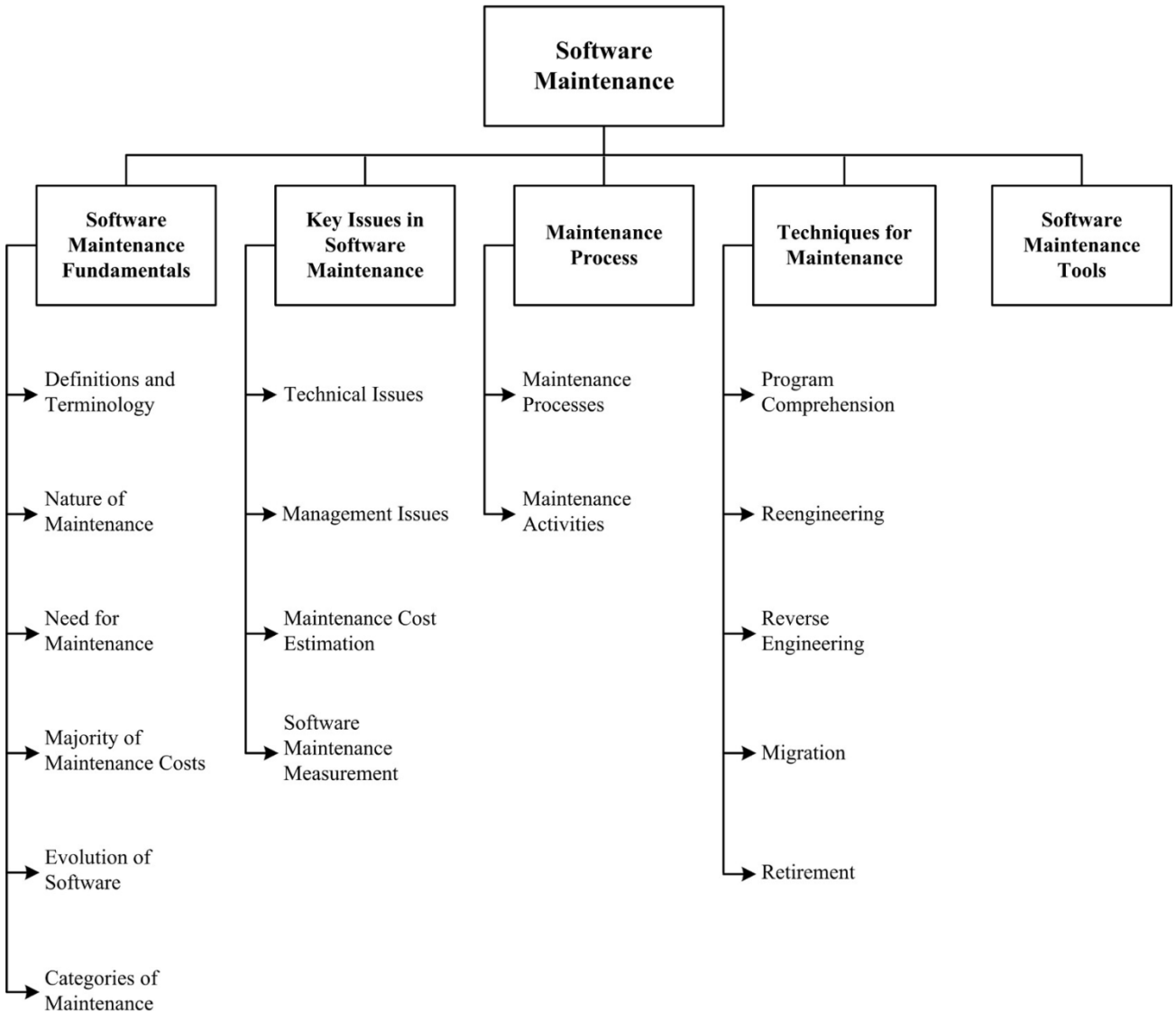

Рис. 10. Область знань супровід програмного забезпечення

6.1. Основи супроводу програмного забезпечення

Секція "Основи супроводу програмного забезпечення (Software Maintenance Fundamentals)" включає концепції і термінологію, що формують основи розуміння ролі та змісту робіт з супроводу програмних систем. Теми даної секції надають відповідні визначення та описують, чому саме існує потреба в супроводі. Категорії супроводу критично важливі для розуміння суті обговорюваних питань.

6.1.1. Визначення і термінологія

Супровід програмного забезпечення визначається стандартом IEEE Standard for Software Maintenance (IEEE 1219) як *модифікація програмного продукту після передачі в експлуатацію для усунення збоїв, поліпшення показників продуктивності та / або інших характеристик (атрибутів) продукту, або адаптація продукту для використання в модифікованому оточенні.* Даний стандарт також стосується питань підготовки до супроводу до передачі системи в експлуатацію, проте, структурно це зроблено на рівні відповідного інформаційного додатку, включеного в стандарт. У свою чергу, стандарт життєвого циклу 12207 (IEEE, ISO / IEC) позиціонує *супровід як один з головних процесів життєвого циклу*. Цей стандарт описує *супровід як процес модифікації програмного продукту в частині його коду і документації для вирішення виникаючих проблем <при експлуатації> або реалізації потреб у покращенні тих чи інших характеристик продукту. Завдання полягає в модифікації продукту при умови збереження його цілісності*. Міжнародний стандарт ISO / IEC 14764 (Standard for Software Engineering — Software Maintenance) визначає супровід програмного забезпечення в тих же термінах, що і стандарт 12207, надаючи особливого значення робіт з підготовки до діяльності по супроводу до передачі системи в реальну експлуатацію, наприклад, питань планування регламентів і операцій із супроводу.

6.1.2. Природа супроводу

Супровід підтримує функціонування програмного продукту протягом усього операційного життєвого циклу, тобто періоду його експлуатації. У процесі супроводу фіксуються і відслідковуються запити на модифікацію (запити на зміни), оцінюється вплив запропонованих змін, модифікується код та інші активи (артефакти) продукту, проводиться необхідне тестування і, нарешті, випускається оновлена версія продукту. Крім того, проводиться навчання користувачів і забезпечується їх щоденна підтримка при роботі з поточною версією продукту. У SWEBOK наголошується, що супровід, з точки зору операцій відстеження та контролю, має ширший зміст, ніж розробка (у загальному розумінні). Обсяг і активність операцій з контролю розробки у великій мірі залежить від сформованих практик, внутрішньокорпоративних регламентів і вимог, а також методологій та концепції управління (проектного менеджменту). Так чи інакше, відстеження і контроль — ключові елементи діяльності з супроводження програмного забезпечення (як і інших IT-активів підприємства).

Стандарт 12207 визначає поняття "maintainer" — у відповідному ДУСТ він йменується як "Персонал супроводу", маючи на увазі організацію, що виконує роботи з супроводу.

SWEBOK використовує даний термін, також, і щодо осіб, які проводять певні роботи по супроводу, на відміну, наприклад, від розробників, що займаються реалізацією системи в програмному коді.

