МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

ГНАТОВСЬКА Г.А.

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«ТЕХНОЛОГІЯ СТВОРЕННЯ ПРОГРАМНИХ ПРОДУКТІВ»

для студентів 3 курсу денної форми навчання

Напрям:підготовки 6.05010101 “Комп’ютерні науки”

Одеса, 2015

ОГЛАВЛЕНИЕ

ВСТУП 5
ПОЧАТКОВІ ВІДОМОСТІ 7
1 БАЗОВІ ПОНЯТТЯ, ВИДИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 8
1.1. Види програмного забезпечення 9
1.2. Програмне забезпечення як виріб 9
1.3. Технологія розробки програмного забезпечення 11
1.4. Проблеми розробки складних програмних систем 13
1.4.1. Блочно-ієрархічний підхід до створення складних систем 14
Ключові терміни 18
2 ЖИТТЄВИЙ ЦИКЛ І ЕТАПИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 19
2.1. Процеси життєвого циклу програмного забезпечення 19
2.2 Процес розробки програмного забезпечення 22
2.3 Еволюція моделей життєвого циклу програмного забезпечення 26
2.4 Каскадна модель (waterflow model) 26
2.5 Ітеративна модель (Iterative and incremental development) – модель з проміжним контролем 30
2.6 Спіральна модель 31
Питання для самоконтролю 35
3 СТАНДАРТИЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 36
3.1. Міжнародні стандарти ISO 37
3.2 Стандарти організації IEEE – Institute of Electrical and Electronics Engineers 39
3.3 Стандарт зрілості компанії-розробника програмного забезпечення CMM - Capability Maturity Model 40
3.4 Стандарт SPICE 45
Питання для самоконтролю 46
4 СУЧАСНІ МЕТОДОЛОГІЇ РОЗРОБЛЕННЯ ПРОГРАМНИХ СИСТЕМ 47
4.1 CASE-засоби та нотації моделювання програмних систем 47
4.2 Жорсткі та гнучкі стратегії в методологіях програмування 50
4.3 Методологія Rational Unified Process (RUP) 53
4.4 Методологія Microsoft Solution Framework (MSF) 55
4.5 Методологія eXtreme Programming (XP) 61
4.6 Гнучке розроблення ПЗ на основі Agile 64
4.7 Архітектура ПЗ. Стандарти опису архітектури 67
4.8. Шаблони проектування. Патерни 68
Питання для самоконтролю 71

5 ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 72
5.1 Забезпечення якості ПЗ 72
5.2. Тестування ПЗ 74
5.3 Методи оцінки якості тестування 76
5.4 Види тестів 78
5.5 Верифікація програм 81
Питання для самоконтролю 82
6 МАРКЕТИНГ ПРОГРАМНИХ ПРОДУКТІВ 84
6.1 Основні ринкові вимоги доПЗ 84
6.2 Оцінка якості ПЗ з позиції маркетингу 85
СПИСОК ВИКОРИСТАНОЇ ТА РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ 97

ВСТУП

Дисципліна “Технологія створення програмних продуктів” входить до складу нормативної частини навчального плану підготовки бакалаврів з галузі знань “Інформатика та обчислювальна техніка”, напряму підготовки 6.050101 “Комп’ютерні науки ”. Викладається відповідно до освітньо-професійної програми, освітньо-кваліфікаційної характеристики та навчального плану підготовки бакалаврів.

Створення програмної системи – вельми трудомістке завдання , і фахівець повинен мати уявлення про методи аналізу, проектування, реалізації та тестування програмних систем, орієнтуватися в існуючих підходах і технологіях.

Проектування програмних продуктів, як і будь-яких інших складних систем, виконується поетапно з використанням блочно-ієрархічного підходу, який передбачає розробку продукту по частинах з наступною збіркою . На кожному етапі виконуються певні проектні операції, які відповідним чином документуються. Крім того програмний продукт повинен супроводжуватися різного роду програмою документацією (керівництво програміста, користувача, оператора). «Технологія створення програмних продуктів» - це дисципліна, яка розглядає додаток теорії , знань і практики для ефективної побудови програмних систем, що задовольняють вимогам користувача і замовника. У рамках дисципліни вивчаються всі процеси, що ведуть до створення програмних продуктів (ПП): від розробки вимог до ПП через проектування , розробку та атестацію до модернізації програмних систем.

Мета дисципліни: забезпечити отримання студентами теоретичних знань і практичних навичок щодо сучасних технологій створення програмних продуктів.

Предмет дисципліни: моделі життєвого циклу ПЗ та основні методології та засоби розробки ПЗ. Технологія створення та документування програмних продуктів.

Структурно-логічне місце дисципліни: попереднє вивчення дисциплін “Алгоритмізація та програмування”, “Об'єктно-орієнтоване програмування”, “Технології комп'ютерного проектування”, “Організація баз даних та знань”, “Управління ІТ-проектами”, “ Моделювання систем”.

Завдання дисципліни: В результаті вивчення дисципліни студенти повинні:

знати:

– основні моделі ЖЦ програмних засобів,
– сучасні методології розробки, умови їх застосування,
– правила документування текстів програм та іменування змінних і об’єктів,
–основні моделі та методи проектування архітектури ПЗ, патерни та шаблони проектування,
– засоби автоматизації розробки програмних продуктів;

вміти:

– вибирати стратегії для планування життєвого циклу системи;

– визначати організаційну, економічну, технічну та операційну здійсненність проекту;

– реалізовувати та тестувати компоненти програмного забезпечення;

– аналізувати вимоги замовника до програмних продуктів;

професійні компетенції:

– знання стандартів, методів і засобів управління процесами життєвого циклу інформаційних систем, продуктів і сервісів інформаційних технологій; володіння технологією розроблення програмного забезпечення відповідно до вимог і обмежень замовника;

– знання сучасних технологій та інструментальних засобів розробки програмних систем, уміння їх застосовувати на всіх етапах життєвого циклу.

ПОЧАТКОВІ ВІДОМОСТІ

Протягом кількох останніх років інформатика - дослідження автоматичної обробки інформації формується в самостійну науку. Ще не всі поняття, що застосовуються в інформатиці, сформульовані точно, і тим не менш, особливих розбіжностей в їх застосуванні немає.

В рамках дисципліни «Технологія створення програмних продуктів» деякі поняття ми будемо використовувати в вузькому сенсі, що описується нижче.

Програма – це тексти будь-яких програм на мові програмування або в об'єктному коді, придатні для виконання на електронно-обчислювальних машинах.

Комплекс програм (КП) – це сукупність взаємопов'язаних програм для електронно-обчислювальних машин, в основному як об'єкт розробки кінцевого програмного продукту на різних етапах його створення, проте ще не досяг завершеного стану, придатного для тиражування та експлуатації з певними якісними показниками.

Програмне забезпечення (ПЗ) – , або програмні засоби (ПЗа), або програмний продукт (ПП) - це сукупність програм і пов'язаних з ними даних певного призначення, що придатні для виконання на електронно-обчислювальних машинах, та які пройшли випробування з зафіксованими показниками якості і забезпечені комплектом документації, достатньої для кваліфікованої експлуатації за призначенням та використання як продукції виробничо-технічного призначення.

У процесі розробки програмного забезпечення переважно будемо використовувати термін «Комплекс програм» (або просто «Програма», якщо він не викличе непорозумінь). І тільки після успішного його завершення, випробування та впровадження - термін «Програмне забезпечення» («Програмні засоби», «Програмний виріб», «Програмний продукт»).

Слід зауважити, що поняття «Програма» використовується програмістами в самому широкому діапазоні значень: від примітивної, простої програми (можливо, ще не до кінця написаної) до готового програмного продукту. І якщо це не буде ускладнювати розуміння сенсу, ми для простоти викладу також будемо вживати його в різних ситуаціях по-різному в традиційному для цього поняття сенсі. Але в основному будемо дотримуватися наведених вище формулювань.

З появою нових технологій в розробку програмного забезпечення (візуальні і об'єктно-орієнтовані), з'явилися і нові поняття: «Проект» і «Додаток» («Додаток для користувача») як об'єкт розробки. Розробка додатка стала його «проектуванням».

1 БАЗОВІ ПОНЯТТЯ, ВИДИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Технологія програмування – дисципліна, що вивчає технологічні процеси програмування та порядок їх проходження.

Системна інженерія (System engineering) – розділ науки, що вивчає питання розробки комп'ютерних систем (архітектура, дизайн, інтеграція ПЗ та ін.).

Програмна інженерія (Software engineering) – дисципліна, спрямована на розробку і супровід програмного забезпечення систем, що функціонують надійно і ефективно, можуть удосконалюватися і еволюціонувати і відповідають вимогам, визначених замовником.

Програмування (Programming) – процес підготовки завдань для їх вирішення за допомогою комп'ютера; ітераційний процес складання програм.

Програма – дані, призначені для управління конкретними компонентами системи обробки інформації з метою реалізації певного алгоритму, послідовність машинних команд, призначених для досягнення конкретного результату.

Програмне забезпечення (ПЗ / Software) – комп'ютерні програми, процедури, а також документація і дані, що стосуються функціонування комп'ютерної системи.

Вперше термін *software* ввів відомий статистик Джон Тьюк (John Tukey) в 1958 році для позначення різниці апаратного забезпечення ЕОМ (*hardware*) від засобів обробки даних. Бьярне Страуструп (Bjarne Stroustrup) відзначив, що добре ПЗ неможливо побачити, але можна відчути, коли воно працює з помилками.

За видами виконуваних функцій програмне забезпечення підрозділяється на *системне, прикладне та інструментальне.* Такий поділ є умовним, оскільки широке впровадження комп'ютеризації призвело до того, що майже кожна програма має ознаки декількох видів ПЗ.

Інтегроване середовище розробки програмного забезпечення (Integrated development environment, IDE) - це система програмних засобів, що використовується програмістами для розробки програмного забезпечення.Як правило, середовище розробки включає текстовий редактор, компілятор і/або інтерпретатор, засоби автоматизації збирання, налагодження та різноманітні інструменти для конструювання графічного інтерфейсу користувача. Значного поширення об'єктно-орієнтованого програмування (ООП) призвело до того, що сучасні інструменти розробки включають браузер класів та інспектор об'єктів.

На сьогодні в середовища розробки підключають систему керування версіями, засоби тестування та ін.

1.1 Види програмного забезпечення

Системне ПЗ (System software) призначене для управління роботою комп'ютера, розподілу його ресурсів, підтримки діалогу з користувачами, а також для часткової автоматизації розроблення нових програм. Як правило, системні програми забезпечують взаємодію інших програм з апаратними складовими, організацію інтерфейсу користувача. Виділяють три типи системного ПЗ:

- *операційна система (ОС)* – програмне забезпечення, що забезпечує інфраструктуру, на якій можуть працювати прикладні програми. Найпоширеніші ОС – Microsoft Windows, Mac OS X та Linux;
- *системи програмування* – призначені для полегшення та часткової автоматизації процесу розроблення та відлагодження програм;
- *сервісні програми (утиліти)* – розширюють можливості ОС. До утиліт відносять архіватори, антивіруси, драйвери та ін.

Прикладне ПЗ (application, application software) – комп'ютерна програма, що вирішує конкретні задачі фахової діяльності користувача.

Інструментальне ПЗ призначене для розроблення всіх видів інформаційно-програмного забезпечення. При цьому під інформаційним забезпеченням розуміють сукупність 6

Інструментальне ПЗ призначене для розробки всіх видів інформаційно-програмного забезпечення. При цьому під інформаційним забезпеченням розуміють сукупність попередньо підготовлених даних, що необхідні для роботи програмного забезпечення. До інструментального ПЗ відносять текстові редактори, системи управління базами даних, транслятори мов програмування.

1.2 Програмне забезпечення як виріб

Сучасні програми вирішують найрізноманітніші завдання за змістом і галузевим значенням.

У багатьох випадках програми створюються в єдиному екземплярі для вирішення приватних дослідницьких завдань, для обробки експериментального матеріалу, прискорення обчислень, моделювання процесів, і т.д. Такі програми не мають масового застосування та доступні для використання тільки тим, хто їх розробив. Вони стають об'єктами науково-технічної творчості і рідко стають промисловими виробами.

Зовсім іншим класом програм є *індустріальні програмні засоби,* які можна кваліфікувати як продукцію виробничо-технічного призначення.

Вони являють собою програми на носіях даних з технічною (експлуатаційною та технологічною) документацією, розроблені відповідно до діючих стандартів і пройшли державні, міжвідомчі або відомчі випробування.

Програмні засоби, прийняті у виробництво, виготовляються за затвердженою в установленому порядку технологією. Вони повинні відповідати затвердженим технічним умовам і діючій нормативно-технічній документації, забезпечуватися гарантіями постачальника.

У цьому випадку мова йде про програмні вироби (про програмний продукт для ЕОМ).

Під ***програмним виробом*** (ПВ) розуміється універсальне програмне забезпечення, яке призначається для широкого кола користувачів, можливо, навіть не відомих заздалегідь, і повинно рекламуватися, підтримуватися в працездатному стані, розширюватися протягом тривалого періоду часу.

Програмний виріб – ц власне програми плюс документація, гарантія якості, рекламні матеріали, навчання, поширення та супровід. Окрема машинна програма або сукупність програм і програмний виріб - далеко не одне і те ж.

Програмний виріб є продукт ретельного планування і цілеспрямованої розробки, що супроводжується чіткою документацією, який пройшов всі необхідні випробування; який описано у відповідних технічних публікаціях, розмножено у необхідній кількості примірників, обслуговується і контролюється постачальником за заздалегідь продуманим планом і може розглядатися як товар.

З питань розробки систем програмного забезпечення (але не програмних виробів) можна зробити такі припущення:

- розробник створює програмне забезпечення для себе або, принаймні, організаційно пов'язаний з користувачами програмного забезпеченні, що розробляється;
- користувач формулює свої вимоги безпосередньо розробнику, якщо останній сам не є одночасно користувачем;
- користувач бере активну участь у розробці або в обслуговуванні програмного забезпечення;
- програмне забезпечення повинно працювати тільки на певній конфігурації комплексу технічних і програмних засобів в обмеженому діапазоні змін його складу і структури даних;
- розробник сам вводить в дію програмне забезпечення у користувача;
- проблеми, що виникли при використанні програмного забезпечення, вирішуються користувачем спільно з розробником або з персоналом, що здійснює його технічне обслуговування (супровід);
- програми не мають масового застосування та доступні для використання тільки тим, хто їх розробив;
- використання програми припиняється після отримання результату.

При розробці програмного виробу (за винятком особливого випадку розробки програмного забезпечення за контрактом для єдиного користувача) можна зробити такі припущення:

- розробник не знайомий з користувачем;
- вимоги користувача формуються або розробником, або передаються йому посередницькою організацією (наприклад, поставляє програмне забезпечення);
- користувачі не беруть участь у розгляді та узгодженні проектних рішень, якщо не вважати рідкісних випадків, коли їх інтереси представлені посередниками;
- програмне забезпечення повинно зберігати працездатність в широкому діапазоні конфігурацій обчислювальних комплексів і при самих різних системних програмних засобах;
- користувачі вводять програмне забезпечення в дію або самі, або з сторонньою допомогою, але ця допомога виходить не від розробника;
- проблеми, що виникли при використанні програмного забезпечення, вирішуються шляхом листування, а іноді через посередника;
- програмний виріб призначений для широкого кола користувачів, можливо, невідомих;
- програмне забезпечення використовується багаторазово і тривалий час.

1.3 Технологія розробки програмного забезпечення

Технології (з грецької: ремесло + наука) – сукупність знань про способи і засоби проведення виробничих процесів.

В одному крайньому випадку одна людина здійснює поетапну розробку програми зі свого термінала в невимушеній обстановці. Природно, він створює порівняно невеликі програми, що не вимагають особливої оцінки.

В іншому звичайному випадку розробляється дуже складне програмне забезпечення, що призначене для функціонування в реальному масштабі часу і вимагає трудовитрат обсягами в тисячі людино-годин.

Ці дві взаємно протилежні ситуації характеризуються різним ступенем формалізації та проведенні процесу розробки програмних засобів.

Ступінь формалізації та проведення процесу розробки програмного забезпечення безпосередньо залежить від цілей його створення, його величини, чисельності групи розробників та інших факторів. Від того, наскільки правильно і вдало з погляду технології розробки програмного забезпечення побудовано додаток, залежить якість і життєздатність кінцевого продукту.

Під ***технологією розробки програмного забезпечення*** (ТРПЗ) розуміється сукупність узагальнених і систематизованих знань, або наука про оптимальні способи (прийоми) проведення процесу розробки програмного

забезпечення, що забезпечує в заданих умовах отримання програмної продукції із заданими властивостями.

Технологія розробки програмного забезпечення являє собою інженерний підхід до розробки програмних засобів ЕОМ, що охоплює методологію програмування, проблеми забезпечення надійності програм, оцінки робочих характеристик і якості проектів.

Технологія розробки програмного забезпечення розглядає питання управління проектуванням систем програмного забезпечення, а також засоби і стандарти розробки програм.

Технологія розробки програмного забезпечення визначає деяку професійну культуру роботи фахівців (не тільки програмістів), що забезпечує заданий рівень продуктивності праці і якості одержуваної в результаті програмної продукції.

Технологія розробки програмного забезпечення охоплює процес розробки програмного забезпечення від появи потреби в ньому до його виготовлення, передачі користувачеві, модифікації в процесі експлуатації і припинення його використання внаслідок морального старіння.

В ідеалі ***технологія розробки програмного забезпечення*** *повинна задовольняти основним нижче перерахованим вимогам.*

1. Необхідна стандартизація мов проектування програм, оформлення та випробування програмних модулів, а також гарантії їх якості. Це дозволить значно скоротити дублюючі розробки, впровадити складальне програмування і вести накопичення на підприємствах і в країні високоякісного програмного продукту для його багаторазового використання як типових комплектуючих виробів.
2. Вести постійний контроль і забезпечення якості програм.
3. Програми не повинні містити неперевірених шляхів і ситуацій функціонування, які призводять до несподіваних результатів.
4. Користувачеві або покупцю програм необхідно дати чітке уявлення про можливості даної програми і технологічних умовах експлуатації, при яких гарантуються певні функції і якості.
5. Технологія розробки програмного забезпечення повинна забезпечувати відторгнення програмного виробу від його розробника, тобто людський фактор у програмуванні має бути зведений до мінімуму.
6. Технологія розробки програмного забезпечення та засоби її підтримки (автоматизації) повинні забезпечувати цілеспрямовану роботу, насамперед колективу програмістів, а не окремих особистостей. Вона повинна спонукати колектив працювати тільки правильно і злагоджено; повинна автоматично блокувати будь-які не санкціоновані технологією дії.

7. Необхідно вести акуратне документування всіх етапів розробки. Документація повинна також заноситися і зберігатися на магнітних носіях. Доступ до цієї інформації має бути відкритим, простим і автоматизованим.
8. Робота користувача повинна забезпечуватися розвиненою інформаційно-довідковою системою.
9. Засоби автоматизації технології повинні охоплювати всі етапи роботи колективу програмістів.
10. Технологія розробки програмного забезпечення повинна бути простою в освоєнні, із засобами підказки, що автоматично включаються.
11. Технологія розробки програмного забезпечення повинна мати засоби автоматичної фіксації в хронологічному порядку всіх дій, які виконуються в процесі колективного виготовлення програмного виробу - повинні вестися і зберігатися в системі журнали (протоколи, щоденники) розробки. Ці засоби повинні дозволяти відновлювати будь-який стан процесу розробки на будь-якому інтервалі виготовлення програмного продукту.

1.4. Проблеми розробки складних програмних систем

Більшість сучасних програмних систем об'єктивно дуже складні. Ця складність обумовлюється багатьма причинами, головною з яких є *логічна складність розв'язуваних ними завдань*.

Поки обчислювальних установок було мало і їхні можливості були обмежені, ЕОМ застосовували в дуже вузьких галузях науки і техніки, причому, в першу чергу, там, де завдання, що вирішувались, були добре детерміновані і вимагали значних обчислень. У наш час, коли створено потужні комп'ютерні мережі, з'явилася можливість перекласти на них рішення складних ресурсномістких завдань, про комп'ютеризацію яких раніше ніхто і не думав. Зараз до процесу комп'ютеризації залучаються зовсім нові предметні області, а для вже освоєних областей ускладнюються вже сформовані постановки завдань.

Додатковими факторами, що збільшують складність розробки програмних систем, є:

- складність формального визначення вимог до програмних систем;
- відсутність задовільних засобів опису поведінки дискретних систем із великою кількістю станів при недетермінованій послідовності вхідних впливів;
- колективна розробка;
- необхідність збільшення ступеня повторюваності кодів.

Складність визначення вимог до програмних систем.

Складність визначення вимог до програмних систем обумовлюється двома факторами. По-перше, при визначенні вимог необхідно врахувати велику кількість різних факторів. По-друге, розробники програмних систем не є фахівцями в автоматизуються предметних областях, що автоматизуються, а фахівці в предметній області, як правило, не можуть сформулювати проблему в потрібному ракурсі.

Відсутність задовільних засобів формального опису поведінки дискретних систем.

У процесі створення програмних систем використовують мови порівняно низького рівня. Це призводить до ранньої деталізації операцій в процесі створення програмного забезпечення і збільшує обсяг описів продуктів, що розробляються, який, як правило, перевищує сотні тисяч операторів мови програмування. Засобів же, що дозволяють детально описувати *поведінку* складних дискретних систем на більш високому рівні, ніж універсальний мова програмування, не існує.

Колективна розробка.

Завдяки великим обсягам проектів розробка програмного забезпечення здійснюється колективом фахівців. Працюючи в колективі, окремі фахівці повинні взаємодіяти один з одним, забезпечуючи цілісність проекту, що за відсутності задовільних засобів опису поведінки складних систем досить складно. Причому, чим більший колектив розробників, тим складніше організувати процес роботи [8].

Необхідність збільшення ступеня повторюваності кодів.

На складність програмного продукту, що розробляється, впливає і те, що для збільшення продуктивності праці компанії прагнуть до створення бібліотек компонентів, які можна було б використовувати в подальших розробках. Однак у цьому випадку компоненти приходиться робити більш універсальними, що зрештою збільшує складність розробки.

Разом узяті, ці фактори суттєво збільшують складність процесу розробки. Однак очевидно, що всі вони безпосередньо пов'язані зі складністю об'єкта розробки - програмної системи.

1.4.1. Блочно-ієрархічний підхід до створення складних систем

Практика показує, що переважна більшість складних систем як у природі, так і в техніці має ієрархічну внутрішню структуру. Це пов'язано з тим, що зазвичай зв'язки елементів складних систем різні як за типом, так і за силою, що і дозволяє розглядати ці системи як деяку *сукупність взаємозалежних підсистем*. Внутрішні зв'язки елементів таких підсистем сильніші, ніж зв'язки між підсистемами.

Наприклад, комп'ютер складається з процесора, пам'яті і зовнішніх пристроїв, а Сонячна система включає Сонце і планети, що обертаються

навколо нього. У свою чергу, використовуючи ті ж розходження зв'язків, можна кожну підсистему розділити на підсистеми і т. д. до самого нижнього «елементарного» рівня, причому вибір рівня, компоненти якого слід вважати елементарними, залишається за дослідником. На елементарному рівні система, як правило, складається з небагатьох типів підсистем, по-різному скомбінованих і організованих. *Ієрархії такого типу отримали назву «ціле-частина»*.

Поведінка системи в цілому зазвичай виявляється складніше поведінки окремих частин, причому завдяки більш сильним внутрішнім зв'язкам особливості системи в основному обумовлені відносинами між її частинами, а не частинами як такими.

У природі існує *ще один вид ієрархії - ієрархія «просте-складне» або ієрархія розвитку* (ускладнення) систем в процесі еволюції. У цій ієрархії будь-яка функціонуюча система є результатом розвитку більш простої системи. Саме даний вид ієрархії реалізується механізмом успадкування об'єктно-орієнтованого програмування.

Будучи в значній мірі відображенням природних і технічних систем, програмні системи зазвичай є ієрархічними, тобто мають описані вище властивості. На цих властивостях ієрархічних систем будується *блочно-ієрархічний підхід до їх дослідження або створення*. Цей підхід передбачає спочатку створювати частини таких об'єктів (блоки, модулі), а потім збирати з них сам об'єкт.

Процес розбиття складного об'єкта на порівняно незалежні частини отримав назву *декомпозиції* .

При декомпозиції враховують, що зв'язки між окремими частинами повинні бути слабшим, ніж зв'язки елементів усередині частин. Крім того, щоб з отриманих частин можна було зібрати об'єкт, що розробляється, в процесі декомпозиції необхідно визначити всі види зв'язків частин між собою.

При створенні дуже складних об'єктів процес декомпозиції виконується багаторазово: кожен блок, у свою чергу, декомпозується на частини, поки не отримують блоки, які порівняно легко розробити.

Даний метод розробки отримав назву *покрокової деталізації.*

Істотно і те, що в процесі декомпозиції намагаються виділити аналогічні блоки, які можна було б розробляти на загальній основі. Таким чином, як уже згадувалося вище, забезпечують збільшення ступеня повторюваності кодів і, відповідно, зниження вартості розробки.

Результат декомпозиції зазвичай представляють у вигляді *схеми ієрархії,* на нижньому рівні якої розміщають відносно прості блоки, а на верхньому - об'єкт, що підлягає розробці.

На кожному ієрархічному рівні опис блоків виконують з певним ступенем деталізації, *абстрагуючись* від несуттєвих деталей. Отже, для кожного рівня використовують свої форми документації і свої моделі, що відображають сутність процесів, які виконуються кожним блоком. Так, для

об'єкта в цілому, як правило, вдається сформулювати лише найзагальніші вимоги, а блоки нижнього рівня повинні бути специфіковані так, щоб з них дійсно можна було зібрати працюючий об'єкт. Іншими словами, *чим більше блок, тим більш абстрактним має бути його опис* (рис. 1.1).

