***Зміст:***

1. Сутність дисципліни інженерія програмного забезпечення.
2. Теоретичний базис інженерії програмного забезпечення.
3. Інженерія програмного забезпечення, як поєднання наукової та інженерної дисциплін.
4. Методи інженерії програмного забезпечення.
5. Сутність SWEBOK.
6. Поняття базового процесу інженерії програмного забезпечення.
7. Інструменти інженерії програмного забезпечення.
8. Інженерія програмного забезпечення, як економічна дисципліна.
9. Інженерія програмного забезпечення, як виробнича дисципліна.
10. Головні галузі інженерії програмного забезпечення.
11. Організаційні галузі інженерії програмного забезпечення.
12. Основні задачі інженерії вимог.
13. Види діяльності в інженерії вимог.
14. Основні задачі проектування програмного забезпечення.
15. Види діяльності в проектуванні програмного забезпечення.
16. Основні задачі конструювання програмного забезпечення.
17. Види діяльності в конструюванні програмного забезпечення.
18. Основні задачі тестування програмного забезпечення.
19. Види діяльності в тестуванні програмного забезпечення.
20. Основні задачі супроводу програмного забезпечення.
21. Види діяльності при супроводі програмного забезпечення.
22. Основні задачі керування конфігурацією програмного забезпечення.
23. Основні задачі керування інженерією програмного забезпечення.
24. Види діяльності в базовому процесі інженерії програмного забезпечення.
25. Основні складові базового процесу інженерії програмного забезпечення.
26. Поняття якості програмного забезпечення.
27. Види діяльності при забезпеченні якості програмного забезпечення.
28. Поняття життєвого циклу програмного забезпечення.
29. Основні процеси життєвого циклу програмного забезпечення.
30. Допоміжні процеси життєвого циклу програмного забезпечення.
31. Моделі життєвого циклу програмного забезпечення.
32. Типи моделей життєвого циклу програмного забезпечення.
33. Особливості каскадної моделі життєвого циклу програмного забезпечення.
34. Особливості інкрементної моделі життєвого циклу програмного забезпечення.
35. Особливості спіральної моделі життєвого циклу програмного забезпечення.
36. Особливості еволюційної моделі життєвого циклу програмного забезпечення.
37. Поняття вимог до програмного забезпечення.
38. Типи вимог до програмного забезпечення.
39. Діяльність при збиранні вимог до програмного забезпечення.
40. Діяльність в інженерії вимог до програмного забезпечення.
41. Призначення фіксації вимог до програмного забезпечення.
42. Призначення трасування вимог до програмного забезпечення.
43. Засоби та інструменти інженерії вимог до програмного забезпечення.
44. Основні задачі в проектуванні архітектури програмної системи.
45. Основні архітектурні моделі, що розробляються для програмної системи.
46. Компроміси щодо забезпечення вимог, при виборі архітектури програмної системи.
47. Основні задачі, що вирішуються побудовою структурної схеми системи.
48. Особливості побудови структури системи на основі моделі репозиторію.
49. Особливості побудови структури системи на основі моделі клієнт/сервер.
50. Особливості побудови структури системи на основі моделі абстрактної машини.
51. Основні моделі керування програмною системою.
52. Особливості моделі централізованого керування програмною системою.
53. Особливості моделі програмної системи, що керується подіями.
54. Особливості моделі програмної системи, що керується перериваннями.
55. Задачі модульної декомпозиції програмної системи.
56. Особливості об’єктної декомпозиції програмної системи.
57. Особливості декомпозиції програмної системи за потоками даних.
58. Особливості проблемно-залежних архітектур.
59. Особливості моделі класів систем.
60. Особливості моделі базових архітектур.

Практичні завдання

1. Побудувати діаграму варіантів використання програмної системи …
2. Побудувати діаграму активності клієнта (сервера) програмної системи …

***1. Сутність дисципліни інженерія програмного забезпечення.***

***Програмна інженерія* –** це наука побудови комп'ютерних програмних систем

на інженерній основі за методами, засобами і інструментами програмування,

сучасними стандартами процесів ЖЦ, менеджменту та керування якістю.

Особливістю виробництва нових систем є технологія їх проектування від аналізу

предметної області до утворення коду для виконання на комп'ютерах. Основа

інженерії проектування – теорія алгоритмів і програмування, теорія обчислень і

розподіленої обробки, теорія обчислювальних мереж та ін.