Стандарт життєвого циклу 12207 також ідентифікує *основні роботи по супроводу*:

- реалізація процесу супроводу;
- аналіз проблем і модифікацій (змін);
- реалізацій модифікацій;
- огляд (оцінка) і прийняття рішень по супроводу;
- міграція (з однієї версії програмного продукту на іншу, з одного продукту на інший);
- виведення системи з експлуатації.

Фахівці з супроводу (персонал супроводу) можуть отримувати знання про програмне продукті безпосередньо від розробників. Взаємодія з розробниками і раннє залучення цих фахівців допомагає зменшити зусилля, необхідні для адекватного супроводу програмної системи. Передача знань персоналу супроводу, його навчання, повинно починатися не пізніше початку дослідної експлуатації продукту. В іншому випадку, зусилля на одночасну підтримку прикладної системи і навчання відповідних фахівців не тільки перевищить реально допустимі норми завантаження персоналу (як групи або служби супроводу і техпідтримки, так і розробників системи), але й знизить ефективність підтримки користувачів на критично важливому етапі первісного використання нової системи

Звичайно, залежно від складності системи, пік навантаження на службу супроводу припадає на перших 2-6 тижнів з моменту передачі системи в реальну експлуатацію. SWEBOK зазначає, що, в деяких випадках, інженери (що створювали систему) не можуть бути залучені до навчання та підтримки персоналу супроводу. Особливо часто це стосується тиражованих або "коробкових" систем. Це створює додаткові труднощі для фахівців, що забезпечують супровід. В той же час, інженери, що займаються технічною підтримкою (більш вузьке коло в команді супроводу, що включає менеджерів, адміністраторів та інших фахівців), повинні (в залежно від типу продукту) мати доступ до активів проекту (наприклад, опису його внутрішньої архітектури), включаючи код, документацію тощо. Саме таким чином починає формуватися інформаційна інфра-структура служби технічної підтримки та супроводу у виробників програмних продуктів.

Практика показує, що інженери з технічної підтримки виробника програмного забезпечення повинні не просто мати доступ до всіх ключовим активам проекту (код, документація, специфікації вимог, внутрішні моделі і тощо), але в їх обов’язки входить створення "патчів", виправлень помилок і, в особливих випадках, такі зміни, до випуску нової версії продукту, створюються із залученням безпосередньо розробників продукту (груп і підрозділів R & D — Research and Development). При цьому, розробники продукту інформуються про знайдені помилки і, в разі знаходження відповідних вирішень проблем фахівцями технічної підтримки, результати їх роботи передаються розробникам для того, щоб ті або включили такі зміни в нову версію програмного продукту, або знайшли більш адекватне рішення в контексті функціональності нової версії продукту.

В *обов’язки інженерів служби супроводу*, загалом випадку, входить:

- перевірка користувацького сценарію, що приводить до збою;
- ідентифікація причин збою, тобто локалізація помилки та причин її появи;
- надання відповідних виправлень або, при неможливості створення таких на даному етапі або в задані терміни — надання обхідного шляху вирішення проблеми для досягнення необхідних бізнес-завдань ("workaround");
- протоколювання всіх робіт і операцій;
- занесення опису проблеми й її вирішення в базу знань служби супроводу;
- передача всієї інформації розробникам;

- своєчасне інформування користувача про статус запиту та деякі інші роботи, зміст яких може варіюватися, залежно від регламентів і корпоративних стандартів.

6.1.3. Потреба в супроводі

Супровід необхідний для забезпечення того, щоб програмний продукт протягом усього періоду експлуатації задовольняв вимогам користувачів. Діяльність по супроводу застосовна для ПЗ, створеного з використанням будь-якої моделі життєвого циклу і методології розробки. Зміни програмної системи можуть бути обумовлені як діями щодо коригування її поведінки або незв'язані з необхідністю коригування (маючи на увазі вже не виправлення помилок, а, наприклад, підвищення продуктивності або розширення функціональності). У загальному випадку, *роботи з супроводу повинні проводитися для вирішення наступних завдань*:

- усунення збоїв;
- поліпшення дизайну:
- реалізація розширень функціональних можливостей;
- створення інтерфейсів взаємодії з іншими (зовнішніми) системами;
- адаптація для можливості роботи на іншій апаратній платформі (або оновленій платформі), застосування нових системних можливостей, функціонування в середовищі оновленої телекомунікаційної інфраструктури тощо;
- міграції успадкованого (legacy) програмного забезпечення
- виведення програмного забезпечення з експлуатації

Діяльність персоналу супроводу включає чотири ключових аспекти:

1) підтримка контролю (керованості) програмного забезпечення протягом усього циклу експлуатації:
2) підтримка модифікацій програмних систем;
3) вдосконалення існуючих функцій[2];
4) запобігання падіння продуктивності програмної системи до неприйнятного рівня.

Говорячи про запобігання деградації продуктивності, треба розуміти, що це може робитися і за рахунок поновлення потужності апаратної частини та відповідною телекомунікаційної інфраструктури, якщо це більш обґрунтовано, ніж модифікація самої програмної системи. Насправді це питання пов'язане з витратами і вартістю відповідних робіт, обладнання та підтримки оновленого системного оточення.

6.1.4. Пріоритет вартості супроводу

Роботи з супроводу споживають якщо не більшу (як зазначено в SWEBOK), то значну частина фінансових ресурсів життєвого циклу програмного забезпечення. Загальне розуміння супроводу має на увазі лише усунення збоїв. Однак, дослідження та опитування протягом багатьох років показують, *що більше 80% зусиль з супроводу пов'язані не стільки усуненням збоїв, скільки з іншими роботами, не пов'язаними з виправленням дефектів*. Багато менеджери з супроводу об'єднують у звітності питання розширення функціональності і виправлення помилок у підтримуваних програмних системах. Така суміш якісно різних робіт призводить до неправильного уявленню про реальну вартість супроводу в частині усунення дефектів. Розуміння різних категорій

[2] Судячи з усього, SWEBOK в даному випадку має на увазі функції не програмного забезпечення, а процесів супроводу і функції персоналу супроводу, як такі.

робіт в рамках діяльності по супроводу допомагає зрозуміти структуру реальних витрат. Крім того, розуміння факторів, що впливають на можливості супроводу системи, допомагають не тільки зберігати необхідний рівень витрат, але і знижувати їх. Існують як технічні, так і інші (наприклад, організаційні), фактори, що впливають на *вартість супроводу*:

- тип програми;
- новизна програмного забезпечення;
- наявність і кваліфікація персоналу по супроводу;
- тривалість використання програмної системи;
- характеристики і специфіка апаратної частини (а також телекомунікаційної інфраструктури);
- якість дизайну (наприклад, модульність або масштабованість), коду, документації та відповідних робіт з тестування системи.