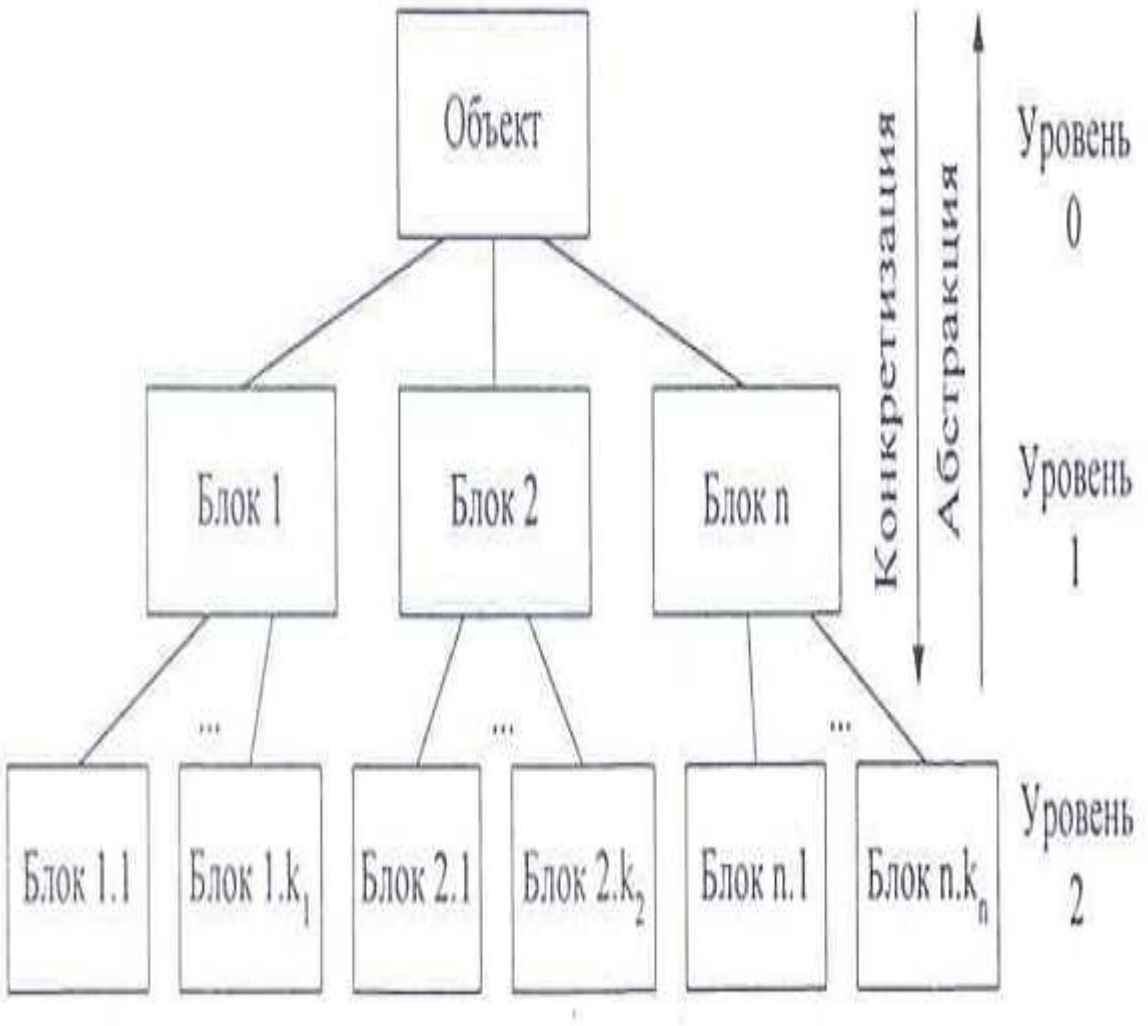

Рисунок1.1. – Співвідношення абстракт око та конкретного при описі блоків у блочно-ієрархічному підході

При дотриманні цього принципу розробник зберігає можливість осмислення проекту і, отже, може приймати найбільш правильні рішення на кожному етапі, що називають *локальною оптимізацією* (на відміну від глобальної оптимізації характеристик об'єктів, яка для дійсно складних об'єктів не завжди можлива).

Примітка. Слід мати на увазі, що поняття складного об'єкта в міру вдосконалення технологій змінюється, і те, що було складним вчора, не обов'язково залишиться складним завтра.

Отже, *в основі блочно-ієрархічного підходу лежать декомпозиція і ієрархічне впорядкування*. Важливу роль відіграють також такі принципи:

- несуперечність – контроль узгодженості елементів між собою;
- повнота – контроль на присутність зайвих елементів;
- формалізація – строгість методичного підходу;
- повторюваність – необхідність виділення однакових блоків для здешевлення і прискорення розробки;
- локальна оптимізація – оптимізація в межах рівня ієрархії.

Сукупність мов моделей, постановок завдань, методів описів деякого ієрархічного рівня прийнято називати *рівнем проектування*.

Кожен об'єкт у процесі проектування, як правило, доводиться розглядати з кількох сторін. Різні погляди на об'єкт проектування прийнято називати *аспектами проектування*.

Крім того, що використання блочно-ієрархічного підходу дає можливість створення складних систем, він також:

- спрощує перевірку працездатності як системи в цілому, так і окремих блоків;
- забезпечує можливість модернізації систем, наприклад, заміни ненадійних блоків із збереженням їх інтерфейсів.

Необхідно відмітити, що використання блочно-ієрархічного підходу стосовно програмним системам стало можливим тільки після конкретизації загальних положень підходу та внесення деяких змін у процес проектування. При цьому *структурний підхід* враховує тільки властивості ієрархії «ціле-частина», а об'єктний - використовує ще й властивості ієрархії «просте-складне».

Ключові терміни

Технологія	комплекс організаційних заходів, операцій і прийомів, які спрямовані на виготовлення, обслуговування, ремонт та / або експлуатацію виробів з номінальною якістю і оптимальними витратами.
Технологія розробки програмного забезпечення (ПЗ)	комплекс організаційних заходів, операцій і прийомів, спрямованих на розробку програмних продуктів високої якості в рамках відведеного бюджету і в строк.
Життєвий цикл програмного забезпечення (ЖЦПЗ)	період часу, який починається з моменту прийняття рішення про необхідність створення програмного продукту і закінчується в момент його повного вилучення з експлуатації.
Каскадна (водоспадна) модель ЖЦПЗ	послідовне виконання етапів створення ПЗ.
Ітераційна спіральна модель ЖЦПЗ	розробка ПЗ здійснюється по спіралі, кожен виток (ітерація) якої передбачає реалізацію певного функціоналу програмної системи.
Інкрементна ітераційна модель ЖЦПЗ	розробка ПЗ реалізується декількома ітераціями з поступовим нарощуванням функціональності системи.

2 ЖИТТЄВИЙ ЦИКЛ І ЕТАПИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У процесі створення ПЗ можна виділити 4 базових етапи / стадії (рис.2.1):

- Специфікація - визначення основних вимог.
- Розробка – створення ПЗ у відповідності зі специфікаціями.
- Тестування - перевірка ПЗ на відповідність вимогам клієнта.
- Супровід / Модернізація – розвиток ПЗ відповідно до змін потреб замовника.

Рисунок 2.1 – Схема життя програмного забезпечення

2.1 Процеси життєвого циклу програмного забезпечення

Життєвим циклом програмного забезпечення називають період від моменту появи ідеї створення деякого програмного забезпечення до моменту завершення його підтримки фірмою-розробником або фірмою, яка виконувала супровід. Життєвий цикл програмного забезпечення – період часу, що починається з моменту прийняття рішення про необхідність створення програмного продукту і закінчується в момент його повного вилучення з експлуатації

Склад процесів життєвого циклу регламентується міжнародними стандартами

ISO / 1EC 12207: 1995 – *«Information Technologe - Software Life Cycle Processes»* («Інформаційні технології – процеси життєвого циклу програмного забезпечення »).

ISO - *International Organization for Standardization* **–** Міжнародна організація по стандартизації.

IЄC – *International Electrotechnical Com* Mission – Міжнародна комісія з електротехніки. Цей стандарт описує структуру життєвого циклу програмного забезпечення та його процеси.

Процес життєвого циклу визначається як сукупність взаємопов'язаних дій, які перетворюють деякі вхідні дані у вихідні.

На рисунку 1.2 представлені процеси життєвого циклу за вказаним стандартом. Кожен процес характеризується певними завданнями і методами їх вирішення, а також вихідними даними і результатами.

Рис. 1.9. Структура процессов жизненного цикла программного обеспечения

Перехід від ручних засобів розробки ПЗ до промислового виробництва програм потребував розвитку теоретичних основ розробки ПЗ. Постійна необхідність внесення змін до програми викликана як помилками, так і розвитком вимог до них, є принциповою властивістю програмного забезпечення.

Діяльність, пов'язана з вирішенням широкого ряду завдань для постійного розвитку, отримала назву *супроводу програмного забезпечення*.

Якщо зусилля, що спрямовані на модернізацію ПЗ, перевищують вигоду від його використання, говорять про *моральне старіння програм*.

Оскільки розробка і супровід ПЗ фактично є проектною діяльністю, частина ключових понять управління проектів знайшла широке застосування в *програмній інженерії*. Таким і поняття життєвого циклу проекту (Project Lifecycle Management, PLM), що в програмной інженерії трансформувалося в поняття життєвого циклу програмного забезпечення.

*ГОСТ 34.601-90 – автоматизовані системи. Стадії створення. Дата введення з 01.01.1992р. – к*омплекс стандартів на автоматизовані системи, що визначає життєвий цикл автоматизованої системи (АС) як сукупність взаємопов'язаних процесів створення та послідовної зміни стану АС, від формування вхідних вимог до неї до закінчення експлуатації та утилізації комплексу засобів автоматизації.

Модель ЖЦ є абстракцією реального процесу, в якій відсутні деталі, несуттєві з точки зору призначення моделі. Поняття ЖЦ виникло під впливом потреби в систематизації робіт у процесі розробки ПЗ. Систематизація була першим етапом на шляху до автоматизації процесу розробки ПЗ.

Наступними кроками переходу до автоматизації процесу розробки ПЗ були такі: встановлення технологічних маршрутів розробки ПЗ; визначення можливості їх автоматизації та виявлення ризиків; розробка інструментів для автоматизації.

Спочатку з'явилися *інструменти підтримки розробки програмного коду і налагодження програм (60-і рр)*. Після усвідомлення недостатності таких засобів для суттєвого підвищення якості програм і створення інструментів управління процесом розробки виникло поняття життєвого циклу ПЗ. Виявлення закономірностей розвитку програмного забезпечення відразу показало нерозвиненість методик конструювання ПЗ і недостатність тестування для визначення якості програмних продуктів. Також на цьому етапі стало зрозуміло, що нечіткість завдання на створення ПЗ викликає більшість проблем розробки та перевірки програм.

У результаті виникли вимоги до постановки задачі, сформувалися підходи до управління вимогами на етапі аналізу та інструменти зв'язку вимог на етапах аналізу та реалізації.

Використання поняття життєвого циклу дозволяє вибрати найбільш ефективні підходи для задач певного етапу життя ПЗ. Залежно від особливостей процесів розробки і супроводу програм існують різні моделі ЖЦ.

Використання певної моделі ЖЦ дозволяє визначитися з основними моментами процесу замовлення, розробки і супроводу ПЗ навіть недосвідченому програмісту. Також використання моделей дозволяє чітко зрозуміти, в який період переходити від версії до версії, та які дії з удосконалення виконувати і на якому етапі.

Знання про закономірності розвитку програмного продукту, які відображаються в обраній моделі ЖЦ, дозволяють отримати надійні орієнтири для планування процесу розробки і супроводу ПЗ, економно витрачати ресурси і підвищувати якість управління всіма процесами.

2.2 Процес розробки програмного забезпечення

Процес розробки (development process) відповідно до стандарту передбачає дії і завдання, що виконуються розробником, і охоплює роботи зі створення програмного забезпечення і його компонентів відповідно до заданих вимог, включаючи оформлення проектної та експлуатаційної документації, а також підготовку матеріалів, необхідних для перевірки працездатності і відповідності якості програмних продуктів, матеріалів, необхідних для навчання персоналу, і т. д.

За стандартом процес розробки включає такі дії :

- *підготовчу роботу* – вибір моделі життєвого циклу (див. далі), стандартів, методів і засобів розробки, а також складання плану робіт;
- *аналіз вимоги до системи* – визначення її функціональних можливостей, користувальницьких вимог, вимог до надійності і безпеки, вимог до зовнішніх інтерфейсів і т. д;
- *проектування архітектури системи* – визначення складу необхідного обладнання, програмного забезпечення та операцій, які виконуються обслуговуючим персоналом;
- *аналіз вимог до програмного забезпечення* – визначення функціональних можливостей, включаючи характеристики продуктивності, середовища функціонування компонентів, зовнішніх інтерфейсів, специфікацій надійності та безпеки, ергономічних вимог, вимог до даних, що використовуються, установці, прийманні, користувацької документації, експлуатації та супроводу;
- *проектування архітектури програмного забезпечення* – визначення структури програмного забезпечення, документування інтерфейсів його компонентів, розробку попередньої версії документації користувача, а також вимог до тестів і плану інтеграції;
- *детальне проектування програмного забезпечення* – докладний опис компонентів програмного забезпечення та інтерфейсів між ними, оновлення користувальницької документації, розробка та документування вимог до тестів і плану тестування компонентів програмного забезпечення, оновлення плану інтеграції компонентів;
- *кодування і тестування програмного забезпечення* – розробка та документування кожного компонента, а також сукупності тестових процедур і даних для їх тестування, тестування компонентів, оновлення

документації користувача, оновлення плану інтеграції програмного забезпечення;

- *інтеграція програмного забезпечення* – збірка програмних компонентів відповідно до плану інтеграції та тестування програмного забезпечення на відповідність кваліфікаційним вимогам, що представляють собою набір критеріїв або умов, які необхідно виконати, щоб кваліфікувати програмний продукт як відповідний своїм специфікаціям і готовий до використання в заданих умовах експлуатації;
- *кваліфікаційне тестування програмного забезпечення* – тестування програмного забезпечення в присутності замовника для демонстрації його відповідності вимогам і готовності до експлуатації; при цьому перевіряється також готовність і повнота технічної документації користувача;
- *інтеграція системи* – збірка всіх компонентів системи, включаючи програмне забезпечення та обладнання;
- *кваліфікаційне тестування системи* – тестування системи на відповідність вимогам до неї і перевірка оформлення та повноти документації;
- *установка програмного забезпечення* – установка програмного забезпечення на обладнанні замовника і перевірка його працездатності;
- *приймання програмного забезпечення* – оцінка результатів кваліфікаційного тестування програмного забезпечення і системи в цілому та документування результатів оцінки спільно із замовником, остаточна передача програмного забезпечення замовнику.

Зазначені дії можна згрупувати, умовно виділивши такі ***основні етапи розробки програмного забезпечення*** (в дужках вказані відповідні стадії розробки *по ГОСТ 19.102-77 «Стадії розробки»):*

1. Постановка задачі (стадія «Технічне завдання»);

2. Аналіз вимог і розробка специфікацій (стадія «Ескізний проект»);

3. Проектування (стадія «Технічний проект»);

4. Реалізація (стадія «Робочий проект»).

Традиційно розробка також включала етап *супроводу* (початку цього етапу відповідає стадія «Впровадження» по ГОСТ). Однак за міжнародним стандартом у відповідності із змінами, що відбулися в індустрії розробки програмного забезпечення, цей процес тепер розглядається окремо.

Умовність виділення етапів пов'язана з тим, що на будь-якому етапі можливе прийняття рішень, які вимагатимуть перегляду рішень, прийнятих раніше.

Стадія – Постановка задачі

У процесі *постановки завдання* чітко формулюють призначення програмного забезпечення і визначають основні вимоги до нього. Кожна вимога є описом необхідної або бажаної властивості програмного забезпечення.

Розрізняють:

функціональні вимоги, що визначають функції, які має виконувати програмне забезпечення;

експлуатаційні вимоги, що визначають особливості його функціонування.

Вимоги до програмного забезпечення, що має *прототипи*, зазвичай визначають за аналогію, враховуючи структуру і характеристики вже існуючого програмного забезпечення.

Для формулювання вимог до програмного забезпечення, що не має аналогів, іноді необхідно провести спеціальні дослідження, що називаються передпроектними. У процесі таких досліджень визначають розрішення завдання, можливо, розробляють методи його рішення (якщо вони нові) і встановлюють найбільш суттєві характеристики програмного забезпечення, що розробляється. Для виконання передпроектних досліджень, як правило, укладають договір на виконання науково-дослідних робіт.

У будь-якому випадку етап постановки завдання закінчується розробкою *технічного завдання*, що фіксує принципові вимоги, і прийняттям основних проектних рішень (див. гл. 3).

Стадія – Аналіз вимог і визначення специфікацій

Специфікаціями називають точний формалізований опис функцій і обмежень програмного забезпечення, що розробляється. Відповідно розрізняють функціональні і експлуатаційні специфікації.

Сукупність специфікацій являє собою *загальну* логічну модель програмного забезпечення.

Для отримання специфікацій виконують аналіз вимог технічного завдання, формулюють змістовну постановку задачі, вибирають математичний апарат формалізації, будують модель предметної області, визначають підзадачі і вибирають або розробляють методи їх вирішення. Частина специфікацій може бути визначена в процесі передпроектних досліджень і, відповідно, зафіксована в технічному завданні.

На цьому етапі також доцільно сформувати тести для пошуку помилок в програмному забезпеченні, що проектується, та обов'язково вказати очікувані результати.

Стадія – Проектування

Основним завданням цього етапу є визначення *докладних* специфікацій програмного забезпечення.

Процес проектування складного програмного забезпечення зазвичай включає:

проектування загальної структури - визначення основних компонентів і їх взаємозв'язків;

декомпозицію компонентів і побудову структурних ієрархій відповідно до рекомендацій блочно-ієрархічного підходу;

проектування компонентів.

Результатом проектування є *детальна модель* програмного забезпечення разом зі специфікаціями його компонентів всіх рівнів. Тип моделі залежить від обраного підходу (структурний, об'єктний або компонентний) і конкретної технології проектування. Однак у кожному разі процес проектування охоплює як проектування програм (підпрограм) і визначення взаємозв'язків між ними, так і проектування даних, з якими взаємодіють ці програми або підпрограми.

Прийнято розпізнавати також ***два аспекти проектування****:*

- *логічне проектування*, яке включає ті проектні операції, які безпосередньо не залежать від наявних технічних і програмних засобів, що складають середовище функціонування майбутнього програмного продукту;
- *фізичне проектування* – прив'язка до конкретних технічних і програмних засобів середовища функціонування, тобто врахування обмежень, що визначені у специфікаціях.

Стадія – Реалізація

Реалізація являє собою процес поетапного написання кодів програми обраною мовою програмування (кодування), їх тестування і налагодження.

Стадія – Супровід

Супровід – це процес створення і впровадження нових версій програмного продукту. Причинами випуску нових версій можуть служити:

необхідність виправлення помилок, виявлених в процесі експлуатації попередніх версій;

необхідність удосконалення попередніх версій, наприклад, поліпшення інтерфейсу, розширення складу виконуваних функцій або підвищення його продуктивності;

зміна середовища функціонування, наприклад, поява нових технічних засобів та / або програмних продуктів, з якими взаємодіє програмне забезпечення, що супроводжується.

На цьому етапі в програмний продукт вносять необхідні зміни, які так само як і в інших випадках, можуть зажадати перегляду проектних рішень, прийнятих на будь-якому попередньому етапі.

Зі зміною моделі життєвого циклу програмного забезпечення роль цього етапу істотно зросла, так як продукти тепер створюються ітераційно:

спочатку випускається порівняно проста версія, потім наступна з великими можливостями, потім наступна і т. д. Саме це і послужило причиною виділення етапу супроводу в окремий процес життєвого циклу відповідно до стандарту 1SO / IEC 12207.

Розглянутий стандарт тільки називає і визначає процеси життєвого циклу програмного забезпечення, не конкретизуючи в деталях як реалізовувати або виконувати дії і завдання, що включені в ці процеси. Ці питання регламентуються відповідними методами, методиками і т. п. Перш ніж перейти до докладного розгляду останніх, проаналізуємо еволюцію схем розробки програмного забезпечення від моменту їх появи до теперішнього часу.

2.3 Еволюція моделей життєвого циклу програмного забезпечення

Моделі життєвого циклу є основою знань технологій програмування та інструментарію, які підтримує. У що технологія базується на певних уявленнях про життєвий цикл і організовує свої методи і інструменти навколо фаз і етапів ЖЦ.

Розвиток методологій програмування в 70-х рр XX ст. привів до формування потреби вивчення життєвого циклу ПЗ. До цього часу моделі ЖЦ розвиваються і модифікуються, уточнюючи і доповнюючи дві базові моделі – каскадну і ітеративну. Ці зміни обумовлені необхідністю організаційної та технологічної підтримки проектів з розробки ПЗ.

Перехід від ручних засобів розроблення ПЗ до промислового виробництва програм потребував розвитку теоретичних основ розроблення ПЗ. Постійна необхідність внесення змін у програми як спричинена помилками, так і розвитком вимог до них, є принциповою властивістю програмного забезпечення. Діяльність, пов’язана з рішенням широкого ряду завдань для постійного розвитку, отримала назву супроводу програмного забезпечення. Якщо зусилля, спрямовані на модернізацію ПЗ, перевищують вигоду від його використання, говорять про моральне старіння програм.

Оскільки розроблення та супровід ПЗ фактично є проектною діяльністю, частина ключових понять управління проектів знайшла широке застосування у програмній інженерії. Таким є і поняття життєвого циклу проекту (Project Lifecycle Management, PLM), що в програмній інженерії трансформувалось у поняття життєвого циклу програмного забезпечення.

Протягом останніх тридцяти років у програмуванні змінилися три моделі життєвого циклу програмного забезпечення; каскадна (водоспадна), модель з проміжним контролем і спіральна.

2.4 Каскадна модель (waterflow model)

Водоспадна модель (каскадна схема) запропонована в 1970 році Вінстоном Ройсом. і використовувалася в розробці програмного

забезпечення (рис. 2.2), яке передбачало, що перехід на наступну стадію здійснюється після того, як повністю будуть завершені проектні операції попередньої стадії і отримані всі вихідні дані для наступної стадії.

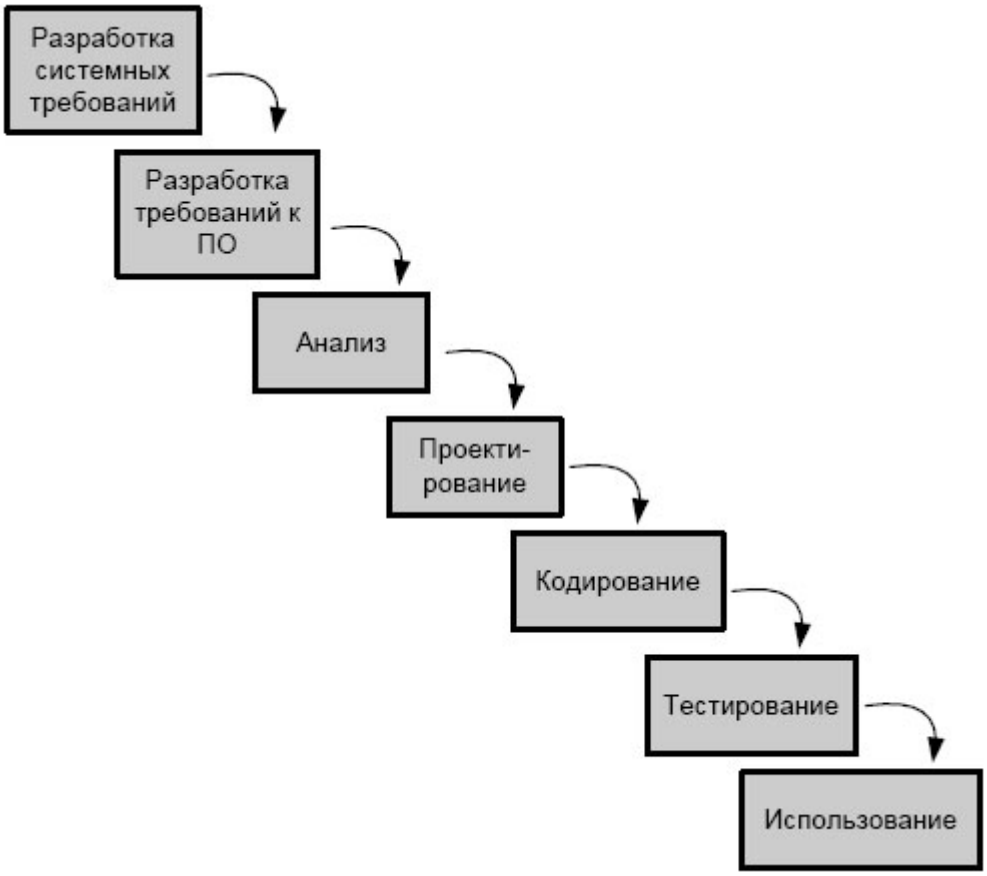

Рисунок 2.2 – Каскадна схема розробки програмного забезпечення

Фактично вперше в процесі розробки ПЗ були виділені різні кроки розробки і розхитані примітивні уявлення про розробку ПЗ у вигляді аналізу системи та її кодування.

Перевагами такої схеми є :

- отримання в кінці кожної стадії закінченого набору проектної документації, що відповідає вимогам повноти і узгодженості;
- простота планування процесу розробки.

Саме таку схему і використовують зазвичай при блочно-ієрархічному підході до розробки складних технічних об'єктів, що забезпечує дуже високі параметри ефективності розробки.