Програмна інженерія (ПІ) містить у собі методи і засоби керування

програмними проектами (планування робіт і регулювання ресурсів), експертне

оцінювання проміжних результатів розроблення під час процесів ЖЦ, оцінювання

ризику побудови програмної системи і досягнутої для неї якості. Ця дисципліна

використовує стандарти (наприклад, ISO/IEC 12207, ДСТУ 9126), що

регламентують процеси ЖЦ, інженерію вимог, тестування і забезпечення якості

шляхом перевірки показників на процесах ЖЦ і кінцевого продукту для їхнього

оцінювання. Інакше кажучи, в програмної інженерії подані питання теоретичної і

практичної побудови різних програмних систем для виконання задач з оброблення

інформації на комп’ютерах з метою отримання корисних даних.

Проектування у ПІ – це конструювання комп'ютерних систем методами та

засобами програмування за такими загальними кроками:

– опис вимог;

– опис специфікацій системи;

– розроблення системи;

– тестування, оцінка надійності і якості системи.

Інженерна діяльність у програмуванні на теперішній час за своєю сутністю

дуже близька до інженерної конвеєрної діяльності в промисловості, тільки тут

готовими «деталями» виступають поки ще не достатньо при промисловому

використанні багаторазових програм і систем. На сучасному етапі розвитку

базисом інженерії проектування програмних систем стали компоненти повторного

використання (reuse), яких достатньо створено у різних областях, і вони подібні до

готових деталей в промисловості. Це є фундаментальним для становлення

конвеєрного виробництва програмних продуктів, як продуктів промислово-

технічного призначення.

*Програмна інженерія* (SE) **–** це

1) застосування систематичного, дисциплінованого та вимірюваного підходу

до розроблення, експлуатації і супроводження програмного забезпечення (ПЗ) із

застосуванням інженерних методів до розроблення ПЗ,

2) навчальна дисципліна, що вивчає вказані вище підходи.

Ця дефініція обмежує суть не лише поняття предмета ПІ, а й об’єкта,

розглядаючи як такий лише програмне забезпечення, а не програмні проекти,

процеси та методи розробки програмних продуктів тощо.

***2.Теоретичний базис інженерії програмного забезпечення.***

Тому необхідно

визначити ПІ і її об’єкти у більш широкому розумінні, доповнюючи аспектами, що

характеризують ПІ як наукову й інженерну дисципліни.

Відомо, що будь-яка наука – це система перевірених практикою знань, які

відображають загальні питання, поняття і закономірності їх розвитку. Вона

встановлює зв’язки з іншими науками і впливає на їх розвиток. Так, програмна

інженерія інтегрує в собі принципи математики та головні положення

фундаментальних наук, а саме, теорії алгоритмів, математичної логіки, теорії

керування, теорії множин, доведення тощо (рис.1.1).

Рис. 1.1. Теоретичний фундамент програмної інженерії

Ці науки створюють теоретичний базис програмної інженерії, необхідний для

побудови абстракцій програм за їх базовими поняттями та принципами, що

перелічені нижче за кожною з фундаментальних наук базису:

– у теорії алгоритмів – нормальні алгоритми, обчислювальні функції,

машина Тюрінга, алгоритмічні алгебри, граф-схеми, моделі алгоритмів і програм

тощо;

– у математичній логіці – логічні числення і логіко-алгебраїчний апарат

специфікації програм;

– у теорії керування – принципи, методи та загальні закони планування і

керування процесами отримання й оброблення інформації в кібернетичних і

управлінських системах;

– у теорії доведення – математичне доведення за аксіомами і твердженнями

програм, вивід теорем, обґрунтування суперечності й алгоритмічно невирішених

проблем, а також теорія верифікації програм, теорія надійності ПЗ;

******

Рис. 1.1. Теоретичний фундамент програмної інженерії

***3 . Інженерія програмного забезпечення, як поєднання наукової та інженерної дисциплін.***

різних семінарах і лекціях. Сутність кожної з дисциплін така:

*– наукова дисципліна* визначена як сукупність формальних методів

специфікації, доведення та верифікації програмних об’єктів, методів їх об’єднання,

теоретичних і прикладних методів програмування та теоретичних моделей

надійності програм та методів їхнього застосування;

*– інженерна дисципліна* сформульована як сукупність технологічних засобів і

методів проектування ПС за фундаментальними моделями ЖЦ, положеннями

сучасного стандарту із процесів ЖЦ, техніки аналізу предметної області,

формулювання вимог з розробленням за ними відповідного вихідного коду, його

супроводу та внесення до нього різного роду змін, включаючи ті, що забезпечують

перенесення програмного продукту на інші комп’ютерні платформи;

*Програмна інженерія* **–** розділ комп’ютерної науки, який

вивчає методи і засоби побудови комп’ютерних програм; відображає

закономірності розвитку та узагальнює накопичений досвід програмування; оперує

об’єктами (модулями, компонентами, програмними аспектами тощо) та визначає

автоматизовані операції щодо їх виробництва та ін.