6.1.5. Еволюція програмного забезпечення

У 1969 році Леман вперше пов'язав діяльність з супроводу і питання еволюції програмного забезпечення. Результати більш ніж 20-ти річних досліджень на чолі з Леманом привели до формулювання ряду важливих положень, ключове з яких стверджує, що діяльність з супроводу, по-суті, являє собою еволюційну розробку програмних систем. Ухваленню тих чи інших рішень у процесі супроводу, допомагає розуміння того, що відбувається з програмною системою в процесі її експлуатації. Існуюче (особливо, корпоративне) програмне забезпечення ніколи не буває повністю завершеним і продовжує еволюціонувати в протягом усього терміну експлуатації. У процесі еволюціонування, програмна система стає все більш складною до тих пір, поки не робляться спеціальні зусилля (у тому числі, в рамках спеціального проекту по модифікації) щодо зменшення його складності.

У той же самий час, якщо можна виділити тенденції розвитку програмної системи і її поведінку досить стабільно, його еволюціонування можна виміряти. Останніми роками робляться спроби розробити відповідні моделі оцінки зусиль з супроводу. У результаті вже створюються певні кошти (чисельні та інструментальні) управління роботами з супроводу, ряд з яких приводиться в посиланнях до даної секції SWEBOK.

6.1.6. Категорії супроводу

Багато джерел, зокрема, стандарт IEEE 1216, визначають *три категорії робіт по супроводу*:

- коректування;
- адаптація;
- вдосконалення.

Така класифікація була оновлена в стандарті ISO / IEC 14764 Standard for Software Engineering — Software Maintenance введенням четвертої складової. Таким чином, сьогодні говорять про *чотири категорії супроводу*:

- коригувальний супровід (corrective maintenance): "реактивна" модифікація програмного продукту вже після передачі в експлуатацію для усунення збоїв;
- адаптивний супровід (adaptive maintenance): модифікація програмного продукту на етапі експлуатації для забезпечення продовження його використання із заданою ефективністю (з точки зору задоволення потреб користувачів) в зміненому оточенні; в

першу чергу, мається на увазі зміна бізнес-оточення, що породжує нові вимоги до системи;

- супровід по вдосконаленню (perfective maintenance): модифікація програмного продукту на етапі експлуатації для підвищення характеристик продуктивності та зручності супроводу;
- профілактичний супровід (preventive maintenance): модифікація програмного продукту на етапі експлуатації для ідентифікації та запобігання прихованих дефектів до того, коли вони приведуть до реальних збоїв.

ISO / IEC 14764 (Standard for Software Engineering — Software Maintenance) класифікує адаптивний супровід і супровід по вдосконаленню як роботи з розширення функціональності продукту.

Цей стандарт також об'єднує коригувальну і профілактичну діяльність в загальну категорію *робіт з коригування системи*. Профілактичний супровід (новітня категорія робіт із супроводу) найбільш часто проводиться для програмних систем, пов'язаних з питаннями безпеки.

Таблиця 2. Категорії супроводу програмного забезпечення.

	Коригувальні роботи	**Роботи з розширення**
"Проактивний" підхід	Профілактичний супровід	Супровід по вдосконаленню
"Реактивний" підхід	Коригувальний супровід	Адаптаційний супровід

6.2. Ключові питання супроводу програмного забезпечення

Для забезпечення ефективного супроводу програмних систем необхідно вирішувати цілий комплекс питань і проблем, пов'язаних з відповідними роботами. Необхідно розуміти, що процес супроводу пред'являє унікальні технічні та управлінські вимоги до персоналу, що займається супроводом і, в першу чергу, до фахівців-інженерів.

Спроба знайти дефект в продукті, що містить 500 000 рядків коду, написаних іншими інженерами — яскравий приклад складнощів, з якими доводиться стикатися інженерам по супроводу. Інший приклад, вже організаційний, постійна боротьба за ресурси з розробниками (найчастіше виявляється в питаннях відволікання розробників від поточної роботи для допомоги у вирішенні проблем супроводу, а також в конкуренції за пріоритети фінансування розробки нової системи або супроводу існуючої). Одночасне планування перспективної версії системи, реалізація наступної версії і підготовка критичних патчів для поточної версії — ще один класичний приклад проблем, з якими доводиться стикатися в процесі експлуатації програмного забезпечення.

Дана секція висвітлює деякі технічні та управлінські питання, пов'язані з супроводом програмних систем. Ці питання і проблеми згруповані в набір тем:

- Технічні питання.
- Керування супроводом.
- Оцінка вартості.
- Вимірювання.

6.2.1. Технічні питання супроводу

Обмежене розуміння (Limited understanding)

Під *обмеженим розумінням* мають на увазі те, як швидко інженер з супроводу може зрозуміти, де необхідно внести виправлення або зміни в код системи, яку він не розробляв.

Дослідження показують, що від 40 до 60 відсотків зусиль по супроводу витрачається на аналіз і розуміння супроводжуваного програмного забезпечення. Формування цілісного погляду має велике значення для інженерів. Цей процес більш складний у випадку аналізу вихідного коду системи, особливо, коли процес еволюції системи від збірки до збірки, від релізу до релізу, в ньому ніяк не зазначений, не документований і коли розробники не можуть пояснити історію і структуру змін, що, на жаль, трапляється досить часто.

Практика показує, що для об'єктно-орієнтованих програм якісно спрощує задачу розуміння коду використання UML-інструментарію, здатного на основі коду відновити не тільки модель класів, але і їх взаємодії у формі діаграм класів (class diagram), комунікацій або співробітництва (collaboration в UML 1.x, перейменована в communication в UML 2.0) і, особливо, послідовностей (sequence diagram), що демонструє структуру взаємних викликів в часі. Якщо відповідний інструментарій надає одночасну візуалізацію коду і діаграми і забезпечує взаємну синхронізацію їх з погляду навігації (вибір методу в будь-який з представлених діаграм автоматично позиціонує відповідним чином редактор коду і, навпаки) — такі засоби автоматизації можуть скоротити час, необхідний для формування уявлення про систему, іноді — навіть не в рази, а на порядок (звичайно, при достатньому рівні знань інженера з супроводу). Якщо до цього додати документованість архітектури та ключових технологічних рішень з боку розробників системи — то завдання, звичайно, не стає тривіальним, однак, перетворюється на цілком розв'язну задачу. Взагалі кажучи, використання відповідних засобів автоматизації побудови моделей за кодом (задача зворотного інжинірингу — rerverse engineering) є обґрунтованою практикою вивчення будь-якої системи або фреймворку. При достатній кваліфікації інженера воно дозволяє вирішувати виникаючі питання коригування коду і розширення функціональності системи, не порушуючи загальні принципи її побудови, природним чином забезпечуючи її еволюцію, без шкоди її цілісності. При такому розумінні, навіть не заглядаючи в код системи або фреймворку, інженер здатний з дуже великою ймовірністю припустити можливі причини збою, а, в загальному випадку, і будь-яких аспектів поведінки системи. Тема зворотного інжинірингу висвітлюється SWEBOK як самостійна техніка супроводу (див. 6.3).