Однак дана схема виявилася придатною тільки до створення систем, для яких в самому початку розробки вдавалося точно і повно сформулювати всі вимоги. Це зменшувало ймовірність виникнення в процесі розробки проблем, що пов'язані з прийняттям невдалого рішення на попередніх стадіях. На практиці такі розробки зустрічається вкрай рідко.

Каскадна модель ЖЦ ПЗ виникла для задоволення потреби в систематизації робіт ще на ранніх етапах розробки програм. Відповідно до цієї моделі програмні системи проходять у своєму розвитку дві фази: розробка, супровід.

Фази розбиваються на ряд етапів. Каскадна модель передбачає послідовне виконання всіх етапів проекту в суворо фіксованому порядку. Перехід на наступний етап означає повне завершення робіт на попередньому етапі (рис. 2.2.. Розробка починається з ідентифікації потреби в новому додатку, а закінчується передачею продукту розробки в експлуатацію. Всі етапи розробки програмного забезпечення регламентуються стандартами підприємства і державним з стандартам ГОСТ 34.601-90.

Першим етапом фази розробки специфікація (Requirements Specification) – розробка системних вимог, розробка вимог до ПЗ .

На етапі постановки завдання замовник спільно з розробниками приймають рішення про створення системи. Визначення вимог включає опис загального контексту завдання, очікуваних функцій системи та її обмежень. У разі позитивного рішення починається аналіз системи відповідно до вимог. Розробники програмного забезпечення намагаються осмислити висунуті замовником вимоги і зафіксувати їх у вигляді специфікацій системи. *Призначення специфікацій – описувати зовнішню поведінку системи, а не її внутрішню організацію.* Перш ніж почати створювати проект за специфікаціями, вимоги мають бути ретельно перевірені на відповідність вихідним цілям, повноту, сумісність (несуперечливість) і однозначність**.**

Завдання етапу аналізу полягає в тому, щоб вибудувати опис програми у вигляді логічної системи, зрозумілої як для замовника та майбутніх користувачів, так і для виконавців проекту. На етапі специфікації обов'язково формується технічне завдання на розробку ПЗ.

Розробка проектних рішень, що відповідають на питання, як має бути реалізована система, щоб вона могла задовольняти певні вимоги, виконується на *етапі проектування (Design).* Оскільки складність системи в цілому може бути дуже великою, головним завданням цього етапу є послідовна декомпозиція системи до рівня очевидно реалізованих модулів або процедур. Результати виконання цього етапу оформляються як технічний проект, вимоги до документів якого встановлені стандартом *ГОСТ 34.201-89.*

На наступному *етапі реалізації (Construction), або кодування*, кожен з модулів програмується на найбільш придатній для даного застосування мові. З точки зору автоматизації цей етап традиційно є найбільш розвинений.

У каскадної моделі фаза розробки закінчується *етапом тестування (Testing and debugging),* автономного і комплексного, і передачею системи в експлуатацію (Installation).

Фаза експлуатації (Використання) та супроводу включає в себе всю діяльність щодо забезпечення нормального функціонування програмних систем, у тому числі фіксування розкритих під час виконання програм

помилок, пошук їх причин та виправлення, підвищення експлуатаційних характеристик системи, адаптацію системи до навколишнього середовища, а також, при необхідності, і більш істотні роботи з удосконалення системи. Фактично відбувається еволюція системи.

У ряді випадків на цю фазу припадає більша частина коштів, які витрачаються в процесі життєвого циклу програмного забезпечення. Зрозуміло, що увага програмістів до тих чи інших етапів розробки залежить від конкретного проекту. Часто розробнику немає необхідності проходити через всі етапи, наприклад, якщо створюється невелика, добре зрозуміла програма з чітко поставленою метою.

Коротка характеристика каскадної схеми розробки програмного забезпечення:

- фіксований набір стадій;
- кожна стадія закінчується документованим результатом;
- наступна стадія починається тільки після закінчення попередньої.

Недоліки:

- негнучкість;
- фаза повинна бути завершена до переходу до наступної;
- набір фаз фіксований;
- важко реагувати на зміни вимог.

Використання: там, де вимоги добре зрозумілі і стабільні.

Необхідність повернень на попередні стадії обумовлена такими причинами:

- неточні специфікації, уточнення яких у процесі розробки може призвести до необхідності перегляду вже прийнятих рішень;
- зміна вимог замовника безпосередньо в процесі розробки;
- швидке моральне старіння технічних і програмних засобів, що використовуються;
- відсутність задовільних засобів опису розробки на стадіях постановки задачі, аналізу та проектування.

Відмова від уточнення (зміни) специфікацій призведе до того, що закінчений продукт не буде задовольняти потребам користувачів.

При відмові від врахування зміни обладнання і програмного середовища користувач отримає морально застарілий продукт.

Відмова від перегляду невдалих проектних рішень призводить до погіршення структури програмного продукту і, відповідно, ускладнить, розтягне за часом і здорожує процес його створення. Реальний процес розробки, таким чином, носить ітераційний характер.

2.5 Ітеративна модель (Iterative and incremental development) – модель з проміжним контролем

Каскадна модель життєвого циклу не є ідеальною, оскільки лише дуже прості завдання проходять всі етапи без будь-яких ітерацій (повернень на попередні кроки процесу).

Наприклад, при програмуванні може виявитися, що реалізація деякої функції неефективна і вступає в протиріччя з вимогами до продуктивності системи. У цьому випадку потрібні зміни проекту, а можливо, і переробка специфікацій.

Для обліку повторюваності етапів процесу розробки створювалися альтернативи каскадної моделі. З таких альтернатив утворилася ***ітеративна модель*** (рис. 2.3). Ця модель передбачає розбиття життєвого циклу проекту на послідовність ітерацій, кожна з яких нагадує "міні-проект" з усіма фазами життєвого циклу.

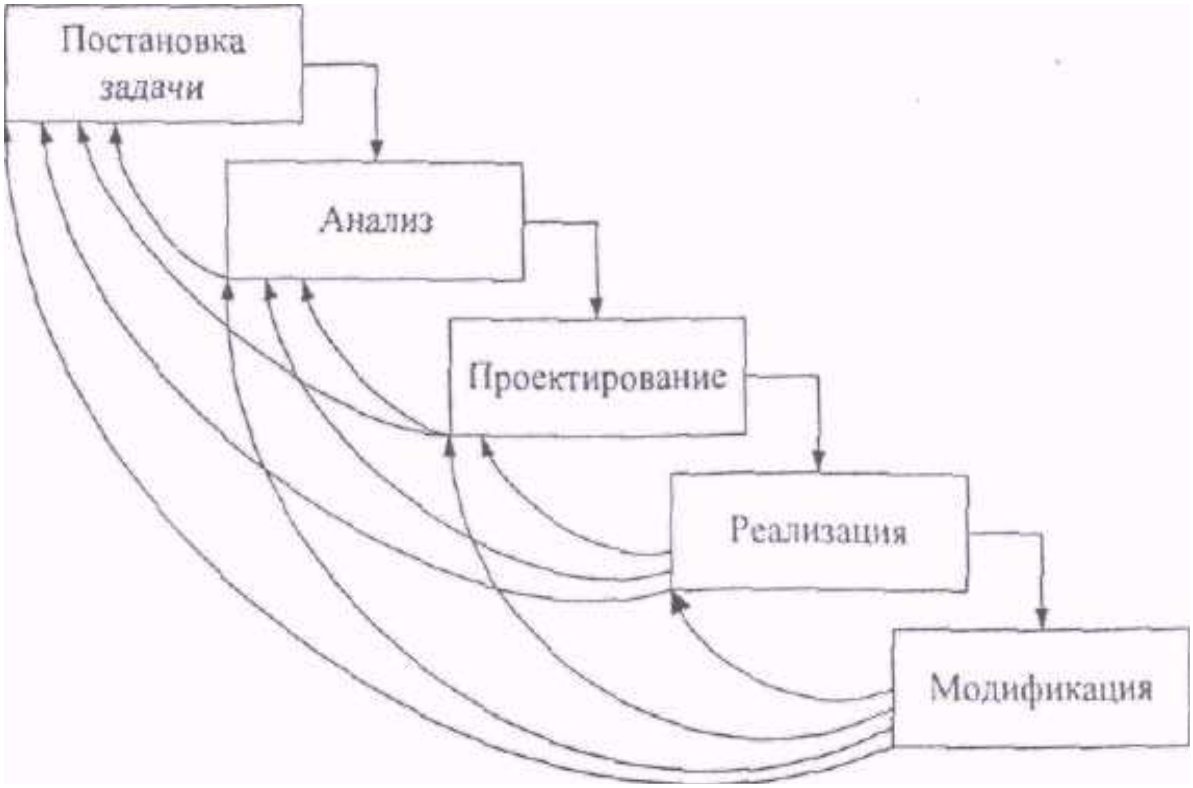

Рис. 2.3 – Ітераційна схема розробки програмного забезпечення

Класична ітераційна модель абсолютизує можливість повернень на попередні етапи. Ця обставина відображає істотний аспект програмних розробок: прагнення заздалегідь передбачити всі ситуації використання системи і неможливість в більшості випадків досягти цього. Всі традиційні технології програмування спрямовані лише на те, щоб мінімізувати

повернення. Але суть від цього не змінюється: при поверненні завжди доводиться повторювати побудову того, що вже вважалося готовим.

Схема, що підтримує ітераційний характер процесу розробки, була названа схемою з проміжним контролем. Контроль, який виконується за даною схемою після завершення кожного етапу, дозволяє при необхідності повернутися на будь-який рівень і внести необхідні зміни. Основна небезпека використання такої схеми пов'язана з тим, що розробка ніколи не буде завершена, постійно перебуваючи в стані уточнення та вдосконалення.

Мета кожної ітерації в розробці ПЗ – отримання працюючої версії програмної системи, що включає функціональність, визначену інтегрованим змістом усіх попередніх і поточної ітерацій. Результат фінальної ітерації містить всю необхідну функціональність продукту. Таким чином, із завершенням кожної ітерації продукт розвивається інкрементно (нарощує функціональність).

З точки зору структури життєвого циклу і процесу розробки така модель є ***ітеративною (iterative)***, а з точки зору розвитку продукту – ***інкрементною (incremental)***. Досвід індустрії розробки ПЗ показує, що неможливо розглядати кожен з цих поглядів ізольовано. Тому цю модель часто називають моделлю *ітеративної і інкрементної розробки (Iterative and incremental development, IID).*

Примітка. Народна мудрість в подібних випадках говорить «краще - ворог хорошого». Залишилося тільки зрозуміти, що можна вважати «хорошим» і як все-таки домогтися кращого ...

Різні варіанти ітераційного підходу реалізовані в більшості сучасних методологій розробки (RUP, MSF, XP).

Стисла характеристика: стадії повторюються неодноразово. Недоліки: система часто погано структурована, проект «непрозорий», після уточнення вимог відкидається частина виконаної раніше роботи; потрібні засоби для швидкого розробки. Використання: підходить для малих і середніх проектів.

2.6 Спіральна модель

Для подолання перелічених проблем в середині 80-х років XX ст. була запропонована спіральна схема (рис. 2.4). Відповідно до даної схеми програмне забезпечення створюється не відразу, а ітераційно з використанням методу прототипування, що базується на створенні прототипів. Саме поява прототипування призвело до того, що процес модифікації програмного забезпечення перестав сприйматися, як «необхідне зло», а став сприйматися як окремий важливий процес.

Прототипом називають діючий програмний продукт, який реалізує окремі функції і зовнішні інтерфейси розроблюваного програмного забезпечення.

На першій ітерації, як правило, специфікують, проектують, реалізують і тестують інтерфейс користувача. На другий - додають деякий обмежений набір функцій.

На наступних етапах цей набір розширюють, нарощуючи можливості даного продукту.

Рисунок 2.4 – Спіральна схема розробки програмного забезпечення

Основною перевагою даної схеми є те, що, починаючи з деякої ітерації, на якій забезпечена певна функціональна повнота, продукт можна надавати користувачеві, що дозволяє:

- скоротити час до появи перших версій програмного продукту;
- зацікавити велику кількість користувачів, що забезпечує швидке просування наступних версій продукту на ринку;
- прискорити формування та уточнення специфікацій за рахунок появи практики використання продукту;
- зменшити ймовірність морального старіння системи за час розробки.

Основною проблемою використання спіральної схеми є визначення моментів переходу на наступні стадії. Для її вирішення зазвичай обмежують терміни проходження кожної стадії, ґрунтуючись на експертних оцінках.

У спіральній моделі розроблення програми має вигляд серії послідовних ітерацій. На перших етапах уточнюються специфікації

продукту, на наступних – додаються нові можливості функції. *Мета цієї моделі* – по закінченні кожної ітерації заново здійсніть оцінку ризиків продовження робіт.

На кожному витку спіралі створюють чергову версію продукту, уточнюють вимоги проекту, визначають його якість и планують роботи наступного витка. Особлива увага приділяється початковим етапам розроблення – аналізу і проектуванню, де реалізованість тих чи інших технічних рішень перевіряється і обґрунтовується за допомогою створення прототипів (макетування).

Завдяки ітеративній природі спіральна модель допускає коригування у ході роботи, що сприяє поліпшенню продукту.

При великому числі ітерацій розробка по цій моделі вимагає глибокої автоматизації всіх процесів, інакше вона стає неефективною. На практиці у замовників і користувачів іноді виникає відчуття нестабільності продукту, оскільки вони не встигають стежити за швидкими змінами в ньому.

Спіральна модель розробки ПЗ вимагає визначення ключових точок проекту – *milestones.*

Виділяють такі основні контрольні точки :

1. Concept of Operations (COO) – концепція використання системи;
2. Life Cycle Objectives (LCO) – цілі і зміст життєвого циклу;
3. Life Cycle Architecture (LCA) – архітектура життєвого циклу;
4. Initial Operational Capability (IOC) – перша версія ПЗ, перевіряється в ході дослідницької експлуатації;
5. Final Operational Capability (FOC) – готове ПЗ, яке експлуатується в реальних умовах /

Коротка характеристика спіральної схеми розробки програмного забезпечення:

- проект має контрольні точки – milestones;
- на кожному витку спіралі створюється прототип;
- на виході – продукт, що відповідає вимогам користувачів.

Переваги:

- ранній аналіз можливостей повторного використання;
- наявні механізми досягнення параметрів якості;
- модель дозволяє контролювати джерела проектних робіт і відповідних витрат;
- модель дозволяє вирішувати інтегровані завдання системного розробки, які охоплюють програмну і апаратну складові створюваного продукту.

*Недоліки: п*ісля уточнення вимог відкидається частина раніше виконаної роботи; потрібні кошти для швидкого розробки.

Використання: підходить для малих і середніх проектів.

Аналіз в ключових точках особливо актуальний для менеджерів і лідерів проектів, які відстежують хід виконання проекту і планують подальші роботи (рис. 2.5).

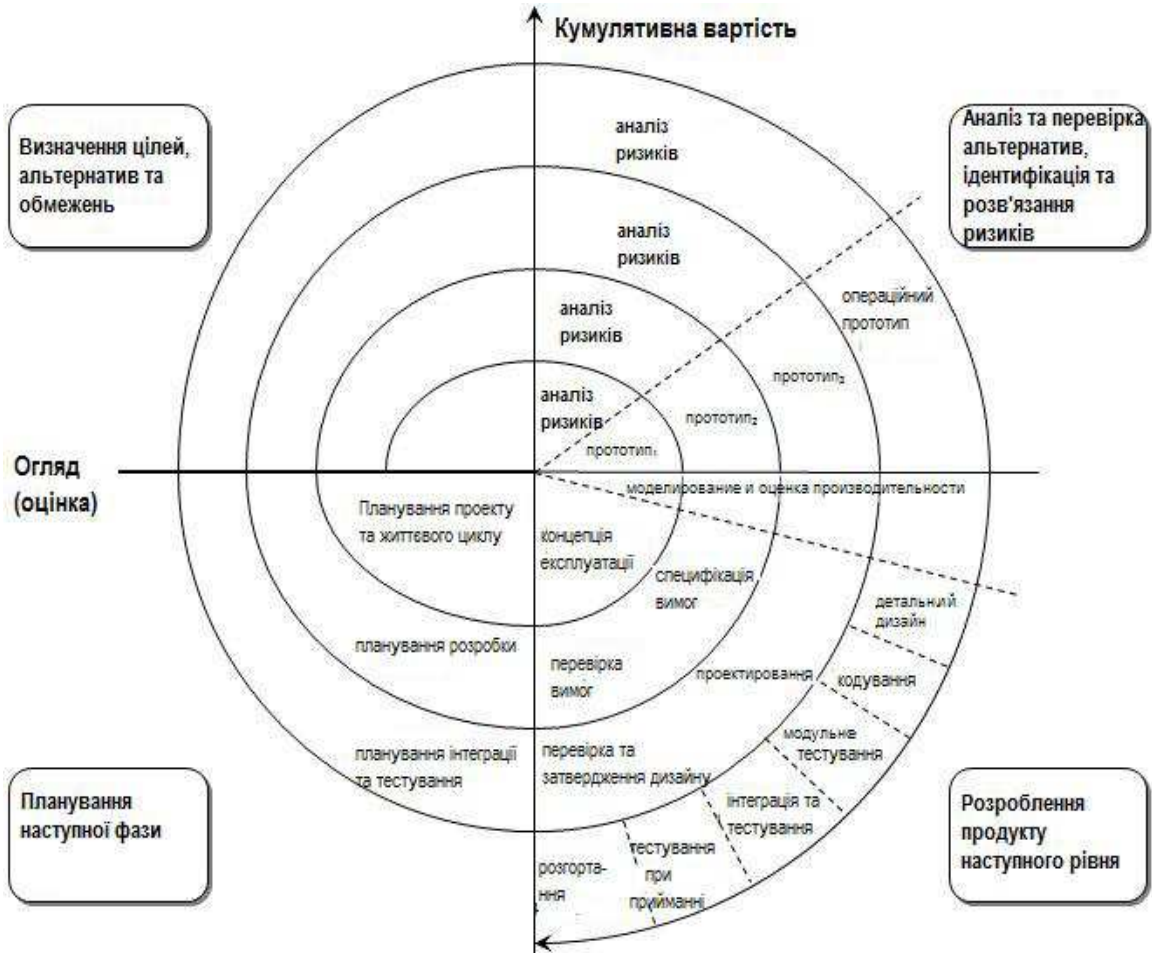

Рисунок 2.5 – Розширена спіральна схема розробки програмного забезпечення

Незважаючи на розвиток теоретичної бази, стадія комплексної автоматизації технологій програмування стала можливою лише при відповідному рівні розвитку техніки. Суттєво вплинуло на перехід до комплексної автоматизації усвідомлення того, що не можна розвивати промислове програмування без підтримки технологічних функцій на всіх етапах життя програм. На початку 90 - х рр. з'явилася CASE - технологія (Computer Added Software Engineering – сукупність інструментів і методів програмної інженерії проектування ПЗ для забезпечення високої якості, мінімальної кількості помилок і спрощення проектування), що об'єднувала системи комплексної автоматизації підтримки розробки та супроводу програм.

Питання для самоконтролю

1) Що таке життєвий цикл ПЗ?
2) Які моделі життєвого циклу ПЗ Вам відомі?
3) Що виконується під час етапів фази розроблення ПЗ?
4) Які моделі життєвого циклу дозволяють оперативно реагувати на зміну вимог до ПЗ?
5) Які характеристики спіральної моделі життєвого циклу є її перевагою? Поясніть.
6) Які ключові точки проекту розроблення ПЗ вам відомі?
7) Які параметри характеризують якість програмного продукту?
8) Як що модель життєвого циклу ПЗ має фіксований набір стадій? Яким чином була модифікована ця модель?

3 СТАНДАРТИЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розвиток будь-якої галузі економіки обов'язково супроводжується формалізацією підходів, що використовуються, і появою стандартів різного рівня. На ранніх етапах окремі підприємства формалізують внутрішні процеси, що забезпечує повторюваність результатів процесу або створення певного продукту.

Для полегшення взаємодії підприємств і зручності споживачів розробляються *галузеві стандарти*. Розвиток кожного виду господарської діяльності призводить до потреби в державних засобах забезпечення якості продукції або процесу, тому розробляються і затверджуються *державні стандарти.*

Для поліпшення умов співпраці, розробки загальнозрозумілих правил конкуренції на міжнародному ринку створюються *об'єднання галузевих органів стандартизації*, результатом діяльності яких є *регіональні стандарти* (діють в обмеженому переліку держав, що приєдналися) або *міжнародні стандарти*.

Одним з перших стандартів, що мав істотний вплив на розвиток теорії проектування та розробки ІС, був *стандарт BSP (Business System Planning).* Даний стандарт був розроблений компанією IBM в середині 70- х рр.

Процес BSP передбачав виділення в ході розробки ІС таких кроків:

- отримання підтримки керівництва;
- визначення процесів підприємства;
- визначення класів даних;
- проведення інтерв'ю;
- обробка та організація результатів інтерв'ю.

Найважливіші кроки процесу BSP спостерігаються у більшості формальних методик. На сьогодні діють такі стандарти, що регламентують процес розробки ПЗ:

ГОСТ 34.601-90 - державний стандарт, поширюється на автоматизовані системи і встановлює стадії і етапи їх створення. У стандарті міститься опис змісту робіт на кожному етапі.

ISO / IEC 12207 - Systems and software engineering - Software Life Cycle Processes - міжнародний стандарт на процеси розробки та організації життєвого циклу ПЗ. Поширюється на всі види замовленого ПЗ. Стандарт не містить опису фаз, стадій і етапів.

SWEBOK (Guide to the Software Engineering Body of Knowledge) - Посібник до зведення знань з програмної інженерії - галузевий стандарт Інституту інженерів по радіоелектроніці та електротехніці (IEEE), що містить основні види діяльності з програмної інженерії.

3.1. Міжнародні стандарти ISO

Базовим стандартом розробки ПЗ *є ISO 12207 – Systems and software engineering - Software Life Cycle Processes* , в якому всі процеси ЖЦ ПЗ розділені на три групи (рис.3.1). У табл. 1 наведено опис основних процесів ЖЦ ПЗ ІВ відповідно до ISO

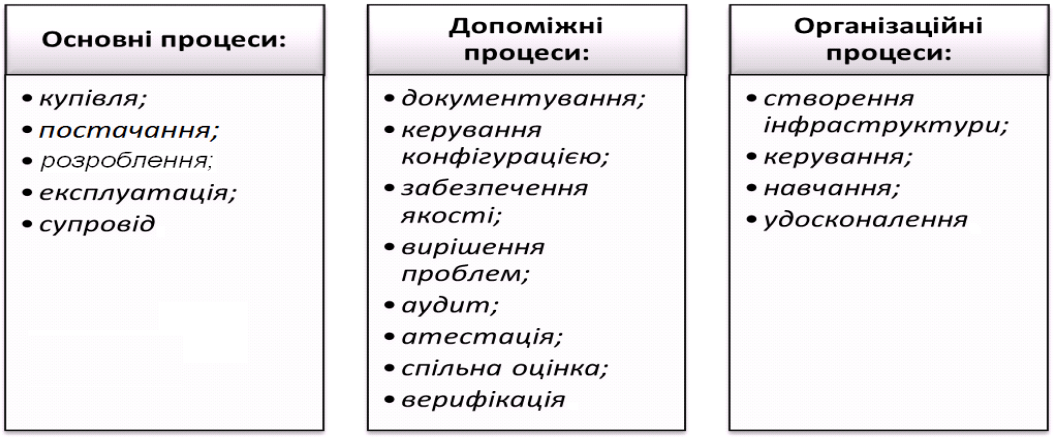

Рисунок 3.1 – Процеси ЖЦ ІС відповідно до стандарту ISO 12207

Таблиця1 - Зміст ***основних процесів*** ЖЦ ПЗ ІВ відповідно до ISO 12207

Процес (Виконавець)	**Дії**	**Вхід**	**Результат**
Купівля (Замовник)	ініціювання; підготовка вимог заявки; підготовка угоди; контроль діяльності постачальника; прийом ІС	рішення про початок; впровадження ІС; результати дослідження діяльності замовника; результати аналізу ринку ІВ / тендера; план поставки / розробки; комплексний тест ІС	техніко-економічне обґрунтування впровадження ІС; технічне завдання на ІС; угода на постачання / розробку; акти приймання етапів роботи; акт приймально-передавальних випро-бувань
Поставки (Розробник ІС)	ініціювання; відповідь на замов-лення; підготовка угоди; планування вико-нання; поставка ІС;	технічне завдання на ІС; рішення про участь у розробці; результати тендеру; технічне завдання на ІС; план управління проектом; створена ІС та документація	рішення про участь у розробці; комерційна пропозиція / конкурсна заявка; угода про постачання / розробки; план управління проектом; реалізація / коригування; акт приймально-передавальних випробувань;

Допоміжні процеси призначені для підтримки виконання основних процесів, забезпечення якості проекту, організації верифікації та тестування ПЗ.

Організаційні процеси визначають дії і завдання замовників і розробників для управління процесами в ході проекту.

Для підтримки практичного використання стандарту ISO 12207 розроблені такі технологічні документи:

- Керівництво для ISO / IEC 12207 (ISO / IEC TR 24748-3: 2011 Systems and software engineering - Life cycle management - Part 3: Guide to the application of ISO / IEC 12207 (Software life cycle processes))
- Керівництво по використанню ISO / IEC 12207 в управлінні проектами (ISO / IEC TR 16326: 2009 Systems and software engineering - Life cycle processes - Project management).

У 2002 був опублікований стандарт на процеси життєвого циклу систем ***ISO / IEC 15288 Systems and software engineering – System life cycle processes***, в розробці якого брали участь фахівці різних галузей: системної інженерії, програмування, управління якістю, людськими ресурсами, безпекою та ін.

Цей документ враховує практичний досвід створення систем в урядових, комерційних, військових та академічних організаціях і може бути застосований для широкого класу систем, але *його основне призначення - підтримка створення комп'ютеризованих систем*.

В даний час діє версія стандарту 2008. *У стандарті ISO / IEC 15288: 2008* в структурі ЖЦ виділені групи процесів за видами діяльності (рис.15).

Стадії створення системи, передбачені в стандарті ISO / IEC 15288: 2008, і основні результати, що мають бути досягнуті до моменту їх завершення, наведені в таблиці 2.

Стандарти ISO / IEC 12207 та ISO / IEC 15288 мають єдину термінологію і розроблені таким чином, щоб могли використовуватися одночасно в проекті.

Таблиця 3.2. - Стадії *створення систем* (ISO / IEC 15288)

№ п/п	Стадія	Опис
1	Формування концепції	Аналіз потреб, вибір концепції та проектних рішень
2	Розробка	Проектування системи
3	Реалізація	Створення системи
4	Експлуатація	Введення в експлуатацію і використання системи
5	Підтримка	Забезпечення функціонування системи
6	Зняття з експлуатації	Зупинка використання, демонтаж, архівування системи

Також потрібно відзначити, що в процесі промислової розробки ПЗ обов'язково використовуються ***стандарти якості серії ISO 9000*** .

Серія ISO 9000 - (управління якістю) включає в себе такі стандарти:

ISO 9000-1. Управління якістю і гарантії якості. Частина 1. Керівництво з вибору і використання.

ISO 9000-2. Управління якістю і гарантії якості. Частина 2. Загальне керівництво по застосуванню стандартів ISO 9001, ISO 9002 та ISO 9003. ISO 9000-3. Управління якістю і гарантії якості. Частина 3.

Керівництво по застосуванню стандарту ISO 9001 при розробці, установці і супроводі ПЗ. ISO 9000-4. Управління якістю і гарантії якості. Частина 4. Керівництво з управління надійністю програм.

Основний стандарт ISO 9001: 2009 задає модель системи якості для процесів проектування, розробки, виробництва, установки і обслуговування (продукту, системи, послуги).

У моделі ISO 9000 лише згадуються вимоги, які повинні бути реалізовані, але не говориться, як це можна зробити. Тому для побудови повноцінної системи якості ISO, крім основної моделі ISO 9001, необхідно використовувати допоміжні галузеві і рекомендаційні стандарти.

3.2 Стандарти організації IEEE – Institute of Electrical and Electronics Engineers

У 1963 в результаті злиття Інституту радіотехніки (Institute of Radio Engineers, IRE) і Американського інституту інженерів - електриків (American Institute of Electrical Engineers, AIEE) була створена міжнародна некомерційна асоціація технічних фахівців - світовий лідер в області розробки стандартів з радіоелектроніки та електротехніки – інститут інженерів з радіоелектроніки та електротехніки IEEE (Institute of Electrical and Electronics Engineers). Дана міжнародна організація об'єднує понад 400 тисяч фахівців з 170 країн.

IEEE здійснює інформаційну та матеріальну підтримку фахівців для організації та розвитку наукової діяльності в електротехніці, електроніці, комп'ютерній техніці та інформатиці.

Керівництво до зведення знань з програмної інженерії ***(SWEBOK****, 2004)* містить опис 10 галузей знань:

Software requirements – програмні вимоги;

Software design – дизайн (архітектура);

Software construction – конструювання програмного забезпечення;

Software testing – тестування;

Software maintenance – експлуатація (підтримка) програмного забезпечення;

Software configuration management – управління конфігураціями;

Software engineering management – управління з програмної інженерії ;

Software engineering process – процеси програмної інженерії;

Software engineering tools and methods – інструменти та методи;

Software quality–- якість програмного забезпечення.

Для кожної галузі ***SWEBOK*** містить опис ключових елементів у вигляді *підобластей (subareas*).

Для кожної підобласті приведена декомпозиція в *вигляді списку тем (topics)* з їх описом.

3.3 Стандарт зрілості компанії-розробника програмного забезпечення CMM - Capability Maturity Model

Говорячи про стандартизацію процесів підприємства потрібно розглянути модель зрілості технологічних процесів організації Capability Maturity Model (CMM), яка розроблена Інститутом інженерів програмного забезпечення (Software Engineering Institute, SEI) і корпорацією Mitre під керівництвом Воттса Хамфрі (Watts Humphrey).

Методологія CMM розроблялася і розвивалася в США як засіб, що дозволяє вибирати кращих виробників ПЗ для виконання держзамовлень в першу чергу міністерства оборони. Для цього були розроблені критерії оцінки зрілості ключових процесів компанії і певний набір дій, необхідних для їх подальшого вдосконалення.

У підсумку методологія виявилася надзвичайно корисною для більшості компаній, які прагнуть якісно поліпшити існуючі процеси проектування, розробки, тестування програмних засобів і звести управління ними в зрозумілих і легко реалізованих алгоритмів і технологій, описаних в єдиному стандарті. Надалі ця модель переросла в *методологію підвищення якості процесів підприємства Capability Maturity Model Integration CMMI.*

Застосування CMMI дозволяє поставити розробку ПЗ на промислову основу, підвищити керованість ключовими процесами і виробничу культуру в цілому, гарантувати якісну роботу та виконання проектів в строк.

Основою для створення CMM стало базове положення про те, що фундаментальна проблема "кризи" процесу розробки якісного ПЗ полягає не у відсутності нових методів і засобів розробки, а в нездатності компанії організувати технологічні процеси і керувати ними.

Для оцінки ступеня готовності підприємства розробляти якісний програмний продукт CMM використовує ключове поняття ***зрілість організації (Maturity)***.

Незрілою вважається організація, в якій:

- відсутнє довгострокове і проектне планування;

- процес розробки програмного забезпечення та його ключові моменти не ідентифіковані, реалізація процесу залежить від поточних умов, конкретних менеджерів і виконавців;
- методи і процедури не стандартизовані і не задокументовані;
- результат не визначений реальними критеріями, які встановлюються запланованими показниками із застосуванням стандартних технологій і розроблених метрик;
- процес вироблення рішення відбувається стихійно, на грані мистецтва.

У цьому випадку велика ймовірність появи несподіваних проблем, перевищення бюджету або невиконання термінів здачі проекту. У такій компанії, як правило, менеджери і розробники не управляють процесами - вони змушені займатися поточними і спонтанними проблемами.

Основні ознаки зрілої організації:

- в компанії чітко визначені і документовані процедури управління вимогами, планування проектної діяльності, управління конфігурацією, створення і тестування програмних продуктів, відпрацьовані механізми управління проектами;
- ці процедури постійно уточнюються і удосконалюються;
- оцінки часу, складності та вартості робіт ґрунтуються на накопиченому досвіді, розроблених метриках і кількісних показниках, що робить їх досить точними;
- актуалізовані зовнішні і створені внутрішні стандарти на ключові процеси та процедури;
- існують обов'язкові для всіх правила оформлення методологічної і програмної документації та документації користувача;
- технології незначно змінюються від проекту до проекту, на основі стабільних і перевірених підходів та методик максимально використовуються напрацьовані в попередніх проектах організаційний і виробничий досвід, програмні модулі, бібліотеки програмних засобів;
- активно апробуються і впроваджуються нові технології, проводиться оцінка їх ефективності.

CMM визначає п'ять рівнів технологічної зрілості компанії (рис. 3.2), за якими замовники можуть оцінювати потенційних претендентів на підпис контракту, а розробники - удосконалювати процеси створення ПЗ.

Кожен з рівнів, крім першого, складається з декількох ключових областей процесу (Key Process Area), що містять цілі (Goal), зобов'язання щодо виконання (Commitment to Perform), можливість виконання (Ability to Perform), дії, що виконуються, (Activity Performed), їх вимір і аналіз (Measurement and Analysis) і перевірку впровадження (Verifying Implementation).

Таким чином, *CMM фактично є комплексом вимог до ключових параметрів ефективного стандартного процесу розробки ПЗ і засобом його постійного поліпшення*. Виконання цих вимог збільшує ймовірність досягнення підприємством поставлених цілей в галузі якості.

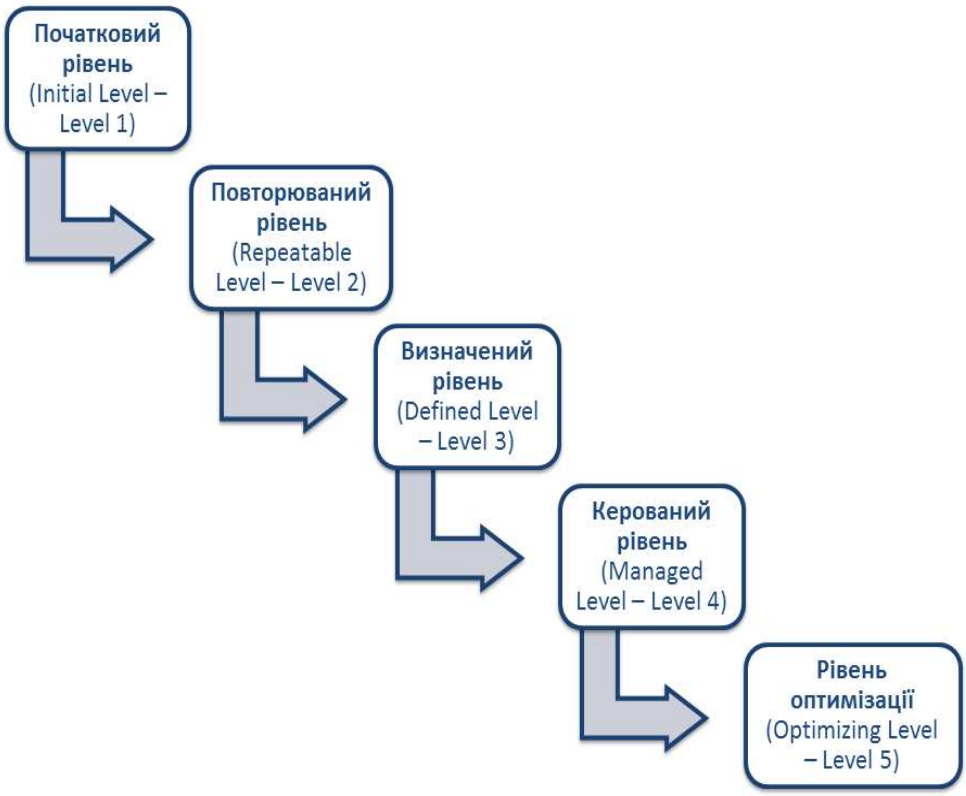

Рисунок 3.2 – Рівні зрілості компанії за стандартом CMM

Початковий рівень (Initial Level - Level 1)

На даному рівні компанія може отримати замовлення, розробити і передати замовнику програмний продукт. Стабільність розробок відсутня. Лише деякі процеси визначені, результат повністю залежить від зусиль окремих співробітників. Успіх одного проекту не гарантує успішності наступного. До цієї категорії можна віднести будь-яку компанію, яка хоч якось виконує взяті на себе зобов'язання. Ключові області процесу цього рівня не зафіксовано.

Повторюваний рівень (Repeatable Level - Level 2)

Цьому рівню відповідають підприємства, що володіють певними технологіями управління та розробки. Управління вимогами та планування в більшості випадків ґрунтуються на розробленій документованій політиці та накопиченому досвіді. Встановлено та введено в повсякденну практику базові показники для оцінки параметрів проекту. Менеджери відстежують виконання робіт і контролюють тимчасові і виробничі витрати.

У компанії розроблені деякі внутрішні стандарти і організовані спеціальні групи *перевірки якості QA* . Зміни версій кінцевого програмного продукту і створених проміжних програмних засобів відслідковуються в системі управління конфігурацією. Є необхідна дисципліна дотримання встановлених правил. Ефективні методики та процеси впроваджуються, що забезпечує можливість повторення успіху попередніх проектів в тій же прикладній галузі. Ключові галузі процесу розробки ПЗ цього рівня наведені на рисунку 3.3.

Визначний рівень (Defined Level - Level 3)

Рівень характеризується деталізованим методологічним підходом до управління (описані і закріплені в документованій політиці типові дії, що необхідні для багаторазового повторення: ролі і відповідальність учасників, стандартні процедури та операції, порядок дій, кількісні показники і метрики процесів, формати документів та ін.).

Для створення і підтримки методологій в актуальному стані в організації підготовлена і постійно функціонує спеціальна група (рис. 3.3)

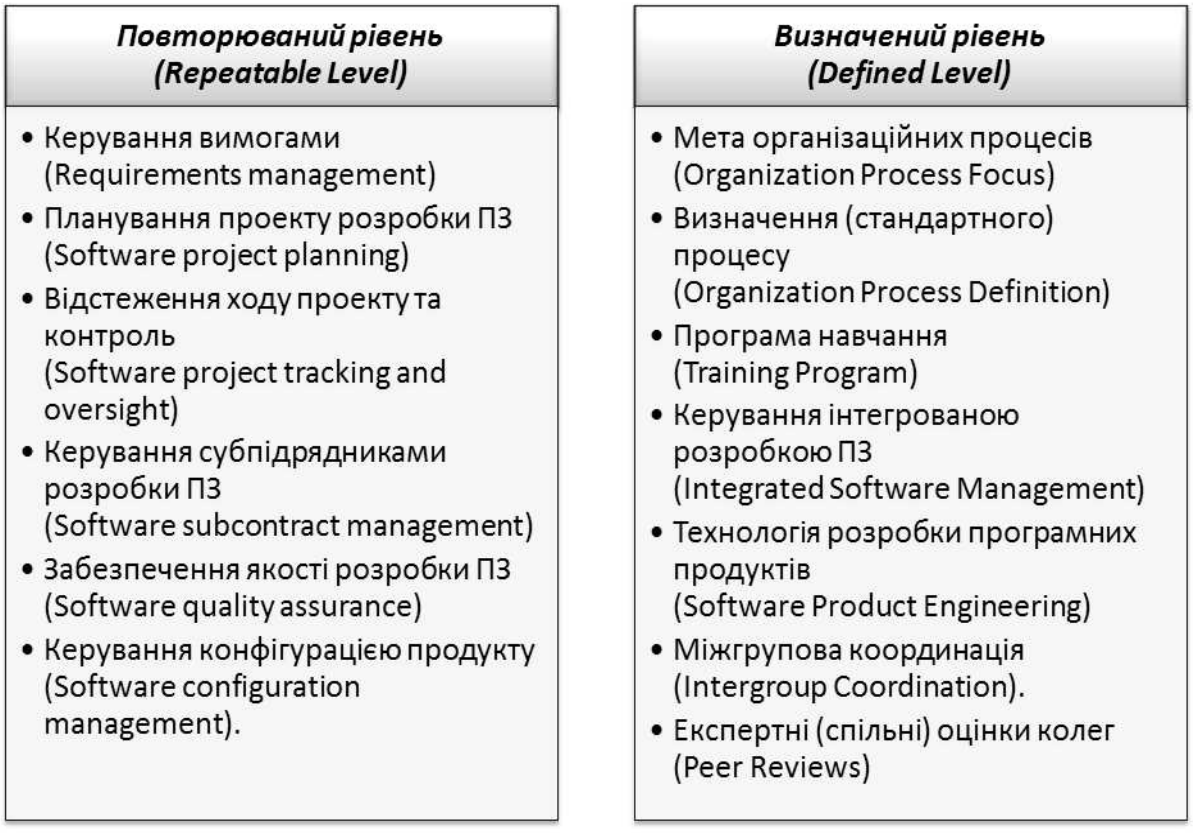

Рис. 3.3 – Ключові області повторюваного і визначного рівнів зрілості компанії

Компанія регулярно проводить тренінги для підвищення професійного рівня своїх співробітників. Починаючи з цього рівня, організація практично перестає залежати від особистісних якостей конкретних розробників і не має тенденції опускатися на більш низькі рівні. Ця незалежність обумовлена продуманим механізмом постановки завдань, планування заходів, виконання

операцій і контролю виконання. Управлінські та інженерні процеси документовані, стандартизовані і інтегровані в уніфіковану для всієї організації технологію створення ПЗ. Кожен проект використовує затверджену версію цієї технології, адаптовану до особливостей поточного проекту. Ключові галузі процесу розробки ПЗ цього рівня наведені на рисунку 3.3.

Керований рівень (Managed Level – Level 4)

Рівень, на якому розроблені і закріплені у відповідних нормативних документах кількісні показники якостей.

Більш високий рівень управління проектами досягається за рахунок зменшення відхилень різних показників проекту від запланованих. При цьому тенденції зміни продуктивності процесу можна відокремити від випадкових варіацій на підставі статистичної обробки результатів вимірювань в процесах. Ключові області процесу розробки ПЗ цього рівня наведені на рисунку 3.4.

Рівень оптимізації (Optimizing Level – Level 5)

Для цього рівня заходи щодо вдосконалення розраховані не тільки на існуючі процеси, а й на впровадження, використання нових технологій і оцінку їх ефективності. Основним завданням всієї організації на цьому рівні є постійне вдосконалення існуючих процесів, в ідеалі спрямоване на запобігання відомої помилки або дефекту і попередження можливих. Застосовується механізм повторного використання компонентів від проекту до проекту (шаблони звітів, формати вимог, процедури і стандартні операції, бібліотеки модулів програмних засобів). Ключові галузі процесу розробки ПЗ цього рівня наведені на рисунку 3.4.

Рисунок 3.4 – Ключові галузі керованого рівня зрілості і рівня оптимізації компанії

CMM визначає такий мінімальний набір вимог: *реалізувати 18 ключових галузей процесу розробки ПЗ, що містять 52 мети, 28 зобов'язань*

компанії, 70 можливостей виконання (гарантій компанії) і 150 ключових практик.

В результаті аудиту та атестації компанії присвоюється певний рівень, за результатами наступних аудитів рівень може підвищуватися або знижуватися. Кожен наступний рівень в обов'язковому порядку включає в себе всі ключові характеристики попередніх. У зв'язку з цим сертифікація компанії в одному з рівнів передбачає безумовне виконання всіх вимог більш низьких рівнів.

До переваг моделі CMM відноситься те, що вона орієнтована на організації, які займаються розробкою програмного забезпечення. У даній моделі вдалося більш детально визначити вимоги, які специфічні для процесів, що пов'язані з розробкою ПЗ. З цієї причини в CMM наведені не тільки вимоги до процесів організації, а й приклади реалізації таких вимог.

Основний недолік CMM полягає в тому, що модель не авторизована як стандарт ні міжнародними, ні національними органами стандартизації. Втім, CMM давно стала промисловим стандартом. До недоліків моделі також необхідно віднести великі зовнішні накладні витрати на приведення процесів компанії у відповідність до моделі CMM, ніж при використанні моделей Міжнародного стандарту ISO 9000.

3.4 Стандарт SPICE

Стандарт SPICE успадкував багато рис більш ранніх стандартів, у тому числі і вже згадуваних ISO 9001 і CMM. Найбільше SPICE нагадує CMM. Точно так само, як і в CMM, основним завданням організації є постійне поліпшення процесу розробки можливостей (в SPICE визначено 6 різних рівнів), але ці рівні застосовуються не тільки до організації в цілому, а й до окремо взятих процесів.

В основі стандарту лежить *оцінка процесів*. Ця оцінка виконується шляхом порівняння процесу розробки програмного забезпечення, що існує в даній організації, з описаною в стандарті моделлю. Аналіз результатів, що отримані на цьому етапі, допомагає визначити сильні і слабкі сторони процесу, а також внутрішні ризики, що властиві даному процесу. Це допомагає оцінити ефективність процесів, визначити причини погіршення якості та пов'язані з цим витрати в часі або вартості. Потім виконується визначення можливостей процесу, тобто можливостей його поліпшення. В результаті в організації може з'явитися розуміння необхідності поліпшення того чи іншого процесу. До цього моменту цілі вдосконалення процесу вже чітко сформульовані і залишається тільки технічна реалізація поставлених завдань. Після цього весь цикл робіт починається спочатку.

Безумовно, вдосконалення процесів життєвого циклу програмного забезпечення абсолютно необхідне. Проте слід мати на увазі, що побудова «більш зрілого» процесу розробки не обов'язково забезпечує створення більш

якісного програмного забезпечення. Це хоча і пов'язані, але абсолютно різні процеси.

Використання формальних моделей і методів дозволяє створювати зрозумілі, несуперечливі специфікації на програмне забезпечення, що розробляється. Звичайно, впровадження таких методів має сенс, хоча воно дуже дороге і трудомістке, а можливості його застосування досить обмежені. Основна ж проблема - проблема складності програмного забезпечення з удосконаленням процесів розробки поки не вирішена.

Створення програмного забезпечення як і раніше висуває підвищені вимоги до кваліфікації тих, хто цим займається: проектувальникам програмного забезпечення і безпосередньо програмістам.

Питання для самоконтролю

1. Які види стандартів вам відомі? Наведіть приклади.
2. Що відповідно до стандарту CMM повинна запровадити компанія-розробник програмних систем, щоб якісно виконувати замовлення?
3. Які міжнародні стандарти ISO регулюють розроблення інформаційних систем?
4. Які міжнародні стандарти якості використовуються під час розроблення інформаційних систем?
6. Які групи процесів у ході розроблення програмного забезпечення виділяють стандарти ISO?
7. Які стандарти організації IEEE регламентують розроблення програмних систем?
8. Які області охоплює керівництво SWEBOK?
9. Які державні стандарти України регламентують розроблення інформаційних систем та програмного забезпечення?
10. Які дії, передбачені стандартом ISO 12207, виконуються в ході розроблення програмних продуктів?
11. Які рівні зрілості компанії виділяє стандарт CMM? Чим вони характеризуються?
12. На якому рівні зрілості компанія може контролювати якість створюваного програмного забезпечення?

4 СУЧАСНІ МЕТОДОЛОГІЇ РОЗРОБЛЕННЯ ПРОГРАМНИХ СИСТЕМ

4.1 CASE-засоби та нотації моделювання програмних систем

На становлення технологій виробничого програмування найбільш помітний вплив мали методологія структурного програмування та об'єктно-орієнтоване програмування. Перша дозволила усвідомити обмеженість здібностей людини, необхідних для створення великих програм. Об'єктно-орієнтоване програмування дало поштовх до розроблення методів декомпозиції, пристосованих для подолання складності проектів, та привело до модернізації основних принципів проектування програм. Незважаючи на ці та інші впливи, стадія комплексної автоматизації технологій програмування стала можливою лише при відповідному рівні розвитку техніки. Важливою обставиною, що дозволила перейти до комплексної автоматизації, стало усвідомлення того, що не можна говорити про промислове програмування без підтримки технологічних функцій на усіх етапах життя програм. Приблизно на початку 90-х рр. XX ст. з'явився термін - CASE-технологія (Computer Aid Software Engineering - комп'ютерна підтримка розроблення програм), яким стали позначати використання систем, що містять комплексні автоматизовані засоби підтримки розроблення і супроводу ПЗ. Найбільш вдалим виявилося використання CASE-систем у тих спеціальних областях, в яких вже були успіхи і досвід технологічної практичної роботи, а також у тих випадках, коли ця область вже була забезпечена надійною теоретичною базою. На рисунку 4.1 наведено етапи розвитку CASE-засобів за областями застосування.

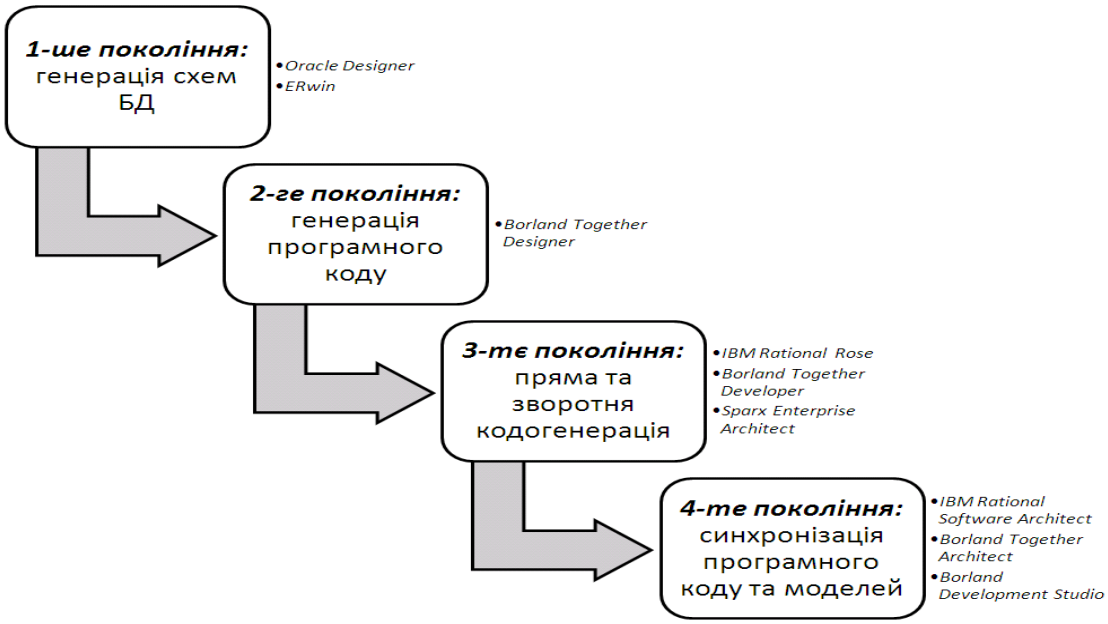

Рисунок 4.1 – Етапи розвитку CASE-засобів за областями застосування

Першими з'явилися CASE-системи розроблення баз даних у розвинених реляційних СУБД (наприклад, Oracle Designer в системі Oracle). Наступними стали засоби генерації програмного коду та налагодження програм. Потреба підтримки не лише процесу програмування, а й етапу аналіз та проектування програмних продуктів привело до створення CASE-систем, які зв’язували моделі ПЗ із програмним кодом – спочатку лише для прямої та зворотної кодогенерації, а потім із підтримкою синхронного зв'язку коду із моделями аналізу та проектування.