Тестування (Testing)

Ціна повторення повного набору тестів для основних модулів системи може бути істотною як за часом, так і за вартістю. Для супроводу системи особливо значущим є вибіркове регресійне тестування (див. область знань Software Testing, тему 2.2.6 Регресійне тестування) системи або її компонент для перевірки того, що внесені зміни не привели до неумисної зміни поведінки програмного забезпечення.

Питання полягає в тому, що часто складно знайти час для необхідного тестування. Не меншою проблемою є координація у проведенні тестів різними членами групи супроводу, які вирішують різні завдання. Якщо ж система виконує критичні функції, тимчасове виведення системи з експлуатації (як кажуть, переведення системи в offline) для виконання тестів часто виявляється просто неможливим.

Одним з ключових питань супроводу є організація робіт з тестування модифікацій експлуатованих систем, аж до попереднього планування і розробки регламентів, відповідно до яких, наприклад, ґрунтуючись на оцінці критичності запитів на зміни (як дефектів, так і важливих розширень функціональності або інтеграційних можливостей) персоналом супроводу будуть проводитися стандартні процедури. До таких процедур, поряд із протоколюванням запитів і проведених робіт, можуть належати: аналіз впливу змін, оцінка ризиків, тестування (різними методами, в різному обсязі), випуск попередніх версій патчів або оновлень в обмежене використання (якщо це дозволяє специфікація системи), використання "клону" системи (розгортання її на ідентичному обладнанні в ідентичній конфігурації) і т.п.

Аналіз впливу (Impact analysis)

Аналіз впливу описує як проводити (зокрема, з точки зору ефективності витрат) повний аналіз можливих наслідків і впливів змін, що вносяться в існуючу систему.

Персонал супроводу повинен володіти необхідними знаннями про специфіку системи (в ідеальному випадку, мати повне уявлення про систему на рівні її розробників) — її зміст і структуру. Інженери використовують ці знання для виконання робіт з аналізу впливу, ідентифікуючи всі системи[3] і програмні продукти, на які можуть вплинути зміни, що вносяться в програмну систему. При цьому, повинні бути визначені ризики, пов’язані з внесенням змін.

Запити на зміни[4] (change requests — CR), іноді згадуються як запити на модифікацію (modification request — MR), часто також називаються звітами про проблеми (problem report — PR), повинні аналізуватися і трансформуватися в терміни програмної системи. Ці кроки виконуються після того, як відповідний запит на зміну починає оброблятися в рамках процесу управління змінами або, як прийнято називати, конфігураційного управління, і фіксується в системі конфігураційного управління (див. область знань Configuration Management).

Цілі аналізу впливу можуть бути сформульовані наступним чином:

- Визначення змісту змін для визначення робіт з планування та реалізації.
- Отримання максимально можливої оцінки ресурсів, необхідних для проведення відповідних робіт.
- Аналіз вартості і вигоди від внесення запитаних змін (зазвичай стосується побажань, запитів на розширення системи)
- Обговорення складності питань, пов’язаних із внесенням відповідних змін.

Складність вирішення питання, поставленого відповідним запитом на зміни, часто є основним чинником визначення того, коли і як буде вирішена проблема. Інженери ідентифікую компоненти, в які необхідно внести зміни. Зазвичай розглядається декілька варіантів вирішення проблеми і виробляється (також, обов’язково, фіксуються в відповідній системі обробки запитів на зміни) найбільш оптимальний шлях її вирішення.

Оптимальність шляху не завжди означає оптимальне технологічне розв’язання. Іноді це може бути тимчасове рішення, може бути навіть порушує архітектурні шаблони системи, однак, обґрунтоване з точки зору термінів і вартості його реалізації. В той

---

[3] Як видно з опису даних робіт в SWEBOK, мова йде не тільки про компоненти системи, а й про її оточенні, включаючи інші системи, що функціонують в тому ж операційному / системному оточенні.

[4] Зазвичай запити на зміни поділяють на дві категорії — "побажання" (suggestions), які стосуються розширенню системи, і "звіти про помилки" (defect або bug report), що направляються користувачами в службу супроводу або інженерами з тестування — розробникам.

самий час, результати аналізу направляються розробникам системи, зазвичай працюючим над наступною версією, для включення відповідної зміни вже в рамках прийнятого стилю кодування, угод, архітектурних шаблонів і т.п. Безумовно, такий шлях багатьом може здатися просто неетичним, з точки зору "справжнього" інженерного підходу. Однак, якщо розробники готують наступну версію системи, зачіпаючи модуль, що модифікується службою супроводу, з погляду бізнес-рішень, "некрасивий", але швидкий шлях досягнення необхідного поводження системи, в більшості випадків, буде виглядати більш обґрунтованим, ніж прийняття на себе персоналом супроводу функцій розробників системи. Іноді, якщо необхідна зміна не настільки критична, щоб рішення було надано "вчора" (хоча користувачі, практично завжди, саме так характеризують свої запити в термінах пріоритету), логічним виглядає відкладання проведення відповідних модифікацій і передача цих робіт безпосередньо розробникам. Як це часто можна почути — "буде доступно в наступному релізі". Економічно, це часто буває більш ніж виправдано.

Якщо програмне забезпечення спочатку розроблялася з урахуванням подальшої підтримки, це може істотно полегшити аналіз впливів, як однієї з ключових робіт із супроводу.

Можливість супроводу (Maintainability)

Можливість супроводу програмної системи визначається, наприклад, глосарієм IEEE (стандарт 610.12-90 Standard Glossary for Software Engineering Terminology, оновлення 2002) як легкість супроводу, розширення, адаптації та коригування для задоволення заданих вимог. Стандарт ISO / IEC 9126-01 (Software Engineering — Product Quality — Part 1: Quality Model, 2001 р.) визначає можливість супроводу як одну з характеристик якості.

Для зменшення вартості подальшого супроводу, протягом усього процесу розробки необхідно специфікувати, оцінювати і контролювати характеристики, що впливають на можливість супроводу. Якщо такі роботи проводяться регулярно, це полегшує подальший супровід, підвищуючи його супруводжуваність (зокрема, як характеристику якості). Часто цього складно домогтися, тому що такого роду характеристики ігноруються при розробці. Розробники зайняті іншими запланованими роботами і також часто нехтують вимогами, що пред’являються до супроводу.

Однією з ключових проблем супроводу є відсутність системної документації, що заважає формуванню розуміння системи і, як наслідок, зумовлює неможливість адекватного аналізу впливу. Ця та інші проблеми можуть бути вирішені при використанні систематичного підходу до побудови зрілих процесів, застосуванню відповідних технік і автоматизації необхідних завдань з підтримки життєвого циклу.

6.2.2. Керування супроводом

Узгодження з організаційними цілями (Alignment with organizational objectivies)

Організаційні цілі описують як повернення інвестицій від діяльності з супроводу програмного забезпечення. Зазвичай, розробка ведеться на проектній основі, з певними часовими та бюджетними обмеженнями. Головний акцент робиться на випуску системи, що відповідає потребам користувачів, в задані терміни і в рамках бюджету.

На відміну від цього, супровід системи переслідує мету максимального продовження терміну експлуатації програмного забезпечення. Такий підхід може ґрунтуватися на необхідності оновлення та розширення програмного забезпечення, як відгуку на мінливі

потреби користувачів. При цьому, оцінка повернення інвестицій стає більш складною і призводить до формуванню у менеджменту погляду щодо того, що діяльність з супроводу споживає значну частину ресурсів без явно вираженої і кількісно обумовленою віддачі для організації.

Проблеми кадрового забезпечення[5] (Staffing)

Дана тема стосується питань залучення й утримання кваліфікованого персоналу по супроводу. Часто, робота по супроводу не виглядає привабливою, інженери по підтримці сприймаються як фахівці "другого класу" (у SWEBOK використовується стійке вираз "second-class citizens"), що призводить до безумовного падіння духу колективу, відповідального за підтримку систем.

Це серйозний виклик для менеджерів, відповідальних за питання супроводу і, взагалі кажучи, є класичною задачею загального менеджменту. Вирішення цього завдання, в першу чергу, знаходиться в руках старшого менеджменту, що формує відповідний стиль відносин між функціональними і допоміжними підрозділами. На більш високому рівні, для організацій і бізнесів — споживачів інформаційних технологій, це завдання пов’язана з внутрішньокорпоративними департаментами автоматизації, які занадто часто сприймаються тільки як центри витрат, а інформаційні технології не розглядаються як актив. В результаті, така позиція призводить до зниження ефективності роботи підрозділів автоматизації, а, отже, і падіння якості інформаційного забезпечення бізнесу, що позначається, в переважній більшості випадків, і на бізнесі, як такому.

Процес (Process)

Процес (у загальному випадку, життєвий цикл) є набором робіт, методів, практик і, свого роду, трансформацій, які використовуються людьми для розробки та супроводу програмних систем і асоційованих з ними продуктів. На рівні процесу, діяльність з супроводу програмного забезпечення має дуже багато спільного з розробкою, наприклад, в частині конфігураційного управління, що є критично важливою складової обох видів діяльності. У той же час, супровід включає роботи, що не представлені в процесі розробки (в 6.3.2 надано опис такого роду унікальних робіт).

Організаційні аспекти супроводу (Organizational aspects of maintenance)

У першу чергу, організаційні питання дають відповідь на питання яка організація буде нести відповідальність та / або які функції необхідно виконувати для забезпечення діяльності з супроводу.

Команда, що розробляла програмний продукт, далеко не завжди відповідає за його супровід.

При вирішенні питання, де (ким) будуть здійснюватися функції із супроводу, може бути прийнято рішення залишити їх безпосередньо тим, хто розробляв систему (як в термінах організації або компанії, так і маючи на увазі безпосередньо колектив розробників), або передати іншій команді або стороні. Часто, вибір супроводжує організації здійснюється виходячи з тих міркувань, які виглядають обґрунтованими для забезпечення адекватної підтримки системи та можливості її еволюціонування для задоволення мінливих потреб користувачів.

Відповідні рішення приймаються в кожному конкретному випадку, з урахуванням його специфіки. Але, що дійсно важливо відзначити, делегування або призначення

---

[5] Такий переклад, замість просто "кадрового забезпечення", більшою мірою відповідає прийнятому використанню терміна staffing.

повноважень і відповідальності по супроводу має бути проведено по відношенню тільки до однієї організації або особи (менеджеру відповідної команди підтримки).

Аутсорсинг (Outsourcing)

Запозичений термін "аутсорсинг" вже прижився не тільки в середовищі IT-менеджерів, він став частиною сучасного бізнесу та управлінських практик. Суть його полягає в передачі робіт, в першу чергу, допоміжних (непрофільних для організації) "на сторону". Великі корпорації передають в управління іншим організаціям цілі портфелі програмних систем, а, іноді, і цілком всю IT-інфраструктуру. У той же час, більш часто супровід передається іншим організаціям тільки для "другорядних" програмних систем (або, як мінімум, не критичних для виконання бізнес-функцій), так як власники таких систем не бажають втрачати контроль над асоційованими з цими системами даними або функціональністю. Відзначається, що часто роботи з супроводу передають "в аутсорсинг" тільки в тих випадках, якщо переконані в стратегічному контролі над супроводом.

Часто для вирішення питань супроводу (при збереженні "Стратегічного контролю"), компанії, для яких інформаційні технології не є профільними, але сприймаються як актив, формують спеціалізовані дочірні бізнес-структури, яким і передаються функції супроводу, а також і безпосередньо розробки програмних систем і, більше того, підтримки та розвитку всієї IT-інфраструктури. Це робиться з тим, щоб функціонуючи як самостійні бізнес-структури, вже колишні внутрішньокорпоративні підрозділи автоматизації, могли забезпечити більшу прозорість витрат, пов’язаних з інформаційним технологіями.

При цьому, підкреслює SWEBOK, контроль складно виміряти. У свою чергу, перед аутсорсером (організацією, приймаючої на себе відповідальність по супроводженню) стоїть серйозна проблема з визначення вмісту відповідних робіт для опису змісту відповідного контракту. Відзначається, що близько 50% сервісів, що надаються аутсорсером, проводяться без відповідного детальної угоди. Компанії, що займаються аутсорсингом, зазвичай затрачають кілька місяців на оцінку програмного забезпечення перш, ніж укладають відповідний контракт. Ще одне важливе питання полягає в необхідності визначення процесу та процедур передачі програмного забезпечення на зовнішній супровід.

6.2.3. Оцінка вартості супроводу

Як вже зазначалося, інженери повинні розуміти різницю в різних категоріях супроводу. Це, у великій мірі, необхідно для оцінки відповідних витрат. З точки зору планування, як складової частини проектної та управлінської діяльності, оцінка вартості є важливим аспектом діяльності з супроводу програмного забезпечення.

Оцінка вартості (Cost Estimation)

ISO / IEC 14764 (Standard for Software Engineering — Software Maintenance) визначає, що "існує два найбільш популярних методу оцінки вартості супроводу — параметрична модель і використання досвіду". Найчастіше, обидва цих підходу комбінуються для підвищення точності оцінки.

Параметричні моделі (Parametric models)

SWEBOK наводить ряд джерел, детально розглядають питання оцінки вартості супроводу і, зокрема, параметричні моделі. Для використання таких моделей використовуються дані попередніх проектів. Поряд із історичними даними використовується метод функціональних точок (див. стандарт IEEE 14143.1-00).

Досвід (Experience)

Серед тих підходів, які дозволяють підвищити точність оцінок, отриманих при використанні параметричних моделей — застосування досвіду (у формі експертної думки, наприклад, при використанні техніки оцінки "Delphi", назва якої походить від "делфійского оракула"), аналогій, а також структури декомпозиції робіт. Найкращі результати отримуються при поєднанні емпіричних методів з наявним досвідом. Отримані дані використовуються як результат програми вимірювання аспектів супроводу.