Сьогодні універсальні CASE-системи будуються у рамках застосування розвинених, але все ж спеціальних методологій. Безперечний прогрес у даній сфері досягнутий для проектування, орієнтованого на моделювання на етапах аналізу і конструювання (CASE-системи 4-го покоління).

Складність процесу розроблення інформаційних систем викликала появу візуальних засобів моделювання (рис.4.2). У рамках об'єктно-орієнтованого підходу Object management group (OMG) розроблена спеціальна уніфікована мова моделювання UML (Unified Modeling Language), що розглядається як основа проектування в методології ітеративного нарощування можливостей об'єктно-орієнтованих програмних систем. Широкого вжитку для моделювання процесів різних галузей екномічної діяльності набула родина візуальних мов IDEF (Integration Definition), що є стандартом моделювання Міністерства оборони США. Паралельно із нотаціями UML та IDEF розвиваються методологія та нотація для професійного моделювання бізнес-процесів ARIS (ARchitecture of Integrated Information Systems.

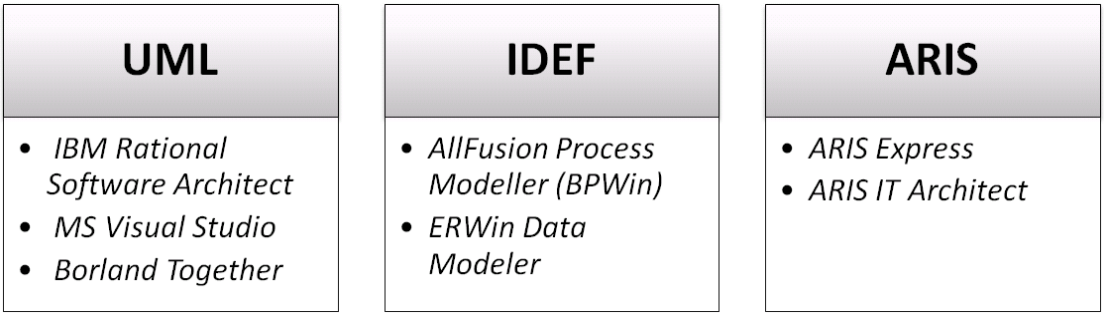

Рисунок 4.2 – Нотації візуального моделювання систем

Зміна життєвого циклу програмного забезпечення при використанні CASE-технологій

CASE-технології являють собою сукупність методологій аналізу, проектування, розробки та супроводу складних програмних систем, які засновані як на структурному, так і на об'єктному підходах, що підтримуються комплексом взаємопов'язаних засобів автоматизації. В основі будь якої CASE-технології лежить парадигма методологія / метод / нотація / засіб.

Методологія будується на базі деякого підходу і визначає кроки роботи, їх послідовність, а також правила розподілу та призначення методів.

Метод визначає спосіб досягнення тієї чи іншої мети - виконання кроку роботи.

Нотацією називають систему позначень, використовуваних для опису деякого класу моделей. Нотації бувають графічні (надання моделей у вигляді графів, діаграм, таблиць, схем і т. п.) і текстові (описи моделей на формальних і природних мовах). В CASE-технологіях нотації використовують для опису структури проектованої системи, елементів даних, етапів обробки і т. п. графічного проекту, організації проекту у вигляді ієрархії рівнів абстракції, а також перевірки відповідності компонентів різних рівнів.

Розрізняють:

CASE-засоби аналізу вимог, проектування специфікацій і структури;

редагування інтерфейсів (перше покоління CASE-I);

CASE-засоби генерації вихідних текстів та реалізації інтегрованого оточення підтримки повного життєвого циклу розробки програмного забезпечення (друге покоління CASE-II).

CASE-I в основному включають кошти для підтримки графічних моделей, проектування специфікацій, екранних редакторів і словників даних.

CASE-II відрізняється істотно великими можливостями, забезпечуючи: контроль, аналіз і зв'язування системної інформації та інформації з управління процесом проектування, побудова прототипів і моделей системи, тестування, верифікацію і аналіз згенерованих програм.

Автоматизуючи трудомісткі операції, сучасні CASE-засоби суттєво підвищують продуктивність праці програмістів і покращують якість створюваного програмного забезпечення.

Вони:

забезпечують автоматизований контроль сумісності специфікацій проекту;

зменшують час створення прототипу системи;

прискорюють процес проектування і розробки;

автоматизують формування проектної документації для всіх етапів життєвого циклу відповідно до сучасних стандартів;

частково генерують коди програм для різних платформ розробки;

підтримують технології повторного використання компонентів системи;

забезпечують можливість відновлення проектної документації за наявними вихідними кодами.

Поява CASE-технологій змінила всі етапи життєвого циклу програмного забезпечення, при цьому найбільші зміни стосуються аналізу і проектування, які припускають суворий і наочний опис розроблюваного програмного забезпечення. Використання CASE-засобів дозволяє істотно знизити трудовитрати на розробку складного програмного забезпечення в основному за рахунок автоматизації процесів документування та контролю. Проте слід мати на увазі, що сучасні CASE-засоби дорогі, а їх використання вимагає *більш високої кваліфікації розробників*. Отже, їх має сенс використовувати в складних проектах, причому, чим складніше розроблюване програмне забезпечення, тим більше виграш від використання CASE-технологій.

На сьогоднішній день практично все промислово вироблене складне програмне забезпечення розробляється з використанням CASE-засобів.

4.2 Жорсткі та гнучкі стратегії в методологіях програмування

Визначаючи стратегії послідовного і ітеративного розвитку проектів. Контроль діяльності проекту – завдання, що вимагає спеціальних організаційних підходів.

Загальною метою є перетворення створення програмного продукту в упорядкований процес, у рамках якого розвиток проекту можна зробити більш прогнозованим і ефективним. Для цього зазвичай створюється детальний опис процесу розробки системи, особливе місце в якому займає планування. Інакше кажучи, ***створюється метод, за допомогою якого передбачається конструювати систему.*** Вдалі методи узагальнюються, в результаті досвід їх застосування перетворюється в методику, або, як зараз прийнято говорити, ***методологію.***

Поряд з усім корисним ці рішення оголошуються обов'язковими, стандартними, їх змушені застосовувати і тоді, коли вихідні передумови втрачені. Щоб слідувати таким методологіям, доводиться виконувати безліч різних приписів, що уповільнює темп основних програмістських робіт. Тому їх називають ***важкими, жорсткими або монументальними.***

Жорсткі методології привабливі для замовників програмних проектів, які завжди можуть перевірити, чи дійсно процес розробки впорядкований і результати відповідають планам. Природно пов'язувати цю можливість з організацією, яка отримує замовлення і забезпечує його менеджмент, оскільки крім найкращих побажань (найчастіше вони-то і порушуються!) в організації можливий адміністративний контроль.

Пропозиції CMM для визначення зрілості організації спираються на те, які процедури управління програмними проектами, відстеження їх розвитку та інші менеджерські методи прийняті як фірмовий стандарт. Методології програмування, що побудовані на базі цих пропозицій, часто розглядають як еталон жорсткості.

На противагу жорстким методологіям останнім часом сформувався компромісний підхід, методології якого об'єднані загальним терміном **agile development.** На російську мову його перекладають як ***швидкий розвиток***, ***гнучкі методології*** і навіть ***"спритні технології".*** У новому підході намагаються надати можливість полегшеної організованої роботи колективів програмістів, коли перевантаженість стандартизованого процесу перешкоджає ефективності.

Вони претендують на те, що їх застосування можливе навіть коли не вдається досить точно представити проект при його зародженні, тобто в досить типовій ситуації невизначеності реальної користувальницької потреби.

У цих випадках пропонується залучення замовника до формування завдань та коригуванні припущень протягом всього розвитку проекту. З його допомогою намагаються виявляти найбільш точно і без зайвих бюрократичних процедур актуальні потреби користувачів, що склалися на поточний момент. В результаті з'являється можливість вказати саме ті вимоги, реалізація яких необхідна і допускає максимально короткі терміни випуску релізу.

Всі методології швидкого розвитку орієнтуються на стратегію ітеративного нарощування можливостей системи. Всі вони надають розробникам значно більшу свободу, ніж, приміром, вимоги стандартів CMM. Але не слід думати, що цей підхід повністю скасовує жорсткі методології – ***необхідний баланс між свободою швидких методологій і дисципліною.***

Для жорстких методологій характерно прагнення забезпечити розробників рецептами, що не вимагають обговорень: потрібно виконувати відомі рецепти, дотримуватися регламенту, щоб відповідні операційні маршрути приводили до допустимих траєкторій. Іншими словами, ***жорсткі методології підтримують*** переважно такі діяльності, які можна назвати ***імперативними***.

На відміну від традиційних підходів *швидкі методології орієнтуються на те,* що діяльність з виробництва програмного забезпечення по суті своїй є переважно ***реативною,*** тобто такою, в якій від розробників потрібно не тільки розпізнавання ситуацій і застосування в них відомих методів, але і конструювання нових методів дії. А це інший, більш високий рівень знань і умінь.

В даний час широке застосування отримують так звані промислові технології створення програмного продукту. Ці технології були розроблені фірмами, які накопичили великий досвід створення ПЗ. Технології представлені описами принципів, методів, процесів і операцій, що використовуються. Такі технології, як правило, підтримуються набором CASE-засобів, охоплюють всі етапи життєвого циклу продукту і успішно

застосовуються для вирішення практичних завдань. Найбільш помітними є такі методології розроблення ПЗ:

Rational Unified Process (RUP) – розробка фірми Rational, довгий час успішно займалася створенням CASE-засобів, що застосовуються на різних етапах життєвого циклу продукту від аналізу до тестування і документування.

Rational Unified Process (RUP), що пропонує ітеративну модель розроблення, що включає чотири фази: початок, дослідження, побудову та впровадження. Проходження через чотири основні фази називається циклом розроблення, кожен цикл завершується генерацією працездатної версії системи. У ході супроводу продукт продовжує розвиватися і знову проходить ті самі фази. Розроблення ПЗ на базі RUP базується на створенні та супроводі моделей на базі UML.

Microsoft Solution Framework (MSF) – розробка фірми Microsoft, призначена для вирішення широкого кола завдань. Технологія масштабована, тобто налагоджується на вирішення завдань будь-якої складності колективом будь-якої чисельності.

Microsoft Solution Framework (MSF) – методологія, що розроблялася для підвищення керованості процесів розроблення окремого підприємства (Microsoft), а потім стала застосовуватися розробниками, які використовують продукти Microsoft. Методологія, подібна до RUP, також включає чотири фази: аналіз, проектування, розроблення, стабілізацію, є ітераційною, припускає використання об'єктно-орієнтованого моделювання. MSF порівняно з RUP здебільшого орієнтована на розроблення бізнес-додатків.

Extreme Programming (XP) – активно розвивається останнім часом технологія, що призначена для вирішення відносно невеликих завдань, відносно невеликими колективами професійних розробників в умовах жорстко обмеженого часу.

eXtreme Programming (XP) – методологія, що приділяє основну увагу ефективній комунікації між замовником і виконавцем упродовж усього проекту з розроблення ІС для відслідковування зміни вимог. В основу підходу покладена командна робота та постійне тестування прототипів.

Гнучке розроблення ПЗ (Agile) – клас методологій розроблення програмного забезпечення на основі ітеративної моделі ЖЦ, в якій вимоги та рішення еволюціонують через співпрацю між самоорганізовуваними командами.

Кожна з цих технологій має свої особливості організації моделі життєвого циклу створення продукту. Розглянемо особливості моделей життєвого циклу трьох найбільш відомих промислових технологій:

4.3 Методологія Rational Unified Process (RUP)

Уніфікований процес розроблення ПЗ Rational Unified Process (RUP) виник як відповідь на потребу розробників у процесі, що об'єднує безліч аспектів розроблення програм і відповідає таким вимогам:

- забезпечує керівництво діяльністю команди;
- керує завданнями окремого розробника і команди в цілому;
- показує, які артефакти необхідно розробити;
- надає критерії відстеження та вимірювання продуктів і функціонування проекту.

Уніфікований процес розроблення ПЗ передбачає розподіл ЖЦ програмного забезпечення на фази (рис. 4.3).

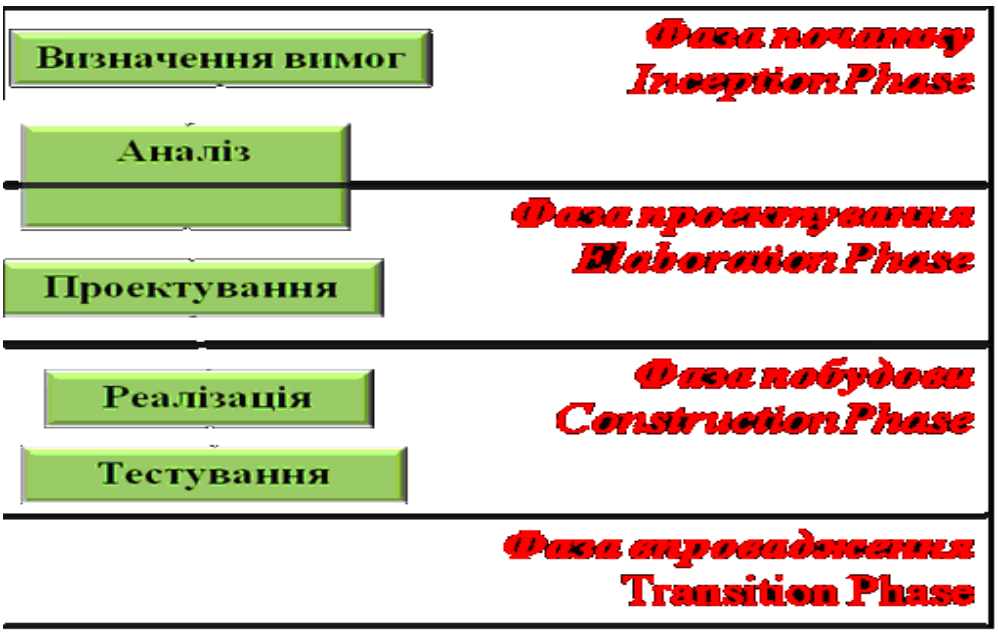

Рисунок 4.3 – Робочі процеси ЖЦ в уніфікованому процесі розроблення ПЗ

Специфіка уніфікованого процесу полягає у трьох словосполученнях – керований варіантами використання, архітектурно-орієнтований, ітеративний та інкрементний.

Модель життєвого циклу RUP є досить складною, детально відпрацьованою ітеративно-інкрементною моделлю з елементами каскадної моделі. У моделі RUP виділяються 4 основні фази, 9 видів діяльності (процесів). Крім того, в моделі описується ряд *практик*, які слід застосовувати або керуватися для успішного виконання проекту. RUP орієнтований на поетапне моделювання продукту, що створюється, за допомогою мови UML.

*Основними фазами **RUP** є:*

• **Фаза початку проекту (Inception).** Визначаються основні цілі проекту, бюджет проекту, основні засоби його виконання - технології, інструменти, ключовий персонал, складаються попередні плани проекту. Основна мета цієї фази - досягти компромісу між усіма зацікавленими особами щодо завдань проекту.

• **Фаза опрацювання (Elaboration).** Основна мета цієї фази - на базі основних, найбільш істотних вимог, розробити стабільну базову архітектуру продукту, яка дозволяє вирішувати поставлені перед системою завдання і надалі використовується як основа розробки системи.

• **Фаза побудови (Construction).** Основна мета цієї фази - детальне прояснення вимог і розробка системи, що задовольняє їм, на основі спроектованої раніше архітектури.

• **Фаза передачі (Transition).** Мета фази - зробити систему повністю доступною кінцевим користувачам. Тут відбувається остаточне розгортання системи в її робочому середовищі, підгонка дрібних деталей під потреби користувачів. В рамках кожної фази можливе проведення декількох ітерацій, кількість яких визначається складністю виконуваного проекту.

Діяльності (основні процеси) RUP діляться на п'ять робочих і чотири підтримуючі.

До робочих діяльностей відносяться:

• ***Моделювання предметної області*** (бізнес-моделювання, Business Modeling). Цілі цієї діяльності - зрозуміти бізнес-контекст, в якому повинна буде працювати система (і переконатися, що всі зацікавлені особи розуміють його однаково), зрозуміти можливі проблеми, оцінити можливі їх рішення та їх наслідки для бізнесу організації, в якій працюватиме система.

• ***Визначення вимог (Requirements)***. Цілі - зрозуміти, що повинна робити система, визначити межі системи і основу для планування проекту та оцінок ресурсозатрат в ньому.

• ***Аналіз та проектування (Analysis and Design).*** Вироблення архітектури системи на основі ключових вимог, створення проектної моделі, представленої у вигляді діаграм UML, що описують продукт з різних точок зору.

• ***Реалізація (Implementation).*** Розробка вихідного коду, компонент системи, тестування та інтегрування компонент.

• ***Тестування (Test).*** Загальна оцінка дефектів продукту, його якість в цілому; оцінка ступеня відповідності вихідним вимогам.

Підтримуючими діяльностями є:

• ***Розгортання (Deployment).*** Цілі – розгорнути систему в її робочому оточенні і оцінити її працездатність.

• ***Управління конфігураціями і змінами (Configuration and Change Management)***. Визначення елементів, що підлягають зберіганню та правил побудови з них узгоджених конфігурацій, підтримку цілісності поточного стану системи, перевірка узгодженості внесених змін.

• ***Управління проектом (Project Management).*** Включає планування, управління персоналом, забезпечення зв'язків з іншими зацікавленими особами, управління ризиками, відстеження поточного стану проекту.

• ***Управління середовищем проекту (Environment)***. Налаштування процесу під конкретний проект, вибір і зміна технологій та інструментів, що використовуються в проекті.

4.4 Методологія Microsoft Solution Framework (MSF)

Одна з особливостей технології MSF полягає в тому, що вона орієнтована не просто на створення програмного продукту, що задовольняє перерахованим вимогам, а на пошук рішення проблем, що стоять перед замовником. Різниця полягає в тому, що перелічені замовником вимоги є проявами деяких більш глибоких проблем і неточність, неповнота, зміна вимог у процесі розробки - наслідок недостатнього розуміння проблем. Тому, в технології MSF велика увага приділяється аналізу проблем замовника і розробці варіантів системи для пошуку рішення цих проблем.

Модель життєвого циклу MSF є деяким гібридом каскадної і спіральної моделей, поєднуючи простоту управління каскадної моделі з гнучкістю спіральної.

Модель життєвого циклу MSF орієнтована на "віхи" (milestones) – ключові точки проекту, що характеризують досягнення якого-небудь істотного результату.

Цей результат може бути оцінений і проаналізований, що має на увазі відповідь на питання: "А чи досягли ми цілей, поставлених на цьому кроці?».

В основу MSF покладено вісім базових принципів (foundational principles):

- ***Сприяння відкритої комунікації (Foster open communications).*** Модель процесу MSF запроваджує вільний інформаційний потік сред членів команди та зацікавлених сторін (key stakeholders) для забезпечення однакового розуміння завдань. Документування ходу

проекту та доступ до цих даних членів команди, зацікавлених сторін та замовників.

- ***Робота у напрямку спільного бачення проекту (Work toward a shared vision)***. Модель процесу MSF запроваджує фазу формування концепції (Envisioning Phase) та окремий ключовий момент затвердження бачення (milestone Vision/Scope Approved) для формування спільного бачення проекту. Бачення включає детальне розуміння цілей та завдань для досягнення рішення поставленої проблеми. Спільне бачення виявляє припущення команди та замовників, потрібні для отримання рішення.
- ***Надання прав і можливостей членам команди (Empower team members)***. Розширення прав і можливостей членів команди для прийняття ними відповідальності за виконану роботу. Таке збільшення відповідальності може бути затверджене у графіках, де фіксується дата закінчення робіт, що також може бути засобом виявлення можливих затримок проекту.
- ***Визначення індивідуальної та спільної відповідальності (Establish clear accountability and shared responsibility)***. Модель командної групи MSF ґрунтується на принципі важливості роботи кожного для отримання якісного рішення проблеми. Усі члени групи розділяють відповідальність за проект.
- ***Зосередження на бізнес-цілях (Focus on delivering business value).*** Рішення повинне приносити користь організації у вигляді додавання вартості бізнесу. Це додавання досягається тільки після повного розгортання рішення у виробничому середовищі.
- ***Бути гнучкими, очікувати на зміни (Stay agile, expect change).*** MSF припускає, що у виробничому середовищі на рішення постійно впливають зміни. Команда повинна бути обізнаною та готовою до керування змінами вимог.
- ***Інвестування у якість (Invest in quality)***. У MSF кожен член команди відповідальний за якість вирішення завдання.Для підтримки якості протягом проекту формується команда тестувальників. Це гарантує, що рішення відповідає рівню якості, визначеному зацікавленими сторонами.
- ***Навчання за досвідом (Learn from all experiences).*** MSF вимагає використання досвіду, отриманого у попередніх проектах. Це дозволяє знайти найкращі методики розроблення.

MSF складається із двох моделей та трьох дисциплін:

- моделі командної групи;
- моделі процесу;
- дисципліни управління проектами;
- дисципліни управління ризиками;
- дисципліни управління підготовкою.

Модель командної групи (MSF Team Model) описує, як організувати колективи і якими принципами керуватися для досягнення максимального успіху в розробленні програм. Різні колективи можуть по-своєму застосовувати на практиці різні елементи цієї моделі – все залежить від масштабу проекту, розміру колективу і кваліфікації його учасників та моделі процесу розроблення.

Формування колективу є складним завданням, що повинне виконуватися психологами та враховувати такі *основні положення*:

- не повинне бути команди з одних лідерів;
- не повинне бути команди з одних виконавців;
- у випадку невдачі команда розформовується;
- система штрафів (якщо проект провалюється – карають усіх).

Модель командної групи визначає тільки ролі, кожна з яких може виконуватися кількома людьми (рис. 4.4). Цікаво, що в моделі командної групи не передбачено єдиноначальності – усі ролі важливі, усі ролі рівноправні, тому MSF називають моделлю рівних (team of peers).

Рисунок 4.4 – Модель командної групи (MSF Team Model)

Program management – керування програмою. Виконавець цієї ролі відповідає за організацію (але не керує): здійснює ведення графіка робіт, ранкові 15-хвилинні наради, забезпечує відповідність стандартам і специфікаціям, фіксацію порушень, написання технічної документації.

Product management – керування продуктом. Виконавці цієї ролі відповідають за спілкування із замовником, написання специфікації, роз'яснення завдань розроблювачам.

Development – найбільш традиційна роль – розроблення і початкове тестування продукту.

User expirience – підвищення ефективності роботи користувачів, написання користувальницької документації.

Release management – розгортання релізу продукту, супровід і його технічна підтримка.

Test – визначення відповідності показників якості релізу встановленим значенням. Виявлення й усунення недоробок, виправлення помилок, інші функції QA.

У MSF затверджується, що таку модель можна масштабувати, розбиваючи систему за функціями.

Модель процесу (MSF Process Model) визначає, коли і які роботи повинні бути виконані.

Основні принципи і практичні прийоми, на яких ґрунтується модель:

- ітеративний підхід (послідовний випуск версій);
- підготовка чіткої документації;
- урахування невизначеності майбутнього;
- облік компромісів;
- керування ризиками;
- підтримка відповідального відношення колективу до строків випуску продукту;
- розбивка великих проектів на більш дрібні керовані частини;
- щоденне складання проекту;
- постійний аналіз ходу робіт.

Process model *має три основні особливості****:***

- розбивка всього процесу на фази;
- використання опорних точок (milestones);
- ітеративність.

Увесь процес розбивається на п’ять взаємозалежних фаз (рис. 4.5). Перш ніж переходити до наступної фази, на попередній повинні бути отримані певні результати (досягнуті головні опорні точки). Фактично процес ітеративний і відповідає спіральній моделі ЖЦ ПЗ. У моделі передбачається наявність основних віх (завершення головних фаз моделі) і проміжних, що відображають внутрішні етапи головних фаз.

Рисунок 4.5 – Модель процесу (MSF Process Model)

- ***Створення загальної картини додатку (Envisioning).*** На цьому етапі вирішуються такі основні завдання:
  - оцінка існуючої ситуації;
  - визначення складу команди;
  - структури проекту, бізнес-цілей, вимог і профілів користувачів;
  - розробка концепції рішення і оцінка ризику.

Встановлюються дві проміжні віхи: "Організовано кістяк команди" і "Створена загальна картина рішення".

- ***Планування (Panning).*** Включає планування і проектування продукту. На основі аналізу вимог розробляється проект і основні архітектурні рішення, функціональні специфікації системи, плани і календарні графіки, середовища розробки, тестування та пілотної експлуатації.

Етап складається з трьох стадій: концептуальне, логічне і фізичне проектування.

На стадії концептуального проектування задача розглядається з точки зору користувача і бізнес-вимог і закінчується визначенням набору сценаріїв використання системи.

При логічному проектуванні задача розглядається з точки зору проектної команди, рішення представляється у вигляді набору сервісів.

І вже на стадії фізичного проектування задача розглядається з точки зору програмістів, уточнюються технології і інтерфейси, що використовуються.

- ***Розробка (Developing).*** Створюється варіант вирішення проблеми, у вигляді коду і документації чергового прототипу, включаючи специфікації і сценарії тестування. Основна віха етапу – "Остаточне затвердження області дії проекту".

Продукт готовий до зовнішнього тестування і стабілізації. Крім того, замовники, користувачі, співробітники служби підтримки та супроводу, а також ключові учасники проекту можуть попередньо оцінити продукт і вказати всі недоліки, які потрібно усунути до його поставки.