6.2.4. Вимірювання у супроводі програмного забезпечення

Форми і дані вимірювань в процесі супроводу можуть об'єднуватися в єдину корпоративну програму кількісних оцінок, які проводяться по відношенню до ПЗ. Багато організацій використовують популярний і практичний підхід для вимірювань, який базується на оцінці кількості проблем та статусу їх рішень (issue-driven measurement). Ідеї цього підходу систематизовані в проекті Practical Software and Systems Measurement (PSM).

Існують загальні (для всього життєвого циклу) метрики і, відповідно, їх категорії, які визначаються Інститутом Програмної Інженерії університету Карнегі-Меллон: розмір, зусилля, розклад і якість. Застосування цих метрик є гарною відправною точкою для оцінки робіт з боку організації, що відповідає за супровід.

Більш детальне обговорення питань вимірювань відносно продуктів і процесів проводиться в галузі знань "Процес програмної інженерії (Software Engineering Process). Питання організації програми вимірювань відносяться до галузі знань "Управління програмної інженерії"(Software Engineering Management).

Спеціалізовані метрики (Specific Measures)

Існують різні методи внутрішньої оцінки продуктивності (benchmarking) персоналу супроводу для порівняння роботи різних груп супроводу. Організація, що веде супровід, повинна визначити метрики, за якими будуть оцінюватися відповідні роботи. Стандарти IEEE 1219 (Standard for Software Maintenance) і ISO / IEC 9126-01 (Software Engineering — Product Quality — Part 1: Quality Model, 2001 г.) пропонують спеціалізовані метрики, орієнтовані саме на питання супроводу і відповідні програми.

Нижче наведені типові метрики оцінки робіт по супроводу, відповідних поширеної класифікації експлуатаційних характеристик програмного забезпечення:

- Аналізованість (Analyzability): оцінка (в першу чергу, додаткових) зусиль або ресурсів, необхідних для діагностики недоліків або причин збоїв, а також для ідентифікації тих фрагментів програмної системи, які повинні бути модифіковані.
- Змінність (Changeability): оцінка зусиль, необхідних для проведення заданих модифікацій.
- Стабільність (Stability): оцінка випадків непередбаченого поведінки системи, включаючи ситуації, виявлені в процесі тестування.
- Тестованість (Testability): оцінка зусиль персоналу супроводу і користувачів по тестуванню модифікованого програмного забезпечення.

6.3. Процес супроводу

Питання організації процесу супроводу безпосередньо пов'язані з відповідними стандартами і практиками реалізації такого процесу. Тема "Роботи з супроводу" (Maintenance Activities) розрізняє питання супроводу і розробки і показує взаємозв'язок с іншими аспектами діяльності програмної інженерії.

Типові потреби в процесах програмної інженерії докладно описані і документовані в різних джерелах. Одна з найбільш детально опрацьованих і поширених моделей процесів, спочатку створених з орієнтацією на програмне забезпечення — CMMI (Capability Maturity Model Integration — інтегрована модель зрілості), розроблена в Інституті програмної інженерії університету Карнегі-Меллон. CMMI приділяє спеціальну увагу процесам супроводу.

6.3.1. Процеси супроводу

Процеси супроводу описують необхідні роботи і детальні входи / виходи цих робіт.

Ці процеси розглядаються в стандартах IEEE 1219 (Standard for Software Maintenance) і ISO / IEC 14764 (Standard for Software Engineering — Software Maintenance).

Процес супроводу починається за стандартом IEEE 1219 з моменту передачі програмної системи в експлуатацію (post-delivery stage) і стосується таких питань, як планування діяльності з супроводу

Стандарт ISO / IEC 14764 уточнює положення, пов’язані з процесом супроводу, стандарту життєвого циклу 12207. Роботи з супроводу, описані в цьому стандарті аналогічні роботам в IEEE 1219, за винятком того, що згруповані трохи інакше (рис. 11).

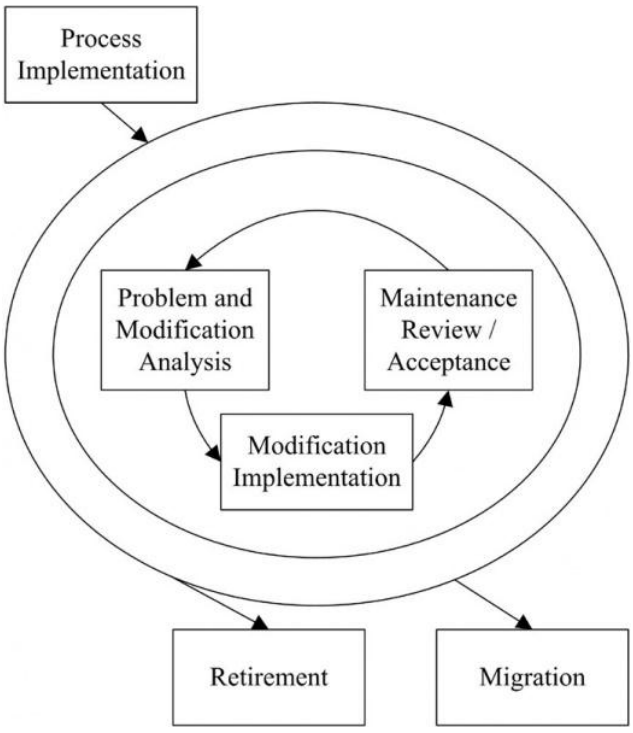

Рис. 11. Процес супроводу згідно стандарту ISO / IEC 14764.

Роботи з супроводу в стандарті 14764 розбиті на завдання:

- Process Implementation — реалізація процесу.
- Problem and Modification Analysis — аналіз проблем і необхідних модифікацій.
- Modification Implementation — проведення модифікацій (реалізація змін).
- Maintenance Review / Acceptance — оцінка і прийняття проведених робіт при супроводі.
- Migration — міграція (на модифіковану або нову версію ПЗ).
- Software Retirement — виведення з експлуатації (припинення експлуатації ПЗ).

У наданих SWEBOK джерелах можна знайти опис історії еволюції відповідних моделей стандартів ISO / IEC і IEEE. Крім того, існує і загальна (узагальнена) модель процесів супроводу. Agile-методології, які активно розвиваються в останні роки, пропонують "полегшені" процеси, в тому числі, й для організації діяльності з супроводу, наприклад, Extreme maintenance.

6.3.2. Роботи з супроводу

Як вже зазначалося, багато робіт з супроводу схожі на аспекти діяльності по розробці. В обох випадках необхідно проводити аналіз, проектування, кодування, тестування і документування. Фахівці з супроводу повинні відслідковувати вимоги так само, як і інженери-розробники і оновлювати документацію у міру розробки та / або випуску оновлених або нових релізів продукту. Стандарт ISO / IEC 14764 рекомендує, щоб персонал або організації, що відповідають за супровід, у разі використання елементів процесів розробки у своїй діяльності, адаптували ці процеси в відповідності зі своїми потребами.

Унікальні роботи (Unique activities)

Існує ряд процесів, робіт і практик, унікальних для діяльності з супроводу. SWEBOK наводить такі приклади такого роду унікальних характеристик:

- Передача (Transition): контрольована і координована діяльність з передачі ПЗ від розробників групі, службі або організації, що відповідає за подальшу підтримку;
- Ухвалення / відхилення запитів на модифікацію: запити на зміни можуть як прийматися і передаватися в роботу, так і відхилятися по різним обґрунтованих причин — обсягом і / або складності необхідних змін, а також необхідних для цього зусиль. Відповідні рішення можуть також прийматися на основі пріоритетності, оцінці обґрунтованості, відсутності ресурсів (у тому числі, відсутності можливості залучення розробників до вирішення завдань з модифікації, при реальному наявності такої потреби), затвердженої запланованістю до реалізації в наступному релізі і т. п.
- Засоби сповіщення персоналу супроводу та відстеження статусу запитів на модифікацію і звітів про помилки (Modification Request and Problem Report Help Desk): функція підтримки кінцевих користувачів, яка ініціює роботи з оцінки, аналізу пріоритетності та вартості модифікацій, пов’язаних з надійшли запитом або повідомленої проблемою.
- Аналіз впливу (Impact Analysis): аналіз можливих наслідків змін, внесених у існуючу систему.
- Підтримка ПЗ (Software Support): роботи з консультування користувачів, що проводяться у відповідь на їх інформаційні запити (request for information), наприклад, що стосуються відповідних бізнес-правил, перевірки, змісту даних і специфічних (ad hoc) питань користувачів і їх повідомлень про проблеми (помилках, збоях, непередбачуваній поведінці, нерозумінню аспектів роботи з системою і т.п.).
- Контракти і зобов’язання: до них відносять класичну угоду про рівень наданого сервісу — Service Level Agreement (SLA), а також інші договірні аспекти, на підставі яких, група / служба / організація з супроводу виконує відповідні роботи.

Додаткові роботи, що підтримують процес супроводу (Support activities)

Довгий переклад назви даної теми пов’язаний із змістом відповідних робіт, описаних SWEBOK, як роботи персоналу супроводу, що не включають явної взаємодії з користувачами, але необхідні для здійснення відповідної діяльності. До таких робіт належать: планування супроводу, конфігураційне управління, перевірка і атестація (V & V —

verification and validation), оцінювання якості ПЗ, різні аспекти огляду, аналізу та оцінки (reviews), аудит (audit) і навчання (training) користувачів.

Також, до таких спеціальних робіт відноситься навчання персоналу супроводу.

У зв'язку із особливою значущістю цих робіт, їм присвячені наступні підтеми даної секції галузі знань із супроводу ПЗ.

Роботи з планування супроводу (Maintenance planning activity)

Планування є більш ніж необхідним для адекватного проведення робіт по супроводу та має стосуватися пов'язаних з цим питань з кількох точок зору:

- Бізнес-планування (організаційний рівень).
- Планування безпосередніх робіт по супроводу (рівень передачі програмного забезпечення).
- Планування релізів / версій (рівень програмного забезпечення).
- Планування обробки конкретних запитів на зміну (рівень запиту) На рівні індивідуального запиту, роботи з планування проводяться разом з проведенням аналізу впливу.

Планування версій вимагає від персоналу супроводу виконання завдань:

- Отримання та збору інформації про дати розміщення індивідуальних запитів і звітів.
- Досягнення угоди з користувачами про зміст (функціональності, поведінці і т.п.) наступних релізів / версій програмного забезпечення.
- Ідентифікації потенційних конфліктів і можливих альтернатив.
- Оцінки ризиків для <функціонування> поточного релізу і розробки плану "відкату" на немодифікований (поточний, до внесення змін, що стосуються розглянутого запиту) варіант системи, у разі виникнення проблем, пов'язаних з модифікацією.
- Інформування всіх зацікавлених осіб. Незважаючи на те, що розробка програмних системи, зазвичай, займає від кілька місяців (що більш типово) до декількох років, супровід (як діяльність з підтримки використання) і активна експлуатація систем займає кілька років, а то і більше.

Проведення оцінки ресурсів — невід'ємна частина планування. Ресурси, необхідні для супроводу повинні бути оцінені і закладені в бюджет ще при розробці системи. Планування робіт з супроводу повинно починатися одночасно з прийняттям рішення про створення системи і узгоджуватися з цілями забезпечення якості (відзначається в IEEE 1061-98 Standard for a Software Quality Metrics Methodology).

Спочатку необхідно визначити концепцію супроводу. Такий документ, наприклад, по стандартом ISO / IEC 14764 (Standard for Software Engineering — Software Maintenance) повинен стосуватися наступних питань:

- Змісту діяльності з супроводу.
- Адаптації процесу супроводу.
- Ідентифікації організації, яка займатиметься супроводом.
- Оцінки вартості супроводу.

Після розробки концепції діяльності з супроводу повинен бути сформований відповідний план супроводу. Цей план повинен готуватися одночасно з розробкою програмної системи. План повинен визначати як користувачі будуть розміщувати свої запити на модифікацію (зміни) або повідомляти про помилки, збої та проблемах. Питанням планування приділяють спеціальну увагу вже згадувані стандарти IEEE 1219 і ISO / IEC 14764.

Стандарт ISO / IEC 14764 надає спеціальні рекомендації з організації плану супроводу.

Нарешті, на рівні бізнес-питань, структура, що відповідає за супровід, повинна проводити загальну діяльність з бізнес-планування, що стосується бюджетування, фінансового менеджменту та управління людськими ресурсами, так само, як і будь-яке інший (у тому числі, профільне, якщо мова йде про споживачів IT) підрозділ компанії.

Конфігураційне управління (Software configuration management)

Стандарт IEEE 1219, присвячений організації супроводу програмного забезпечення, визначає конфігураційне управління як критично важливий елемент процесу супроводу. Процедури конфігураційного управління повинні забезпечувати перевірку, атестацію та аудит на всіх кроках, необхідних для ідентифікації, авторизації, реалізації та випуску програмного продукту.

Більше того, недостатньо просто відстежувати запити на зміни і повідомлення про проблеми (modification requests, problem reports). Повинні бути контрольовані і сам програмний продукт, і будь-які зміни (не тільки в коді, але документації, специфікаціях тощо, тобто будь-яких активах продукту та проекту). Такий контроль встановлюється реалізацією і строгим проходженням затвердженим процесом конфігураційного управління (software configuration management, SCM). Область знань "Конфігураційне управління" докладно описує і обговорює процеси, відповідно до яких, розміщуються, оцінюються і затверджуються запити на зміни. У ряді окремих аспектів і характеристик, конфігураційне управління при супроводі й розробці дещо відрізняється, що повинно контролюватися вже на операційному рівні. Реалізація SCM-процесу забезпечується розробкою і проходженням планом конфігураційного управління.