- ***Стабілізація (Stabilizing).*** Підготовка до випуску остаточної версії продукту, доведення його до заданого рівня якості. Тут виконується комплекс робіт з тестування (виявлення та усунення дефектів), перевіряється сценарій розгортання продукту. Коли рішення стає досить стійким, проводиться його пілотна експлуатація в тестовому середовищі із залученням користувачів і застосуванням реальних сценаріїв роботи.
- ***Розгортання (Deploying).*** Виконується установка рішення і необхідних компонентів оточення, проводиться його стабілізація в промислових умовах і передача проекту групі супроводу. Крім того, аналізується проект в цілому на предмет рівня задоволеності замовника.

Важливу роль відіграють ***опорні точки (milestones),*** у яких аналізується стан робіт і виробляється їхня синхронізація. У цих точках додаток або його специфікації не заморожуються. Опорні точки дозволяють проаналізувати стан проекту і внести необхідні корективи, наприклад, переналагоджуватися під вимоги, що змінилися, замовника або відреагувати на ризики, можливі в ході подальшої роботи. Для кожної опорної точки визначається, які результати повинні бути отримані до цього моменту.

Кожна фаза процесу розроблення завершується головною опорною точкою (major milestone) – моментом, коли всі члени колективу синхронізують отримані результати. Призначення таких точок у тому, що вони дозволяють оцінити життєздатність проекту. Їхні результати видимі не тільки колективу розроблювачів, але й замовникові. Після аналізу результатів колектив розробників і замовник спільно вирішують, чи можна переходити на наступну фазу. Таким чином, головні опорні точки - це критерії переходу з однієї фази проекту на іншу.

Усередині кожної фази визначаються проміжні опорні точки (interium milestones). Вони, як і головні, слугують для аналізу й синхронізації досягнутого, а не для заморожування проекту. Але на відміну від головних опорних точок проміжні видні тільки членам колективу розробників.

Ітеративність процесу MSF полягає в його багаторазовому повторенні упродовж усього циклу розроблення та існування продукту. На кожній успішній ітерації у продукт включаються тільки ті нові інструменти та функції, які задовольняють постійно змінювані вимоги бізнесу.

4.5 Методологія eXtreme Programming (XP)

Екстремальне програмування є прикладом так званого методу «живої розробки» *(Agile Development Method).*

Екстремальне програмування (eXtreme Programming, XP) – спрощена методологія організації виробництва для невеликих і середніх за розміром команд розробників, які займаються розробленням програмного продукту в умовах незрозумілих або швидко змінних вимог.

Програмування відповідно до методик XP доводить використання загальноприйнятих принципів програмування до екстремальних рівнів:

- перегляд коду виконується постійно (з урахуванням того, що програмування ведеться парами);
- кожен учасник проекту тестує код програми постійно (тестування модулів), навіть замовники проводять функціональне тестування;
- проектування є складовою частиною повсякденної роботи кожного розробника (перероблення коду);
- розроблення виконується з урахуванням вимоги збереження в системі найбільш простого дизайну, що забезпечує поточний необхідний рівень функціональності (простіші речі надійніші в роботі);
- увага до архітектури системи на кожному етапі проекту;
- інтеграційне тестування виконується після кожної найменшої зміни у системі (триваюча інтеграція);
- ітерації невеликі – тривають години, кілька днів (постійне планування).

Для формування стилю розроблення, що дозволить досягти потрібної якості рішення, ***в XP пронується керуватись чотирма цінностями***:

- *комунікацією (communication)* – дисципліна XP, спрямована на забезпечення безперервної комунікації усіх учасників проекту. Під час тестування, програмування в парах та попереднього оцінювання робіт замовники, розробники та менеджери змушені постійно спілкуватись;
- *простотою (simplicity)* – неможливо заздалегідь точно визначити, як буде розвиватися проект, тому вирішувати необхідно лише завдання, що виникають сьогодні, у найбільш простий спосіб;
- *зворотним зв'язком (feedback)* – спосіб отримати точні та конкретні дані про стан проекту. Постійне виконання тестів для перевірки усіх змін у

системі забезпечує програміста звороним зв’язком про якість роботи. Кожен новий опис замовником вимог до системи одразу ж оцінюється розробниками і забезпечує замовника зворотним зв’язком із інформацію про якість опису. Менеджер, який контролює строки виконання робіт, забезпечує усіх учісників проекту інформацією про виконання планових термнів розроблення. В ХР найбільш важливі функції системи реалізуються в першу чергу у реально працюючий продукт, тому замовник досить швидко може оцінити хід виконання та якість розроблення, а також відповідність замовленню;

- *хоробрість (courage)* – для того, щоб розроблення не втратила своєї актуальності, в ХР рішення повинні прийматися з максимальною швидкістю, для чого потрібна певна сміливість.

Для визначення методів вирішення проблем розроблення ПЗ на основі обраних цінностей потрібно керуватися такими ***фундаментальними принципами оцінки методів вирішення поставленого завдання:***

- *швидкий зворотний зв'язок* – інформація про стан системи повинна якомога швидше надходити до зацікавлених сторін проекту. У рамках дисципліни ХР ці дані повинні із максимальною швидкістю надходити, інтерпретуватися та на основі їх аналізу швидко виконуватися модифікації системи;
- *прийнятна простота* – кожна проблема повинна вирішуватись у найбільш простий спосіб. У рамках ХР значні зусилля (тестування, переробка коду, комунікації) покладаються для вирішення завдань сьогодення таким чином, щоб завтра внесення змін не потребувало великих зусиль;
- *поступова зміна* – кардинальні зміни часто приводять до провалу проекту, щоб заплановані модифікації мали успіх, їх потрібно реалізовувати як серію невеликих змін, після кожної з яких перевіряється працездатність системи;
- *прийнятна зміна* – найвигідніша стратегія – та, що дозволяє вирішити найбільш важливу проблему і залишає максимальну свободу дій;
- *якісна робота* – кожен учасник проекту повинен прагнути максимально якісно виконувати свої завдання і сприяти якісній роботі інших.

Кожен принцип втілює заявлені цінності. З-поміж альтернативних методів, які втілюють принципи, в ХР пропонується обирати метод рішення, який відповідає виконанню більшості принципів.

Модель життєвого циклу ХР є ітераційно-інкрементною моделлю швидкого створення (і модифікації) прототипів продукту, що задовольняють черговій вимозі (user story).

Основними фазами моделі можна вважати:

- ***«Вкидання» архітектури*** - початковий етап проекту, на якому створюється бачення продукту, приймаються основні рішення з архітектури та технологій, що застосовуються. Результатом початкового етапу є метафора (metaphor) системи, яка в досить простому і зрозумілому команді вигляді повинна описувати основний механізм роботи системи.
- ***Історії використання (User Story)*** - етап збору вимог, що записується на спеціальних картках у вигляді сценаріїв виконання окремих функцій. Історії використання є вимогами для планування чергової версії і одночасної розробки приймальних тестів (Acceptance tests) для її перевірки.
- ***Планування версії (релізу).*** Проводиться на зборах за участю замовника шляхом вибору User Stories, які увійдуть в наступну версію. Одночасно приймаються рішення, пов'язані з реалізацією версії. Мета планування - отримання оцінок того, що і як можна зробити за 1-3 тижні створення наступної версії продукту.
- ***Розробка*** проводиться відповідно до плану і включає тільки ті функції, які були відібрані на етапі планування.
- ***Тестування*** проводиться за участю замовника, який бере участь у складанні тестів.
- ***Випуск релізу*** - розроблена версія передається замовнику для використання або бета-тестування.

По завершенню циклу робиться перехід на наступну ітерацію розробки.

Особливості моделі життєвого циклу XP прояснюють наступні принципи цього методу. Насамперед, це принципи «живої» розробки ПЗ, що зафіксовані в маніфесті «живої» розробки:

- люди та їх спілкування більш важливі, ніж процеси та інструменти;
- працююча програма більш важлива, ніж вичерпна документація;
- співпраця із замовником більш важлива, ніж обговорення деталей контракту;
- відпрацювання змін більш важливе, ніж дотримання планів.

Крім того, в XP є кілька правил ***(технік),*** *що характеризують особливості моделі його життєвого циклу:*

- ***Живе планування (planning game) –*** якнайшвидше визначити обсяг робіт, який потрібно зробити до наступної версії ПЗ. Рішення приймається на основі, в першу чергу, бізнес-пріоритетів замовника і, по-друге, технічних оцінок. Плани змінюються, як тільки вони починають розходитися з дійсністю або побажаннями замовника.
- ***Часта зміна версій (small releases) –*** перша працююча версія повинна з'явитися якомога швидше, і тут же повинна почати використовуватися. Наступні версії готуються через досить короткі проміжки часу.

- ***Прості проектні рішення (simple design)*** – у кожен момент часу система повинна бути сконструйована так просто, наскільки це можливо. Нові функції додаються тільки після ясного прохання про це. Вся зайва складність вилучається, як тільки виявляється.
- ***Розробка на основі тестування (test-driven development)*** - спочатку пишуться тести, потім реалізуються модулі так, щоб тести спрацьовували. Замовники заздалегідь пишуть тести, що демонструють основні можливості системи, щоб можна було побачити, що система дійсно запрацювала.
- ***Постійна переробка (refactoring)*** - системи для усунення зайвої складності, збільшення зрозумілості коду, підвищення його гнучкості. При цьому перевага віддається більш елегантним і гнучким рішенням, у порівнянні з тими, які просто дають потрібний результат.
- ***Програмування парами (pair programming)*** - весь код пишеться двома програмістами на одному комп'ютері, що підвищує його якість (відсутність помилок, зрозумілість, читання, ...).
- ***Постійна інтеграція (continuous integration)*** - система збирається і проходить інтеграційне тестування якомога частіше, по кілька разів на день кожного разу, коли пара програмістів закінчує реалізацію чергової функції.
- ***40-годинний робочий тиждень*** - понаднормова робота розглядається як ознака великих проблем у проекті. Не допускається понаднормова робота 2 тижні поспіль - це виснажує програмістів і робить їх роботу значно менш продуктивною.

4.6 Гнучке розроблення ПЗ на основі Agile

Постійна зміна вимог під час розроблення ПЗ, необхідність забезпечення ефективної співпраці команди розробників потребували методів, які б забезпечили якість розроблення та супроводу програмних систем і уникнули недоліків занадто формалізованих методів, більшість з яких базувалась на каскадній моделі ЖЦ.

У 90-х рр. XX ст. активно стали виникати різноманітні підходи до створення ПЗ, які забезпечували гнучку роботу програмістів.

У результаті у 2001 році був написаний ***Маніфест гнучкого розроблення (Agile manifesto)***, що зафіксував цінності ефективних підходів до розроблення програмних продуктів, таких, як XP, Feature driven development, Scrum, Adaptive software development, Pragmatic Programming (прагматичне програмування).

Текст маніфесту підписали 17 найавторитетніших фахівців у цій галузі діяльності – Кент Бек (Kent Beck), Алістер Коуберн (Alistair Cockburn), Мартін Фаулер (Martin Fowler) та інші:

Agile-маніфест розроблення програмного забезпечення

Ми постійно відкриваємо для себе досконаліші методи розроблення програмного забезпечення, займаючись розробленням безпосередньо та допомагаючи у цьому іншим. Завдяки цій роботі ми змогли зрозуміти, що:

Люди та співпраця *важливіші за процеси та інструменти;*

Працюючий продукт важливіший за вичерпну документацію;

Співпраця із замовником важливіша за обговорення умов контракту;

Готовність до змін важливіша за дотримання плану.

Хоча цінності, що справа, важливі, ми все ж цінуємо більше те, що зліва.

У результаті з'явився термін – ***Гнучке розроблення програмного забезпечення (Agile software development)*** – клас методологій розроблення програмного забезпечення, що базуються на ітеративному розробленні, в якому вимоги та рішення еволюціонують через співпрацю між самоорганізовуваними багатофункціональними командами.

У маніфесті озвучені основні принципи гнучкого розроблення ПЗ:

- *Найвищий пріоритет* – задоволення потреб замовника шляхом завчасного та регулярного постачання програмного забезпечення.
- *Схвальне ставлення до змін*, навіть на завершальних стадіях розроблення. Agile-процеси використовують зміни для забезпечення конкурентоспроможності замовника.
- *Працюючий продукт* необхідно випускати якомога частіше, з періодичністю від двох тижнів до двох-трьох місяців.
- Упродовж усього проекту *розробники і представники бізнесу повинні працювати разом щодня*.
- *Над проектом повинні працювати вмотивовані професіонали*, які працюють у зручних умовах, із повною підтримкою та довірою менеджменту проекту.
- *Особиста комунікація* – найефективніший та найпрактичніший метод обміну інформацією в команді.
- *Працюючий продукт* – головний показник прогресу.
- Інвестори, розробники і користувачі повинні мати можливість підтримувати *постійний ритм роботи*. Agile допомагає налагодити такий сталий процес розроблення.
- *Постійна увага* до технічної досконалості і якості проектування підвищує гнучкість проекту.
- *Простота* – мистецтво мінімізації зайвої роботи – дуже необхідна.
- Найкращі вимоги, архітектурні та технічні рішення виникають у командах, які здатні *самоорганізовуватись*.
- *Команда постійно шукає способи підвищення ефективності* та відповідно коригує свою роботу.

Більшість гнучких методологій націлені на мінімізацію ризиків шляхом зведення розроблення до серії коротких циклів – ітерацій, які, як правило, тривають один-два тижні. Кожна ітерація є програмним проектом у мініатюрі і включає всі завдання, необхідні для отримання мінімального приросту функціональності: планування, аналіз вимог, проектування, кодування, тестування та документування.

Гнучкий програмний проект передбачає в кінці кожної ітерації отримання працездатного, готового до встановлення у реальному середовищі продукту. Після закінчення кожної ітерації команда виконує переоцінку пріоритетів розроблення.

Гнучкі методи орієнтовані на різні аспекти життєвого циклу розроблення програмного забезпечення. Деякі акцентують увагу на практичних завданнях (екстремальне програмування, прагматичне програмування, Agile-моделювання), інші зосереджені на управлінні програмними проектами (Scrum). Також серед agile-методів існують підходи, які забезпечують повне охоплення життєвого циклу розроблення – метод розроблення динамічних систем (dynamic systems development method, DSDM) та уніфікований процес розроблення (RUP), розглянутий вище. Більшість гнучких методів для використання упродовж життєвого циклу ПЗ потрібно доповнювати іншими підходами, окрім DSDM та RUP.

Особливістю гнучких методів є те, що на даний час відсутні дані про провальні agile-проекти, але є результати опитувань, які підтвердили їх успішне використання. Тому і потужні компанії-розробники активно їх запроваджують у свою діяльність, наприклад, Oracle запроваджує гнучкі методи управління ЖЦ продуктів (Agile product lifecycle management), фірма IBM є власником продуктів, які підтримують використання методології RUP.

Потрібно виділити фактори, які можуть негативно вплинути на використання гнучких методів:

- масштабні зусилля у галузі розвитку (більше 20 розробників);
- розподілена у просторі команда;
- примусове впровадження гнучкого процесу усупереч вимогам команди розробників;

Небажане використання agile-методів у критично важливих системах, де відмова ПЗ неможливий ні в якому разі (наприклад, управління повітряним рухом).

Agile акцентує увагу на безпосередньому спілкуванні між учасниками проекту. Більшість agile-команд розміщені в одному офісі (bullpen). До команди обов’язково включають представників замовника (замовники, які визначають продукт, менеджери продукту, бізнес-аналітики або користувачі). Також до команди входять тестувальники, дизайнери інтерфейсу, технічні автори та менеджери.

Із agile-методів потрібно виділити Scrum, що встановлює правила керування процесом розроблення та дозволяє використовувати вже існуючі практики кодування, коригуючи вимоги або вносячи тактичні зміни. Використання цієї методології дає можливість виявляти і усувати відхилення від бажаного результату на найбільш ранніх етапах розроблення програмного продукту.

Scrum-методи ітераційні та інкрементні. Кожна ітерація (спринт) триває упродовж 15-30 днів і повинна закінчитися приростом функціональних можливостей.

Під час спринту обов'язково відбуваються зустрічі дійових осіб:

- ***Планування спринту (Planning Meeting)*** – відбувається на початку ітерації. Не більше 4-8 годин.
- ***Мітинг (Daily Scrum)*** – відбувається кожного дня упродовж спринту. Триває 15 хвилин.
- ***Демонстрація (Demo Meeting)*** – відбувається в кінці ітерації (спринту). Обмежена 4 годинами.
- ***Ретроспектива (Retrospective Meeting)***. Усі члени команди розповідають про своє ставлення до ходу спринту. Обмежено 1-3 годинами.

Дійові особи розподіляються на дві групи:

повністю задіяні у процесі розроблення («свині»):

- ***Власник Продукту (Product Owner);***
- ***Керівник (ScrumMaster);***
- ***Команда (Scrum Team);***

причетні до розроблення («кури»):

- ***Користувачі (Users);***
- ***Клієнти, Продавці (Stakeholders);***
- ***Експерти-консультанти (Consulting Experts).***

Віддаючи перевагу безпосередньому спілкуванню, agile-методи зменшують обсяг письмової документації порівняно з іншими методами, що викликає критику прихильників більш формалізованих підходів.

4.7 Архітектура ПЗ. Стандарти опису архітектури

Архітектура програми включає в себе найбільш важливі статичні і динамічні аспекти системи та будується на основі вимог до результату. Як було відмічено вище, вимоги відбиваються у варіантах використання. Однак прецеденти також залежать від безлічі інших чинників, таких, як вибір платформи для роботи програми (тобто комп'ютерної архітектури,

операційної системи, СКБД, мережевих протоколів), доступність готових блоків багаторазового використання (наприклад, каркасу GUI), спосіб розгортання, успадковані системи і нефункціональні вимоги (наприклад, продуктивність і надійність).

Архітектура ПЗ – це представлення всього проекту із виділенням важливих характеристик без акценту на деталях.

Кожен продукт має функції і форму. Одне без іншого не існує. Функції відповідають варіанту використання, а форма – архітектурі. В реальних умовах архітектура і прецеденти розробляються паралельно. Архітектура повинна бути спроектована так, щоб дозволити системі розвиватися не тільки в момент початкового розроблення, але і в майбутніх поколіннях.

Щоб знайти таку форму, архітектор повинен працювати, повністю розуміючи ключові функції, тобто ключові варіанти використання системи. Ці ключові варіанти використання становлять 5-10% усіх ВВ, але вони дуже важливі, оскільки містять функції ядра системи.

Архітектор виконує такі кроки:

- *визначає платформу системи* – створює грубий ескіз архітектури, починаючи з тієї частини, що не пов'язана з варіантами використання. Хоча ця частина архітектури не залежить від прецедентів, архітектор повинен у загальних рисах розуміти варіанти використання до створення ескізу архітектури;
- далі архітектор *працює із підмножиною виділених ВВ*, кожен з яких відповідає одній із ключових функцій розроблюваної системи. Кожен з обраних прецедентів детально описується і реалізується в поняттях підсистем, класів і компонентів;
- після того як ВВ описані та повністю розроблені, більша частина архітектури досліджена. Створена архітектура, у свою чергу, буде базою для повного розроблення інших ВВ. Цей процес продовжується до того часу, поки архітектура не буде визнана стабільною.

4.8. Шаблони проектування. Патерни

Створюючи об’єкт, у тому числі й програмний продукт, розробник часто стикається із завданнями, які вже хто-небудь вирішив. У 70-ті рр. XX ст архітектор Кристофер Александер (Christopher Alexander) запропонував шаблони проектування будинків та міст. Через десятиліття ця ідея переросла у процесі розроблення ПЗ до шаблонів проектування інтерфейсу користувача, запропонованих Вардом Каннінгемом (Ward Cunningham) та Кентом Беком.

Далі ідея була активно підхоплена та розвинута у вигляді каталогу патернів ООП. Цей каталог став дуже популярним серед розробників і часто

згадується як патерни GoF («Gang of Four», або «банда чотирьох», – за кількістю авторів). Ідея повторного використання не тільки коду, а й архітектурних та проектних рішень виявилася настільки успішною, що сьогодні патерни проектування широко застосовуються в різних методиках розроблення програмного забезпечення.

У роботі під ***патернами проектування*** об'єктно-орієнтованих систем розуміють опис взаємодії об'єктів і класів, адаптованих для вирішення спільної задачі проектування в конкретному контексті.

Існує багато патернів розроблення програмних систем, які відмінні сферою застосуання, масштабом, змістом, стилем опису. Наприклад, залежно від області використання існують такі патерни, як патерни аналізу, проектування, кодування, рефакторингу, тестування, реалізації корпоративних застосувань, шаблони роботи з базами даних, шаблони розподілених додатків, шаблони роботи із багатопотоковістю, шаблони документування та ін. В останнє десятиліття шаблони впроваджені навіть у роботу менеджера процесу розроблення ПЗ (Джеймса Коплі, Ніла Харрісона «Organizational Patterns of Agile Software Development», Тома Демарко, Тіма Лістера «Adrenaline Junkies and Template Zombies: Understanding Patterns of Project Behavior»).

У даний час найбільш популярними ***патернами є патерни проектування.*** Однією з найпоширеніших класифікацій таких патернів є класифікація за ступенем деталізації та рівнем абстракції розглянутих систем.

Патерни проектування програмних систем поділяються на три категорії (рис. 4.6).

Архітектурні патерни, найбільш високорівневі, описують структурну схему програмної системи в цілому. У даній схемі зазначаються окремі функціональні складові системи (підсистеми), а також взаємовідносини між ними. Прикладом архітектурного патерну є добре відома програмна парадигма "модель-вигляд-контролер" (model-view-controller – MVC).

Рисунок 4.6 – Шаблони проектування програмних систем

Ідіоми, низькорівневі патерни, мають справу з питаннями реалізації певної проблеми з урахуванням особливостей мови програмування. При цьому часто одні й ті самі ідіоми для різних мов програмування мають різний вигляд або взагалі відсутні. Наприклад, в C++ для усунення можливих втрат пам'яті можуть використовуватися інтелектуальні покажчики. Інтелектуальний покажчик містить покажчик на ділянку динамічно виділеної пам'яті, що буде автоматично звільнений при виході із зони видимості. У середовищі Java такої проблеми просто не існує, оскільки там використовується автоматичне складання сміття. Як правило, для використання ідіом потрібно глибоко знати особливості застосовуваної мови програмування.

Завдання кожного патерну – дати чіткий опис проблеми та її вирішення у відповідній області. У загальному випадку опис патерну завжди містить такі елементи:

- ***Назва патерну*** – унікальне змістове ім'я, що однозначно визначає дану задачу або проблему і її рішення.
- ***Розв'язувана задача*** – надається розуміння того, чому розв'язувана проблема дійсно є такою, чітко описує її межі.
- ***Рішення*** – зазначається, як саме дане рішення пов'язане з проблемою, наводяться шляхи її вирішення.
- ***Результати використання патерну*** – як правило, це переваги, недоліки та компроміси.

На рисунку 4.7 показана класифікація шаблонів проектування інформаційних систем.

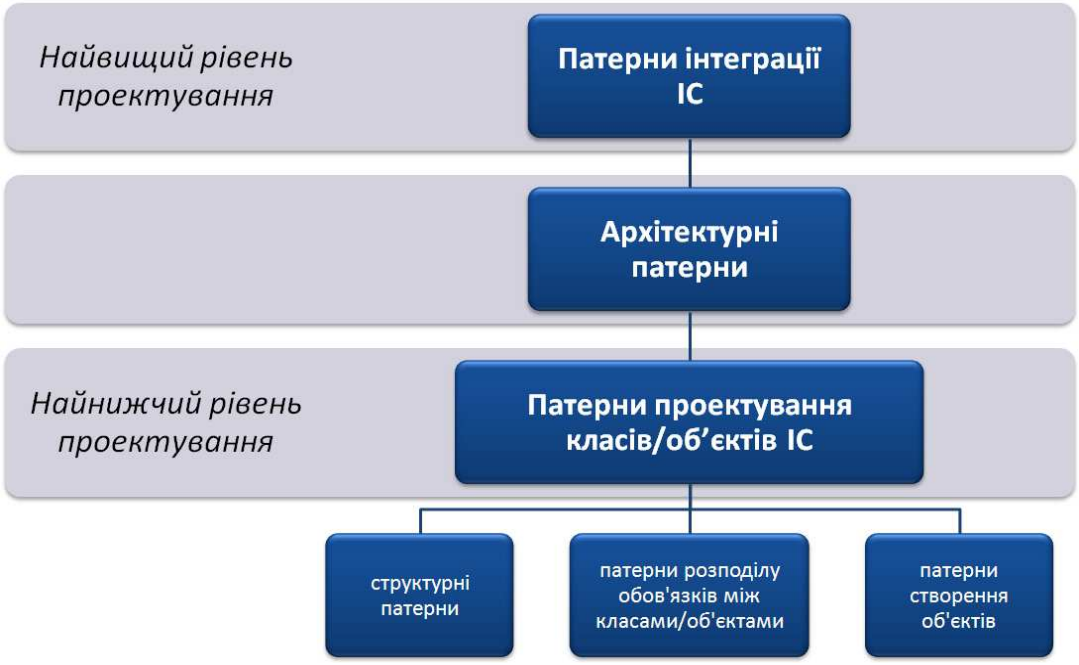

Рисунок 4.7 – Шаблони проектування інформаційнх систем

Архітектурні патерни також об’єднуються у групи:

- *структурні, архітектурні патерни* – слугують для організації класів або об’єктів системи у базовій підструктурі;
- *патерни управління* – для забезпечення необхідного системного функціоналу.