Організація, підрозділ або група супроводу (в особі представників) бере участь у роботі часто формованого органу Configuration Control Board, відповідального за розгляд і прийняття в роботу запитів на зміни. Основною метою такої участі є, на думку SWEBOK, визначення вмісту наступних релізів / версій.

Якість програмного забезпечення (Software quality)

Не варто сподіватися, що в процесі і результаті супроводу, якість програмного забезпечення буде підвищуватися. Для підтримки процесу супроводу повинні плануватися і реалізовуватися відповідні процедури і процеси, спрямовані на підвищення якості. Роботи і техніки по забезпеченню якості (SQA), перевірці та атестації (V & V), огляду, аналізу та оцінці (review), а також аудиту, повинні проходити в контексті взаємодії та узгодження з усіма іншими процесами, спрямованими на досягнення бажаного рівня якості. SWEBOK, ґрунтуючись на стандарті ISO / IEC 14764 (Standard for Software Engineering — Software Maintenance), рекомендує адаптувати відповідні процеси, техніки і активи, що відносяться до розробки ПЗ. До них, наприклад, відносяться документація з тестування та результати тестів. Додаткові деталі можна знайти у відповідній галузі знань "Якість програмного забезпечення" (Software Quality).

6.4. Техніки супроводу

Секція "Techniques for Maintenance" вводить деякі загальноприйняті техніки, використовувані в процесі супроводу програмних систем.

6.4.1. Розуміння програмних систем

Для реалізації змін програмісти витрачають значну частину часу на читання і формування розуміння програмного продукту. Засоби роботи з кодом є ключовим інструментом для вирішення цього завдання. Чітка, однозначна і лаконічна документація забезпечує адекватне розуміння програмних систем.

6.4.2. Реінжиніринг[6]

Реінжиніринг визначається як *детальна оцінка* (examination) і *перебудова програмного забезпечення для формування розуміння, відтворення (на рівні моделі) та подальшої реалізації його функцій в новій формі* (наприклад, з використанням нових технологій і платформ, при збереженні існуючої і розширенням і полегшенням можливостей додавань нової функціональності). Відзначається, що в індустрії існують різні позиції щодо реінжинірингу — одні вважають, що реінжиніринг є найбільш радикальною і витратною формою змін програмних систем, інші, що такий підхід може застосовуватися і для не настільки кардинальних змін (наприклад, як зміна платформи чи архітектури). Реінжиніринг, зазвичай, проводиться не так для поліпшення можливостей супроводу (maintainability), скільки для заміни застарілого програмного забезпечення. В принципі, реінжиніринг можна розглядати як самостійний проект, що включає в себе, як зазначає SWEBOK, формування концепції, застосування відповідних інструментів і технік, аналіз і застосування досвіду проведення реінжинірингу, а також оцінку ризиків і переваг, пов'язаних з такими роботами.

Слід зазначити, що реалізація продукту в новій якості (формі) при збереженні основної функціональності оригінального продукту, є невід'ємною і визначальною частиною реінжинірингу, на відміну від зворотного інжинірингу, розглянутого нижче, і є важливою складовою частиною реінжинірингу.

6.4.3. Зворотний інжиніринг

Зворотний інжиніринг (часто плутають з реінжинірингом, в тому числі, в розумінні SWEBOK) — *це процес аналізу ПЗ з метою ідентифікації програмних компонент і зв'язків між ними, а також формування уявлення про ПЗ*, з подальшою перебудовою в новій формі (вже, в процесі реінжинірингу).

Зворотний інжиніринг є пасивним, припускаючи відсутність діяльності по зміні або створення нового програмного забезпечення. Зазвичай, в результаті зусиль по зворотному інжинірингу створюються моделі викликів (call graphs) і потоків управління (control flow graphs) на основі початкового коду системи. Один з типів зворотного інжинірингу — створення нової документації на існуючу систему (redocumentation). Інший з поширених типів — відновлення дизайну системи (design recovery).

До питань зворотного інжинірингу, як і до питань реінжинірингу, також належать роботи з рефакторингу. *Рефакторинг — трансформація програмного забезпечення, в процесі якої програмна система реорганізується (без переписування коду) з метою поліпшення структури, без зміни поведінки*. Збереження "форми" (платформи, архітек-

[6] Для обох тем — 6.4.2 та 6.4.3 можливе застосування слова реконструкція, залежно від контексту, що означає як повну перебудову або відтворення чого-небудь, ідентичного за певним характеристикам оригінальному зразку, так і відновлення будь-якої сутності за збереженими та / або доступним зовнішніми ознаками, де відновлення може мати на увазі знову-таки створення нового зразка або подання про оригінальну сутності.

турних і технологічних рішень) існуючої програмної системи дозволяє розглядати рефакторинг як один з варіантів зворотного інжинірингу.

Можливе об’єднання тем даної секції, включаючи реінжиніринг і зворотний інжиніринг, в загальну тему "Reverse and Reengineering" даної галузі знань "Супровід", з подальшою деталізацій у вигляді "підтем" Такий підхід відповідає структурі SWEBOK. При цьому, відповідна структура може бути організована, наприклад, наступним чином:

- Формування уявлення про експлуатовану / супроводжувану систему: включає відновлення, в першу чергу, бізнес- і функціональних вимог до системі:
- Відновлення детального дизайну системи: включає ідентифікацію всіх компонентів системи та створення моделі викликів та інших зв’язків між компонентами;
- Рефакторинг: як можливий процес структурних змін, що вносяться до системи, зокрема для поліпшення можливостей щодо її подальшого супроводу (включаючи модифікацію, пов’язану з розширенням функціональності);
- Переробка системи: створення нового релізу / версії системи в тій же формі (наприклад, з використанням тієї ж технологічної платформи), що і поточна (експлуатована) версія;
- Створення нової системи: розглядає поточну версію і систему в цілому, як застарілу — legacy.

Рекомендована література

1. Андон Ф.И., Коваль Г.И., Коротун Т.М. Основы инженерии качества программных систем. 2-е издание. — К.: Академпериодика, 2007. — 672 с.
2. Соммервилл Я. Инженерия программного обеспечения. — М.: Вильямс, 2002. — 624 с.
3. Канер С., Фолк Дж., Нгуен Х.К.. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. – К.: Диасофт, 2001. — 544 с.
4. ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту та оформленню.
5. Дастин Э., Рэшка Дж., Пол Дж. Автоматизированное тестирование программного обеспечения. – М.: "ЛОРИ", 2003. — 568 с.
6. ДСТУ 2844-94. Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення.
7. ISO/IEC 9126-1: 2001. Software engineeering. Product quality. Part 1: Quality Model.
8. Брауде Э. Дж. Технология разработки программного обеспечения. СПб.: Питер, 2004 — 655 с.
9. Орлик С. Введение в программную инженерию и управление жизненным циклом программного обеспечения. Программная инженерия. Сопровождение программного обеспечения. http://sorlik.blogspot.com
10. Марченко Е. Что такое качество программного обеспечения. http://software-testing.ru/library/testing/general-testing