У свою чергу, патерни управління розділені на патерни централізованого управління (патерни, в яких одна з підсистем повністю відповідає за управління, запускає і завершує роботу решти підсистем), патерни управління, які передбачають децентралізоване реагування на події (згідно з цим патернам на зовнішні події відповідає відповідна підсистема), та патерни, які описують організацію зв'язку з базою даних

Патерни інтеграції інформаційних систем знаходяться на верхньому рівні класифікації патернів проектування. Аналогічно патернам більш низьких рівнів класифікації серед патернів інтеграції виділена група структурних патернів. Структурні патерни описують основні компоненти інтегрованої метасистеми. Для опису взаємодії окремих корпоративних систем, включених до інтегрованої метасистеми, використовується група патернів, об'єднаних відповідно до методу інтеграції. Інтеграція корпоративних інформаційних систем передбачає організований обмін даними між системами, для якого використовується відповідний патерн. Необхідно зазначити, що на відміну від патернів проектування класів/об’єктів і архітектурних системних патернів віднесення окремого патерну інтеграції до того чи іншого виду є менш умовним.

Про шаблони проектування знають розробники, проектувальники, архітектори, але про це явище абсолютно нічого не відомо користувачам, для яких і розроблялася мова шаблонів. На сьогоднішній день основна роль шаблонів – це повторне використання досвіду в різних областях розроблення ПЗ, усунення комунікаційного бар'єру всередині команди розробників і між ними, підвищення якості створюваного продукту за рахунок використання перевірених роками рішень.

Питання для самоконтролю

1) Перелічіть візуальні нотації для моделювання предметної області та визначте області їх застосування.
2) Які методології розроблення вам відомі? Наведіть приклади.
3) Перелічіть переваги гнучкого розроблення ПЗ та зазначте її недоліки.
4) Які моделі програмної системи розробляються в уніфікованому процесі RUP? Порівняйте їх.
5) Визначте особливості методології Microsoft Solution Framework.
6) Які цінності та принципи покладені в основу методів eXtreme Programming?
7) Що є особливостями методів Scrum?
8) Порівняйте методології RUP, MSF та XP.

5 ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Як уже згадувалося вище, поточний період на ринку програмного забезпечення характеризується переходом від штучного ремісничого виробництва програмних продуктів до їх промислового створення. Відповідно зросли вимоги до якості розроблюваного програмного забезпечення, що потребує вдосконалення процесів їх розробки.

На даний момент існує кілька стандартів, пов'язаних з оцінкою якості цих процесів, яке забезпечує організація-розробник.

До найбільш відомих відносять:

міжнародні стандарти серії ISO 9000 (ISO 9000 - ISO 9004);

CMM - Capability Maturity Model - модель зрілості (вдосконалення) процесів створення програмного забезпечення, що запропонована SEI (Software Engineering Institute – інститут програмування при університеті Карнегі-Меллон);

робоча версія міжнародного стандарту ISO / IEC 15504: Information Technology - Software Process Assessment; ця версія більш відома під назвою SPICE - (Software Process Improvement And Capability Determination - визначення можливостей і поліпшення процесу створення програмного забезпечення).

У серії ISO 9000 сформульовані необхідні умови для досягнення деякого мінімального рівня організації процесу, але не дається ніяких рекомендацій щодо подальшого вдосконалення процесів.

5.1 Забезпечення якості ПЗ

Якість програмного забезпечення (Software quality) асоціація IEEE визначає як ступінь відповідності програмного забезпечення встановленій комбінації властивостей. У Міжнародному стандарті якості ISO 9000:2007 це поняття визначається як сукупність характеристик ПЗ, які забезпечують встановлені та очікувані вимоги .

До характеристик якості ПЗ відносять (рис. 5.1):

- ***функціональність*** – виконання заявлених функцій, відповідність стандартам, функціональна сумісність, безпека, точність;
- ***надійність*** – стійкість до відмов, можливість відновлення, завершеність;
- ***ефективність*** – економія часу, ефективність використання;
- ***зручність у використанні*** – ергономічність, інтуїтивна зрозумілість, повна документація;
- ***зручність для супроводу*** – стабільність, придатність для контролю та внесення змін;

- ***портативність*** – зручність установки, можливість заміни, сумісність.

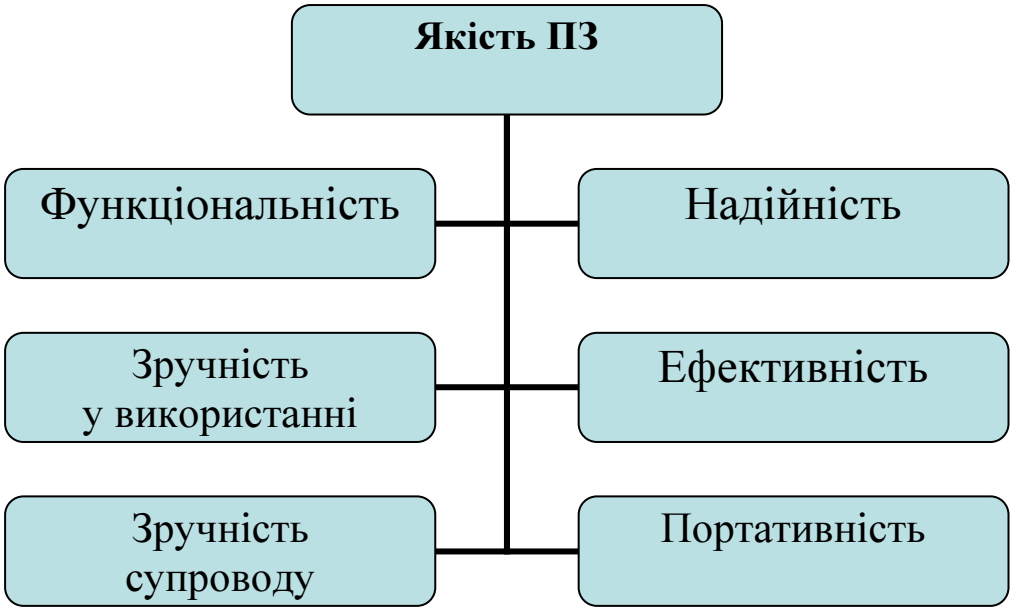

Рисунок 5.1 – Характеристики якості ПЗ

Забезпечення якості (Quality assurance, QA) – сукупність заходів, що охоплюють усі технологічні етапи розроблення, випуску та експлуатації ПЗ інформаційних систем на різних стадіях життєвого циклу ПЗ, для забезпечення його якості.

При виконанні цих заходів для кожного продукту перевіряються:

- повнота та коректність документації;
- коректність процедур встановлення та запуску;
- ергономічність використання;
- повнота тестування.

У 80-х рр. XX ст. розмір проектів із створення програмних систем зріс настільки, що вручну стало неможливо тестувати ПЗ, тому активно стали розроблятися засоби автоматизації процесу тестування. Через дестиліття поняття якості ПЗ розширилося настільки, що було виділено окремий вид діяльності при створенні ПЗ – забезпечення якості (Quality assurance, QA). На цей час автоматизоване тестування значно поширене, а засоби автоматизованого тестування часто вбудовані у середовище програмування.

Для виконання заходів забезпечення якості ПЗ розробники часто використовують спеціальні групи контролю якості, які мають назву QA. Група QA всередині компанії фактично виконує роль вимогливого користувача. У деяких компаніях заборонено неформальне спілкування між групою розробників та групою тестувальників. Від відповідальності групи

QA залежить успіх продукту. Якщо ПЗ не сподобалося, з будь-яких причин користувачам продукт назавжди втрачає репутацію і споживача.

Незнайдені на стадії розроблення помилки коштують дорого. Традиційна стратегія розроблення ПЗ підпорядковується фундаментальному правилу, що визначає, що впродовж роботи над проектом вартість внесення змін у створюване ПЗ збільшується за експонентою (рис. 5.2).

Фактично від того, наскільки якісно виконані початкові етапи розроблення ПЗ залежить його вартість. З метою підвищення якості розроблення та уникнення ризиків, пов'язаних із нечіткими або постійно змінюваними вимогами, була створена методологія XP (eXtreme Programming).

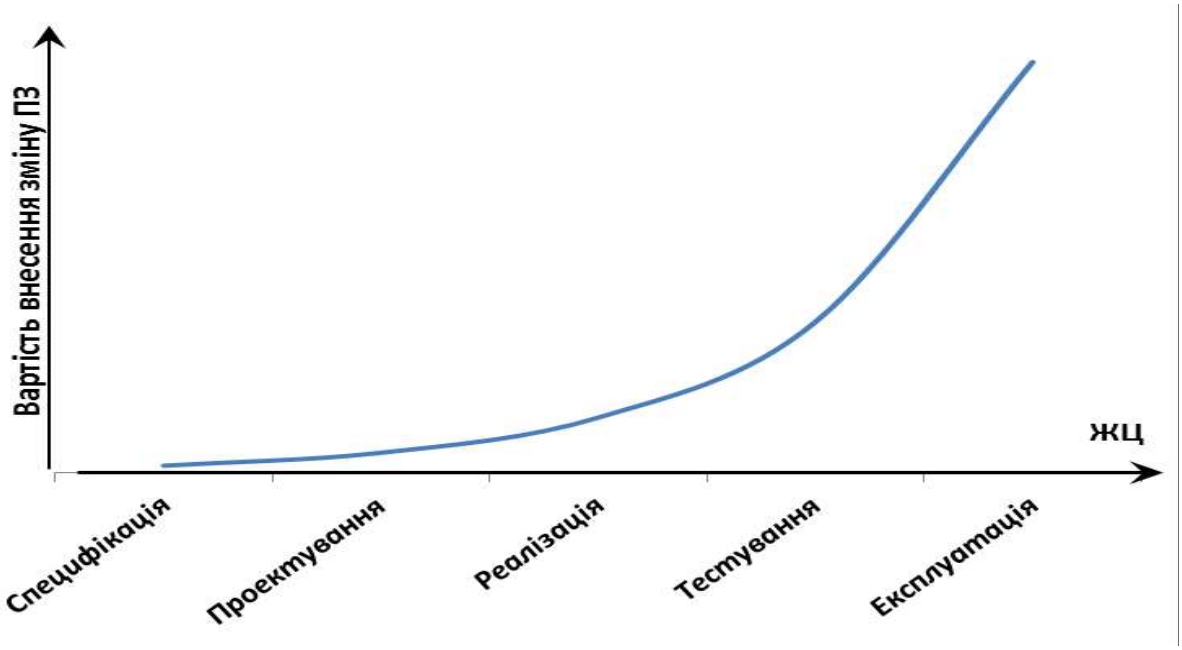

Рисунок 5.2 – Залежність вартості змін проектів від часу

5.2. Тестування ПЗ

Тестування (Software Testing) – діяльність, що виконується для оцінювання та поліпшення якості ПЗ. Ця діяльність базується на виявленні дефектів і проблем програмного забезпечення [14]. Тестування ПЗ включає в себе діяльність із планування робіт (Test Management), проектування тестів (Test Design), виконнання тестування (Test Execution) та аналізу отриманих результатів (Test Analysis).

Потрібно відмітити, що програмування та тестування – різні за суттю види діяльності. Якщо програмування – процес синтезу, то тестування – процес аналізу.

Тестування потребує від виконавця в пешу чергу уважності та педантичності. Тестувальник повинен шукати недоліки будь-де в системі.

Якщо початкове тестування для налагодження коду виконує сам програміст, то наступні етапи перевірки повинні готувати і виконувати інші особи, щоб перевірка була повноцінною, а не тільки для очікуваних проблемних місць.

Для планування потрібно мати уявлення про потрібні обсяги тестування. Важливі статистичні дані щодо тестування:

- на 1000 рядків коду програміст у середньому робить 100 помилок, 70% з яких усуваються на стадії налагодження коду;
- при системному тестуванні у відділі контролю якості на ліквідацію однієї помилки потрібно від чотирьох до шістнадцяти годин;
- для усунення помилки, що була виявлена у ході експлуатації, потрібно від 33 до 88 годин.

Ці дані показують, як важливо виявити та усунути проблеми ще на ранніх етапах розроблення. Цікавим підходом для підвищення якості програмування є парне програмування, що зменшує кількість помилок на 15 % порівняно із традиційним одиночним кодуванням.

Найдавнішим прийомом тестування є перевірка усіх вводів та виводів даних, передбачених та неприпустимих (complete testing). Програма повинна адекватно реагувати на помилкові ситуації, без втрати стійкості в роботі. Також при тестуванні потрібно перевірити виконання усіх передбачених функцій системи. Перелік таких тестів складається на ранніх етапах проектування паралельно із описом функцій програмного продукту.

Види тестування можна класифікувати за різними показниками:

1. **За рівнем знання системи:**
   - *чорний ящик (Black-box testing)* – без перегляду програмного коду;
   - *білий ящик (White-box testing)* – тестування із вивченням коду програми;
   - *сірий ящик (Gray-box testing)* – тестування із частковим вивченням коду програми.
2. **За об’єктом тестування:**
   - *функціональне (Functional testing)* – перевірка виконання заданих функцій;
   - *тестування продуктивності (Performance testing)* – перевірка стабільності роботи програми або її частини при заданому навантаженні (Load testing), при перевантаженні (Stress testing) та тривалому середньому навантаженні (Stability testing);
   - *юзабіліті-тестування (Usability testing)* – перевірка ергономічності ПЗ;
   - *перевірка інтерфейсу користувача (UI testing);*
   - *перевірка безпеки (Security testing)* – тестування роботи системи з точки зору безпеки інформації;

- *тестування сумісноті (Compatibility testing)* – нефункціональне тестування, метою якого є перевірка коректної роботи ПЗ у певному оточенні.

3. **За часом виконання тестування:**
   - *альфа-тестування (Alpha testing)* – перевірка роботи ПЗ перед передачею в експлуатацію силами компанії-розробника;
   - *півгодинне тестування (Smoke testing)* – коротка перевірка функцій системи, як правило, один тест для кожної функції;
   - *тестування нової функціональності (New feature testing);*
   - *регресійне тестуванн (Regression testing)* – перевірка роботи вже протестованих модулів;
   - *тестування під час передачі ПЗ (Acceptance testing);*
   - *бета-тестування (Beta testing)* – тестування ПЗ перед випуском сторонніми силами.
4. **За ступенем ізольованості компонентів:**
   - *компонентне тестування (Component / Unit testing)* – перевірка коректності окремого модуля або компонента системи;
   - *інтеграційне тестування (Integration testing)* – перевірка роботи модулів, об'єднаних у групу;
   - *системне тестування (System / end-to-end testing)* – перевірка роботи зібраного ПЗ з метою встановлення відповідності очікуваним функціям.
5. **За ступенем автоматизації:**
   - ручне тестування (Manual testing);
   - автоматизоване тестуванняя (Automated testing);
   - напівавтоаматизоване тестування (Semiautomated testing).
6. **За ступенем підготовленості до тестування:**
   - *формальне тестування (Formal testing)* – тестування відповідно до плану, встановлених процедур;
   - *інтуїтивне тестування (Ad hoc testing)* – тестування без попереднього плану дозволяє на раніх стадіях виявляти помилки.
7. **7. За ознакою позитивності сценаріїв:**
   - перевірка позитивних сценаріїв (Positive testing);
   - перевірка негативних сценаріїв (Negative testing).

5.3 Методи оцінки якості тестування

Тестування, як і будь-який процес, повино мати методи оцінки якості тестування.

Одним із способів є спосіб, якому в програму спеціально вставляється певна кількість помилок. Після проведення тестування оцінюється, скільки таких помилок було знайдено. Частка знайдених помилок показує якість тестування і допомагає оцінити, що обсяг робіт потрібно виконати для повного усунення проблем. Плануючи тестування, потрібно визначати очікувані результати тестування, які покажуть готовність створюваного продукту. Бажано затвердити їх із замовником для чіткого визначення показників завершення проекту.

Якщо проектувальники тестів повинні мати інтегральне уявлення про систему, то виконавці тестів не потребують високої кваліфікації. Часто цю роль виконують новачки у компаніях, які не мають досвіду роботи.

Для підвищення рівня контролю якості кожного проекту повинна формуватися **база даних помилок**, в яку вноситься така інформація:

- хто знайшов помилку,коли;
- опис помилки;
- модуль, у якому була знайдена помилка;
- версія продукту;
- статус помилки:
  - open: знайдена;
  - fixed: виправлена;
  - can't reproduce: неможливо повторити;
  - by design: помилка проектування;
  - wont fix: не помилка;
  - postponed: зараз виправити важко, буде виправлена у наступній версії;
  - regression: виправлена помилка з’явилася знову;
- важливість (severity) помилки:
  - crash: повна втрата даних;
  - major problem: програма частково не працює, часткова втрата даних;
  - minor problem: програма працює не так, як очікувалось, але дані не втрачаються;
  - trivial: зараз не потрібно виправляти;
- пріоритет помилки:
  - highest: не можна поставити продукт із такою помилкою, не можна перейти до наступної версії;
  - high: поставити не можна, але можна перейти до наступної версії;
  - medium: помилка буде виправлена;
  - low: косметичні поліпшення – помилка залишиється до наступної версії.

Така база даних дозволяє проаналізувати якість роботи окремих програмістів, їх груп, оцінювати якість проекту та можливості використовуваних інструментальних засобів. Також керівники можуть відслідковувати ризики розроблення за помилками найвищих рівнів пріоритету та важливості.

Дані з бази даних помилок дозволяють приймати рішення про можливість випуску програмного продукту (помилки з важливістю "crash" або з пріоритетом "highest/high" не дозволяють поставляти ПЗ). Якщо ПЗ обов’язково потрібно поставити замовнику, а часу на виправлення помилок немає, за домовленістю можна відкинути функції, які виконуються із помилками.

5.4 Види тестів

Одним з найбільш об'єктивних методів оцінки якості програм є її випробування.

Випробування програми може проводитися, наприклад, з метою визначення міри відповідності готової програми вимогам, що сформульовані у технічному завданні.

Так само шляхом випробування можуть бути отримані точні оцінки таких параметрів як:

- середній час вирішення завдання;
- максимальний обсяг необхідний оперативної пам'яті;
- показники завантаження зовнішніх пристроїв необхідних для оцінки вартості рішення задачі.

Іншою, не менш важливою, метою проведення випробувань є спрямований пошук помилок. З якою б метою не проводилося випробування програми, одним з найважливіших питань є підбір вхідних даних програми. Такі, спеціально підібрані з певними цілями, вихідні дані називаються **тестами.** Випробування програми тестами – **тестування** – починається вже в процесі її налагодження.

Перший тестовий приклад пропонується програмою відразу після її трансляції. Далі тестування і налагодження тісно переплітаються між собою. Якщо черговий тест не проходить, тобто програма видає результат, що не відповідає тестовим даним, то виникає проблема знайти і виправити помилку, а це вже **налагодження.**

З іншого боку, для локалізації помилки необхідно вибрати такі вихідні дані, тобто тестові приклади, які б сприяли найбільш рельєфному прояву виявленої, але ще не локалізованої помилки.

Тестування – етап процесу створення програми, що полягає в підборі тестових прикладів і випробування на них програми, з метою виявлення в ній помилки.

Налагодження – етап процесу створення програми, що полягає в локалізації та виправленні помилок, виявлених у ході трансляції та тестування.

Етап тестування включає три основних елементи:

1. генерація тестових прикладів, контроль процесу тестування;
2. аналіз вихідних і проміжних результатів програми;
3. врахування внеску кожного тестового прикладу в процес тестування.

Для першого і другого елементів вихідними даними є документація програми, на основі якої генеруються тестові приклади, що забезпечують режим виконання програми, визначає ступінь відповідності отриманого результату зазначеного в прикладі.

Третій елемент визначається критерієм повноти тестування, який тісно пов'язаний з конкретним способом тестування.

Тестування ведеться на чотирьох рівнях:

1. рівень окремого програмного компонента (модуля);
2. рівень сполучень, на якому шукаються помилки між-компонентного інтерфейсу;
3. рівень зовнішніх функцій, на якому шукаються розбіжності програмних функцій і зовнішніх специфікацій програм;
4. комплексне тестування, випробування програмного комплексу на відповідність вихідним цілям.

Цілком природно тестування ув'язується з організацією розробки програм. Тому, два способи утворюють базис, на якому будуються різні методи тестування.

Висхідне тестування передбачає, що програма збирається і тестується знизу вгору. Модулі (компоненти самого нижнього рівня) тестуються автономно. Надійність тестування цих модулів визначає успіх подальшого процесу. Далі відбувається перехід до модулів, які звертаються до вже відтестованих модулів. На цьому етапі виникає необхідність перевірки інтерфейсів.

Для реалізації висхідного тестування необхідно для кожного модуля написати невелику керуючу підпрограму - драйвер. У розпорядження драйвера надаються значення вхідних змінних і структури даних. Драйвер послідовно викликає модуль, що тестується, при кожному виклику пропонуючи йому новий тестовий приклад. Основні недоліки:

- серйозні помилки в специфікаціях, алгоритмах і інтерфейсах можуть проявлятися лише при тестуванні комплексів вищого рівня, тобто на завершальній стадії.

- Необхідність розробки драйверів і тестів для кожного рівня тестування, що веде до великого обсягу додаткових програм, які стають даремними при завершенні роботи над комплексом.

Спадне проектування. Автономно тестується тільки головна програма. По завершенні тестування головного комплексу до нього послідовно приєднуються комплекси та компоненти наступного рівня і т.д., до тих пір, поки не буде зібрана і випробувана вся програма.

Як же тестувати комплекси, в той час як компоненти, що входять в них, ще не перевірені і можливо ще і не написані? Для імітації функцій ще не створених модулів використовуються заглушки, які імітують роботу відсутнього модуля.

Недоліки спадного тестування збігаються з вадами спадного проектування. Підвищуються вимоги до складності і якості заглушок, які потім ліквідуються. Переваги:

- Метод дозволяє поєднати тестування модуля, тестування сполучень і тестування вхідних функцій.
- Рівномірний розподіл роботи з тестування протягом усього періоду створення комплексу - це дозволяє виявити помилки в головному модулі на ранній стадії розробки.

На практиці рідко вдається використовувати один спосіб, існує ряд комбінованих способів.

Методи тестування компонентів

Перший метод розглядає тестування програм як «чорний ящик», при цьому внутрішня структура програми (компонента) не враховується, тести будуються на підставі функціональних властивостей програми, тобто спираючись на її функціональні специфікації. Такий підхід називається ***функціональним тестуванням.***

Структурне тестування, при його використанні враховується внутрішня структура програми. Аналіз проходження даних від входу до виходу, ця інформація використовується для раціональної організації тестування.

Для оцінки повноти тестування використовують три критерії:

1. тестування вважається закінченим, якщо кожен оператор був виконаний хоча б раз;
2. тестування вважається закінченим, якщо в процесі вирішення тестових прикладів по кожній дузі блок-схеми програми був здійснений хоча б один перехід;
3. тестування закінчується, якщо в процесі вирішення тестового прикладу кожен шлях від входу до виходу пройдений хоча б раз.

Налагодження - процес пошуку та усунення помилок у програмі, що спирається на результати самої програми, повідомлення транслятора, і оперативної системи. Важливою особливість процесу налагодження є

можливість робити експерименти на ЕОМ з метою виявлення помилок. Але експеримент повинен вестися в суворій відповідності з планом.

Процес налагодження полягає в багаторазовому повторенні трьох етапів:

1. виявлення помилки;
2. локалізація помилки;
3. виправлення помилки.

Виявлення помилки при налагодженні здійснюється шляхом прорахунку на ЕОМ спеціально підібраних завдань, результати вирішення яких заздалегідь відомі. Якщо контрольний приклад вирішено правильно, то підбирається більш складне завдання. Якщо програма не йде, то в ній є, принаймні, одна помилка.

Наступний етап полягає у встановленні точного місця знаходження помилки. ***Локалізація помилок*** є процес вирішення неправильних завдань.

Етап ***виправлення помилки*** є найбільш простим, але основна складність не внести нову помилку при виправленні.

5.5 Верифікація програм

Особливістю програмного продукту є практична неможливість в нетривіальних випадках здійснити всебічне і повне його випробування з метою виявлення помилок.

Відомий вислів Дейкстри говорить про те, що експериментальне тестування програм може служити доказом наявності в них помилок, але ніколи не доведе їх відсутність. Тому природно прагнення програмістів знайти можливість формулювати і доводити деякі твердження стосовно правильності створеної програми подібно до того, як у математиці формулюються і доводяться теореми і рішення.

Верифікація – *ідея математичного доказу коректності програм*.

Верифікація зазвичай зводиться до доказу того факту, що програма є коректною відносно її вхідної і вихідної специфікацій.

Найбільш відомий метод називається **методом індуктивних тверджень.**

У цьому методі перший крок полягає в записі тверджень щодо властивостей вхідних і вихідних даних програми, а також результатів у ряді проміжних точок, що називаються **точками розрізу.** Ці твердження формулюються в деякій формально-логічній системі. На основі цих тверджень і семантики операторної схеми програми шляхом певних перетворень формулюються ***верифікаційні умови,*** а потім ці умови доводяться. Якщо вони виявляються істинними, то програма коректна щодо вхідних і вихідних даних.

Якщо довести істинність умов неможливо, то або в програмі є помилки, або помилки є у процедурі доказу (наприклад, хибне твердження в деякій точці розрізу).

Якщо програміст хоче написати правильну програму, йому необхідно виконати такі етапи:

- написати програму;
- визначити вхідні і вихідні умови для цієї програми;
- розрізати цикли і забезпечити кожну точку розрізу індуктивним затвердженням;
- отримати верифікаційні умови;
- довести істинність верифікаційних умов; якщо це не вдається, то не виключено, що якесь індуктивне твердження записано невірно або програма містить помилку.

Очевидно, що методи верифікації знайдуть широке застосування в практиці у тому випадку, якщо вдасться їх формувати настільки явно, що можна буде здійснювати верифікацію програм за допомогою ЕОМ.

Використання в практиці програмування систем доказу правильності програм ефективно лише за допомогою ЕОМ.

Методи математичного доказів теорем в даний час інтенсивно розвиваються і удосконалюються. Їх застосування в програмуванні сприяє подальшому прогресу у цій галузі.

Ідеї доказу правильності програм роблять свій вплив на загальну культуру програмування. Необхідність формального опису вхідних і вихідних даних спонукає програміста чітко визначати інтерфейси між модулями програми. Механізм тверджень являє собою чудовий засіб специфікацій модулів. Далі, намагаючись винайти індуктивні твердження, програміст змушений більш ретельно і глибоко аналізувати свою програму і таким чином виявляти в ній помилки.

Питання для самоконтролю

1) Які типи вимог до ПЗ можна виділити? Як виконується опис цих вимог?
2) Як пов’язані між собою строки проектних робіт, їх обсяг та вартість? Яким чином, керуючи цими показниками, можна забезпечити якість проекту?
3) Які елементи повинен містити проектний план?
4) Які форми планів проектних робіт вам відомі? У чому принципова різниця між ними?
5) Охарактеризуйте принципи керування проектом.
6) Які складові методології розроблення потрібно враховувати при її виборі?

7) З якими ризиками стикаються розробники ПЗ?
8) Перелічіть та коротко опишіть характеристики якості ПЗ.
9) Що таке Quality assurance, чим воно відрізняється від тестування?
10) Опишіть види тестування.
11) Яку інформацію доцільно зберігати в базі даних помилок?

6 МАРКЕТИНГ ПРОГРАМНИХ ПРОДУКТІВ

Комерційно обгрунтована розробка програмного забезпечення і його маркетинг на ринку можливий у тому випадку, якщо програми є товаром для даного ринку. Перетворення програм у “програмний товар“ здійснюється шляхом доробки цих програм під вимоги ринку, а також подальшим забезпеченням організаційних, технічних і комерційних умов по маркетингу, експлуатації, підтримці і розвитку цього пакету (товару) на ринку.

Відносно низька ціна на програмне забезпечення для персональних комп’ютерів та іншої електронної техніки дозволяє багатьом користувачам купувати його без детального тестування та отримання необхідного обслуговування з боку постачальника. Це є основною причиною появи постачальників програмного забезпечення, що приймають замовлення на постачання й організують це постачання поштою (електронною поштою). На сьогодні через глобальні комп’ютерні мережі не тільки здійснюється продаж, але й організується демонстрація програмного забезпечення, ознайомлення користувача з документацією, а також надання потенційному покупцю інформації про переваги і недоліки в порівнянні з аналогічними пакетами.

Програмне забезпечення перетворюється в товар, якщо воно задовольняє певним вимогам ринку. Тому для будь-якого постачальника для успішної торгівлі програмним забезпеченням дуже важливо виконувати всі основні вимоги ринку, що ставляться.

6.1 Основні ринкові вимоги доПЗ

Основні вимоги ринку можна сформулювати такими пунктами:

- функціональні вимоги;
- характеристики якості;
- вимоги до документації;
- вимоги по розвитку програмного забезпечення, обслуговуванню і навчанню;
- вимоги по цінах;
- маркетинг програмного забезпечення.

В площині функціональних вимог програмне забезпечення повинно :

- забезпечувати функціональні можливості, що достатні для розв’язання певного класу проблем, що задовольняють вимогам конкретної категорії покупців;
- підтримувати широко використовувані на ринку ЕОМ і сучасне периферійне устаткування;

- опрацьовувати інформацію і спілкуватися з користувачем із використанням алфавіту і мови, зрозумілої користувачу.

Функціональні можливості програмного забезпечення “вкладаються“ при його розробці на основі аналізу і вивчення попиту ринку, і повинні відповідати поточним потребам ринку і навіть випереджати їх. Функціональні можливості відбивають ті ідеї, що реалізує програмне забезпечення для конкретної сфери застосування. Конкурентоспроможність програмного забезпечення на ринку в першу чергу визначається сучасністю і цінністю алгоритмів і ідей, що реалізовані в програмному забезпеченні. Якщо програмне забезпечення розробляється за замовленням конкретного користувача, то його функції визначаються разом із замовником. Програмне забезпечення повинне підтримувати сучасні периферійні пристрої, які широко використовуються на ринку з даним класом ЕОМ, і забезпечувати користувачу можливість задання вхідної і вихідної інформації на його рідній мові. Для системного програмного забезпечення інтерфейс із користувачем можливий англійською мовою. У багатьох випадках використання систем розробки програм також допускається на рівні англійської мови, проте, прикладне програмне забезпечення повинне забезпечувати користувачу спілкування на рідній мові.

В програмне забезпечення повинен входити контрольний приклад, що перевіряє функціонування програмного забезпечення. Можливості контрольного прикладу визначаються розробником. Програмне забезпечення оформляється, як окремий незалежний компонент, що, поставляється на машинному носії (магнітні стрічки, диски).

6.2 Оцінка якості ПЗ з позиції маркетингу

Широке використання обчислювальної техніки і програмного забезпечення в критичних для життя людей сферах застосування (оборона, медицина, транспорт і т.д.) істотно підвищило значення якісних характеристик програмного забезпечення і його надійності. Якість програмного забезпечення може бути визначена, як набір властивостей і характеристик, що дозволяють задовольнити заданим потребам користувачів. Таких характеристик дуже багато. У залежності від призначення і специфіки поставленої задачі кількість і вага характеристик якості може бути різноманітним. Проте існують узвичаєні характеристики, що експортер повинен чітко показати клієнту, і які є істотними при оцінці конкурентоспроможності програмного забезпечення. Основними з них є такі:

- функціональність, тобто спроможність програмного забезпечення правильно виконати набір передбачених функцій для задоволення вимог користувачів ;
- надійність, тобто спроможність забезпечити необхідний рівень працездатності (безперебійної роботи) при заданих умовах;
- простота використання, тобто наявність таких атрибутів, що дозволяють мінімізувати зусилля користувача при вивченні, використанні й оцінці програмного забезпечення;
- економія ресурсів і часу, тобто спроможність програмного забезпечення виконати необхідні функції, використовуючи допустиму кількість ресурсів у визначений відрізок часу;
- сумісність, тобто можливість швидкого і легкого настроювання програмного забезпечення на інше устаткування (середовище функціонування);
- супровід, тобто наявність таких атрибутів, що сприяє мінімізації зусиль по модифікації і внесенню виправлень у програмному забезпеченні.

У загальному значенні “продаж“ програмного забезпечення не означає передачі прав власності, як наприклад, у випадку продажі устаткування. Під продажем програм розуміється надання користувачу ліцензії на користування цими програмами. Користувач може одержати ліцензію на користування програмного забезпечення безстроково або на визначений період. Умови оплати можуть базуватися або на одноразовому платежі або помісячному. Звичайно перший варіант оплати застосовується за програмне забезпечення для персонального комп’ютера, другий - за програмне забезпечення для великих ЕОМ. Алгоритм утворення цін на програми (програмне забезпечення) залежить від багатьох чинників: конкуренції на ринку аналогічного програмного забезпечення, вартості розробки, ринкового попиту на даний клас товарів, повноти відповідності вимогам ринку, потенційної бази клієнтів та інших. Деякі з цих чинників, наприклад, вартість розробки, для користувача є несуттєвими. З погляду користувача цінність програмного забезпечення визначається рівнем його відповідності вимогам ринку. І, відповідно до цього, вплив вимог ринку є визначальним в утворенні ціни на програмне забезпечення.

Розробнику (постачальнику) програмного забезпечення на ринок варто чітко розуміти, що в умовах жорсткої ринкової конкуренції продаж програмного забезпечення без організації і проведення маркетингу практично неможливий. Під маркетингом слід розуміти процес виконання

всього комплексу заходів, пов'язаних із виходом на ринок і роботою там із клієнтами., як правило, ці заходи включають: придбання прав від розробника на маркетинг програмного забезпечення, тестування і “приведення“ до товарного вигляду, підготовка і випуск рекламних, навчальних і інших інформаційних матеріалів, організація реклами і демонстрації, визначення реальної ціни, аналіз і створення каналів збуту на ринку, організація роботи з клієнтами (навчання, обслуговування, реакція на скарги клієнтів і т.д.), розробка комерційних принципів роботи з агентом і (або) клієнтами (фінансові питання, контроль за зберіганням права власності на програмне забезпечення), постачання програмного забезпечення місцевому агенту в справах продажу і постійної роботи з агентом. Роботи з маркетингу виконують звичайно ті фірми, що беруть на себе роль фірми-видавця програмного забезпечення (s/w publisher) або фірми-агента.

Застосування користувачем тих або інших критеріїв для оцінки програмного забезпечення залежить від багатьох чинників: досвіду покупця, специфіки вимог покупця, кількості наявних на ринку аналогічного програмного забезпечення, рівня підтримки постачальником і т.д. Проте, можна виділити певний набір критеріїв, що використовуються більшістю покупців, і який є достатнім для оцінки будь-якого програмного забезпечення. Звичайно, вибір потрібного програмного забезпечення і його оцінки є тривалим процесом, але він необхідний, якщо користувач розраховує одержати економію чи інші вигоди у своїй роботі від експлуатації пакету.

Основними критеріями оцінки програмного забезпечення є:

1. Відповідність основних функцій програмного забезпечення вимогам покупця.
2. Степінь точної і повної відповідності можливостей програмного забезпечення потребам покупця. Тобто проводиться оцінка програмного забезпечення на відповідність даному критерію, яка містить у собі розгляд таких питань:

- чи відповідають формати і зміст вхідної і вихідної інформації вимогам покупця;
- чи відповідають формати функціональних можливостей програмного забезпечення потребам покупця;
- чи організація файлів і керування даними задовольняють покупця;
- які є додаткові можливості для вводу, виводу й обробки даних;

- який контроль помилок забезпечує програмне забезпечення;
- які передбачені процедури по відновленню програмного забезпечення у випадку відмови;
- як забезпечується захист даних і файлів.

3. Чи буде програмне забезпечення працювати на ЕОМ покупця? Оцінка по цьому критерію полягає в розгляді таких питань:

- чи існує версія програмного забезпечення для моделі ЕОМ покупця;
- скільки потрібно оперативної пам'яті для нормальної роботи програмного забезпечення;
- скільки і які типи пристроїв підтримуються програмним забезпеченням;
- скільки потрібно каналів вводу - виводу;
- будуть потрібні чи ні додаткові апаратні засоби і які;

У випадку, якщо буде потрібно додаткове устаткування, повинні бути розглянуті питання ціни і можливості використання такого устаткування. Якщо конфігурація ЕОМ є близькою до мінімальної, тоді необхідно вирішити чи ефективно використовувати програмне забезпечення на такій конфігурації.

4. Яку програмну підтримку необхідно для використання програмного забезпечення на ЕОМ, і які вимоги до цього накладаються?

- чи може програмне забезпечення працювати під керуванням операційної системи, що використовується звичайно покупцем на його ЕОМ;
- чи можливо використовувати наявне в покупця програмне забезпечення разом із новим і в якій мірі;
- на якій мові програмування написане програмне забезпечення і чи є ця мова стандартною і широко використовуваною;
- чи можуть спеціалісти покупця обслуговувати і модифікувати програмне забезпечення у разі потреби.

5. Чи відповідає продуктивність програмного забезпечення вимогам покупця? Продуктивність програмного забезпечення, що має однакові функціональні можливості, може дуже істотно відрізнятися і залежати від багатьох чинників (таких як: розмір

використовуваної пам'яті, методи опрацювання інформації, методи вводу - виводу і т.д.), і тому оцінити її досить важко. Найкращий спосіб оцінки - тестування програмного забезпечення на реальних задачах на ЕОМ покупця. Проте, це дорогий засіб, і покупці звичайно користуються інформацією про характеристики програмного забезпечення з різноманітних довідників, а також ознайомлюються з відгуками користувачів, які вже його використовують.

6. Наскільки легко можна адаптувати програмне забезпечення у тому випадку, якщо вимоги клієнта по опрацюванню даних будуть змінюватися. Цей критерій включає розгляд таких питань:

- чи володіє програмне забезпечення функціональними можливостями, що на даний час не входять в потреби клієнта, але які будуть потрібні в майбутньому;
- чи є обмеження на розміри записів, файлів, якщо клієнт планує збільшити об'єм і структуру даних, що оброблятимуться.

7. У якій формі поставляється програмне забезпечення?

Звичайно програмне забезпечення поставляється в об'єктному вигляді, тобто у вигляді машинних кодів. Проте, у тих випадках, коли користувач планує змінювати його і обслуговувати його надалі, він намагається одержати пакет у вигляді вихідного модуля і роздруків на вихідній мові., як правило, ціна програмного забезпечення у вихідному вигляді значно перевищує об'єктного модуля.

8. Наскільки складний процес встановлення програмного забезпечення на ЕОМ і пов'язані з цим роботи. При цьому покупець розглядає такі питання:

- які зміни буде потрібно зробити в існуючій в нього системі, процедурах і методиках;
- чи потрібно змінювати структуру файлів і їхнє розміщення на носіях, - якщо так, тоді, як це зробити і коли;
- чи бере участь постачальник при встановленні програмного забезпечення, на скільки ефективна його допомога, як оцінюється вартість такої допомоги;
- чи потрібно розробляти спеціальну документацію на додаток до тієї, що надає постачальник;

- які потрібні людські ресурси, та яке навчання буде потрібно для них.

9. Легкість експлуатації програмного забезпечення користувачем оцінюється по таких основних пунктах:

- чи розробляється програмне забезпечення з орієнтацією на обчислювальну систему користувача і чи існує повна документація, що описує процедуру експлуатації програмного забезпечення на ЕОМ;
- чи є вхідні форми і документи, а також інструкції з їхнього підготовки достатньо ясними і зрозумілими;
- чи вважає кожен працівник, що буде використовувати це програмне забезпечення, що програмне забезпечення задовольняє його потреби з мінімальними витратами.

10. Кількість і якість документації, що поставляється з програмним забезпеченням.

Документація повинна складатися з :

- рекламної,
- експлуатаційної,
- документації розробника (супровід).

Рекламна документація призначена для маркетингу (потреб продавця) і повинна містити основні дані про програмне забезпечення (функціональні можливості, основні характеристики, включаючи вимоги до ЕОМ і пам'яті, підтримувані пристрої, умови експлуатації; кількість вже наявних користувачів, можливостях у порівнянні з аналогічними продуктами). Рекламна документація повинна бути надрукована англійською або рідною мовою користувача і складатися з проспектів і загальних описів.

Експлуатаційна документація призначена для користувачів даного програмного забезпечення, тобто:

- -системних спеціалістів, що встановлюють програмне забезпечення на ЕОМ;
- -прикладних спеціалістів і програмістів, що використовують програмне забезпечення при розробці алгоритму і програмуванні прикладних програм.

Вона повинна містити, як правило, такі документи :

- загальний опис (концепції і можливості) англійською чи рідною мовою користувача;
- інструкцію для встановлення на рідній мові користувача в середовищі операційної системи і використовуваного устаткування (ЕОМ);
- інструкцію по експлуатації (використанню) програмного забезпечення;
- повідомлення, що видаються програмним забезпеченням;
- інструкцію для запуску контрольного прикладу.

Кількість і обсяги документації залежить від складності і компонентного складу документації (опис мови, опис застосування даного програмного забезпечення, інструкція для оператора і т.д.). Інструкція з експлуатації для прикладного програмного забезпечення повинна бути написана рідною мовою користувача. Документація повинна бути написана простою, доступною мовою, особливо, якщо ця документація призначена для користувачів персональних комп'ютерів. Документація розробника насамперед призначається для персоналу, що здійснює розвиток і підтримку цього програмного забезпечення. Якщо передбачається розвиток і супровід такого програмного забезпечення, документація повинна містити також:

- текстовий опис логіки програмного забезпечення;
- докладні блок-схеми;
- тексти програм на вихідній мові програмування.

Документація розробника може бути написана на рідній мові розробника.

11. Яку підтримку програмного забезпечення буде забезпечувати постачальник? Розробник (постачальник) повинен супроводжувати використання програмного забезпечення і його обслуговування (виправлення помилок, як у програмах, так і в документації). Розвиток програмного забезпечення є істотним не тільки для користувача, який з початком його використання здійснює свою діяльність в заданій ідеології (перехід до використання іншого програмного забезпечення у загальному випадку означав би для користувача перенавчання персоналу, перепланування методики опрацювання даних і т.д.), але і не менш важливо для самого розробника. Постійний розвиток програмного забезпечення дозволяє розробнику зберігати вже

існуючу базу клієнтів, а також робити малоефективним несанкціоноване копіювання.

Розробник разом із постачальником програмного забезпечення повинен забезпечити такі основні послуги по обслуговуванню клієнтів :

- встановити програмне забезпечення у клієнта, якщо буде потрібно;
- періодично поставляти клієнту зміни до програмного забезпечення і документації;
- виправляти виникаючі помилки в програмному забезпеченні (вимоги по швидкості виправлення помилок залежить від характеру помилки, частоти використання клієнтом - щодня, щотижня, щомісяця і т.д.);
- проводити консультації для клієнтів про можливості програмного забезпечення і його використання;
- брати участь у разі потреби, у спільних із клієнтом проектах із застосуванням даного програмного забезпечення;
- періодично інформувати клієнтів про поточний стан і перспективи розвитку програмного забезпечення через розсилання технічних повідомлень, організацію семінарів.

Обслуговування і навчання користувачів - платне. Навчання проводиться на рідній мові користувача, в окремих випадках - на англійській, можливі варіанти з перекладачем.

12. Коли було розроблено програмне забезпечення, як довго воно експлуатується і скільки клієнтів його використовують? Відповіді на ці питання дадуть уявлення про налагодженість, якості й ефективності програмного забезпечення.

13. Яка загальна вартість придбання і використання програмного забезпечення? У загальну вартість необхідно включати, як вартість самого програмного забезпечення, так і ті витрати, що можуть виникнути в результаті модифікації під вимоги користувача, зміни вже існуючої в користувача системи, навчання персоналу і конвертування програм і файлів користувача.

14. Оцінка постачальника програмного забезпечення.

- оцінка кваліфікації і досвіду роботи на ринку розробника і постачальника програмного забезпечення;

- досвід і популярність фірми-агента, що здійснює маркетинг програмного забезпечення на місцевому ринку.

15. На яких фінансових умовах продається програмне забезпечення, тобто

- можливість покупки і (або) оренди;
- які існують специфічні умови при купівлі / оренді;
- чи існують обмеження на використання або модифікацію програмного забезпечення, а також на надання сервісу по обслуговуванню та які;
- яка дається знижка при експлуатації пакету на декілька ЕОМ в одного користувача;
- чи існують додатково оплачувані режими в програмному забезпеченні, якщо так, тоді яка їхня ціна;
- ціна за обслуговування і навчання.

6.3 Процес придбання ПЗ користувачем

Під процесом придбання програмного забезпечення будемо розуміти послідовність дій покупця від моменту ухвалення рішення про необхідність придбання потрібного програмного забезпечення до моменту початку експлуатації. Нижче описується така послідовність дій покупця, що визначає прийнятий на ринку процес вибору і купівлі програмного забезпечення.

1. *Визначення вимог до програмного забезпечення*. Покупець, виходячи з аналізу задач, що стоять перед ним, визначає можливі шляхи їх вирішення. Для цього йому насамперед потрібно визначити:

- основні функції, які повинно виконувати програмне забезпечення; які звичайно визначаються через опис вхідних даних, алгоритмів їх опрацювання та форми вихідних даних;
- конфігурацію устаткування, операційну систему, мови програмування й інші технічні і програмні засоби, що покупець планує використовувати разом із даним програмним забезпеченням;
- можливі зв'язки з вже існуючим програмним забезпеченням в покупця.

2. Збір інформації про наявність на ринку програмного забезпечення, що потенційно може задовольнити покупця. Джерелами такої інформації звичайно служать :

- спеціалізовані звіти;
- огляди і статті по програмному забезпеченню у періодичних часописах і газетах по програмуванню;
- рекламні матеріали фірм-постачальників програмного забезпечення;
- консультанти в сфері маркетингу програмного забезпечення;
- користувачі програмного забезпечення.

3. *Аналіз програмного забезпечення*, яке потенційно задовольняє користувача, з метою вибору декількох прикладних пакетів для їх подальшої детальної оцінки і порівняння. Такий детальний аналіз проводиться по таких основних критеріях програмного забезпечення:

- - ціна;
- - основні функціональні можливості;
- - необхідне устаткування.

4. *Одержання інформації про програмне забезпечення* у користувачів пакету. Через користувачів програмного забезпечення покупець з'ясовує ті реальні проблеми, із якими зіткнувся користувач при встановленні та експлуатації. Наприклад, обмеження на використання, виявлені помилки, які вдосконалення були б бажані, як у дійсності організовані підтримка і допомога з боку постачальника.

5. *Прийняття рішення*. Оцінити яке із розглянутого програмного забезпечення найповніше задовольняє вимогам покупця і при цьому має нижчу ціну і прийняти рішення про придбання.

6. *Обговорення і підписання контракту*. Не існує єдиного стандарту на контракт на купівлю або оренду програмного забезпечення. Проте, незважаючи на те, що кожний постачальник програмного забезпечення пропонує покупцю свій варіант контракту, основні умови і положення їх однотипові і визначаються, як постачальником, так і покупцем. Нижче перераховуються загальновизнані положення контракту.

- Функціональні можливості, спеціальні й інші властивості і можливості програмного забезпечення повинні бути чітко специфіковані. Це можна

зробити, наприклад, посиланням у контракті на повний комплект документації.

- У контракті специфікується предмет постачання, як-от: детальний перелік складових частин програмного забезпечення, у якій формі воно поставляється (вихідний або об'єктний код), специфікація повного комплекту документації.
- Може бути визначений гарантійний період з моменту встановлення програмного забезпечення, протягом якого постачальник здійснює обслуговування і допомогу покупцю в освоєнні роботи безплатно або на пільгових умовах.
- Навчання, обслуговування та встановлення програмного забезпечення, запропоновані постачальником, повинні бути чітко визначені з вказівкою їхньої вартості.
- Повинні бути визначені: методика та максимальний термін виправлення помилок у програмах і документації в післягарантійний період.
- Виправлення помилок у програмному забезпеченні в післягарантійний період постачальник програмного забезпечення здійснює за власний рахунок.
- Для перших користувачів програмного забезпечення може бути організована постачальником додаткова допомога в освоєнні (наприклад, триваліший гарантійний період, безкоштовна експлуатація в початковий період та іншу допомогу).
- Користувач може мати право модифікувати програмне забезпечення. У цьому випадку постачальник не зобов'язаний здійснювати обслуговування модифікованого програмного забезпечення.
- У випадку банкрутства або припинення своєї діяльності з іншої причини постачальник повинен гарантувати користувачу надання усіх вихідних матеріалів про програмне забезпечення для того, щоб користувач міг самостійно підтримувати роботу програмного забезпечення. В цьому випадку в контракті повинні бути обговорені комерційні й інші умови надання таких матеріалів.
- Постачальник повинен гарантувати патентну чистоту програмного забезпечення.
- Користувач повинен гарантувати постачальнику зберігання права власності постачальника на програмне забезпечення, тобто не копіювати, не передавати, не продавати і т.д. програмне забезпечення третій стороні без дозволу постачальника.
- Користувач повинен мати право протягом гарантійного періоду розірвати контракт і повернути програмне забезпечення без додаткових

витрат для себе в тому випадку, якщо воно не відповідає специфікації і не виконує або виконує, але невірно, описані функції.

- Якщо програмне забезпечення орендується на певний строк, тоді після закінчення цього терміну користувач повинен або повернути всі копії програмного забезпечення і документації постачальнику, або повернути всі копії документації і знищити програми.
- У контракті повинні бути визначені процедура і критерії приймання програмного забезпечення користувачем.
- Платежі повинні здійснюватися пропорційно частинам програмного забезпечення, що поставляються, наданому сервісу, а також після того, як користувач переконався в правильній працездатності продукту відповідно до специфікації його властивостей і функцій.
- У контракті повинен бути визначений штраф, як для постачальника, так і для користувача у випадку порушення узятих ними зобов'язань або положень контракту.
- Ціна на програмне забезпечення формується з врахуванням багатьох чинників: витрат на розробку, рівня конкуренції на ринку, рівня попиту ринку на даний клас продуктів, відповідності характеристик пакету вимогам ринку, досвіду і популярності розробника і продавця на ринку. Тому визначення реальної ціни програмного забезпечення на ринку потребує врахування всіх цих характеристик, причому окремі з цих чинників можна реально оцінити тільки під час маркетингу продукту на ринку. З цього випливає, що до початку продаж, програмне забезпечення повинне бути продемонстрованим потенційним клієнтам і, як правило, відтестоване клієнтами, і тільки після цього можна приймати рішення про його продаж.
- Встановлення програмного забезпечення на ЕОМ і дослідна експлуатація. Аналіз результатів дослідної експлуатації та порівняння отриманих результатів з очікуваними. Якщо які-небудь функції програмного забезпечення або які-небудь зобов'язання постачальника не виконуються, повинні бути початі відповідні дії, які покупець повинен обумовити в контракті.

ЛІТЕРАТУРА

1. Дегтярьова Л.М., Гроза П.М., Сомов С.В. Навчальний посібник з дисципліни «Технології розробки програмного забезпечення». – Полтава: ПолтНТУ, 2017. – 218 с.
2. Карпенко М.Ю., Манакова Н.О., Гавриленко І.О. Технології створення програмних продуктів та інформаційних систем: навч. посібник. Харків. нац. ун-т міськ. госп-ва ім. О.М. Бекетова. – Харків : ХНУМГ ім. О.М. Бекетова, 2017. – 93 с.
3. Лаврищева Е.М., Грищенко В.М. Сборочное программирование. Основы индустрии программных продуктов. – Второе изд. – К.: Наук. думка, 2009. – 371 с.
4. О.В. Алексенко. Технології програмування та створення програмних продуктів. Конспект лекцій. – Суми : СумДУ, 2013. – 133 с.