*Програмна інженерія як наукова дисципліна* охоплює *теоретичні, формальні*

методи та відповідні засоби побудови складних програмних об’єктів. *Побудова* має

на меті аналіз предметної області, що автоматизується, і продукування

результуючого коду для виконання на комп'ютері. Інтегровані в програмну

інженерію фундаментальні науки, згадані вище (на рис.1.1), а також наука

програмування, становлять її загальну теоретичну основу, яка надає базові поняття,

об’єкти і формальні механізми, необхідні для надання програмним продуктам

загальних властивостей та специфічних рис відповідно до встановлених до них

вимог. ПІ як наука містить у собі:

1) основні поняття і об'єкти;

2) теорію програмування і методи керування виготовленням продукту;

3) засоби і інструменти процесів розроблення продукту.

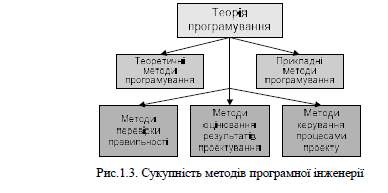
**1. Основні поняття** програмної інженерії – це дані і їх структури (прості і

складні), функції і композиції, базові об’єкти (модуль, компонент, каркас,

контейнер, компонент повторного використання (КПВ) тощо) і цільові об’єкти

(програмне забезпечення, програмна система, сімейство систем, програмний

проект, складні програмні застосування тощо).



*Програмна інженерія як інженерна дисципліна* (або власне інженерія) – це

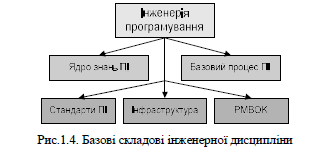
сукупність прийомів виконання діяльності, пов’язаної з виготовленням

програмного продукту для різних видів цільових об’єктів із застосуванням методів,

засобів і інструментів наукової складової програмної інженерії [8–10]. Основу

інженерії становлять наступні базові елементи процесу виготовлення програмного

продукту (рис. 1.4):



визначень методів і засобів розробки та керування програмними проектами, які

можуть застосовуватися в інженерії програмування;

2) базовий процес ПІ, як стрижень процесної діяльності в організації-

розробнику програмного продукту;

3) стандарти, як набір регламентованих правил конструювання проміжних

артефактів у процесах ЖЦ;

4) інфраструктура – умови середовища та методичне забезпечення базового

процесу ПІ і підтримка дій його виконавців, що займаються виробництвом

програмного продукту;

5) менеджмент проекту (РМВОК) – ядро знань з керування промисловими

проектами – набір стандартних процесів, а також принципів і методів планування і

контролювання роботами у проекті [11];

6) засоби та інструменти розробки програмних продуктів.

З інженерної точки зору в програмній інженерії розв’язуються задачі

виготовлення ПП, подані як технологічні процеси формування вимог, проектуванняі супроводу продукту, а також перевірки операцій базового процесу щодо

правильності виконання різних функціональних задач та виконання робіт за

проектом у заданий замовником строк.

***4. Методи інженерії програмного забезпечення.***

**Методи інженерії ПЗ** – це евристичні методи (heuristic methods), формальні

методи (formal methods) і методи прототипування (prototyping methods).

*Евристичні методи* містять у собі: структурні методи, засновані на

функціональній парадигмі; методи, орієнтовані на структури даних, якими

маніпулює ПЗ; об’єктно-орієнтовані методи, що розглядають предметну область як

колекцію об'єктів; методи, орієнтовані на конкретну область застосування,

наприклад, на системи реального часу, безпеки та ін.

*Формальні методи* засновані на формальних специфікаціях, аналізі,

доведенні і верифікації програм. Специфікація записується мовою, синтаксис і

семантика якої визначені формально і засновані на математичних концепціях

(алгебрі, теорії множин, логіці). Розрізняються наступні категорії формальних

методів:

*– мови і нотації специфікації* (specification languages and notations),

орієнтовані на модель, властивості і поведінку;

*– уточнення специфікації* (refinement specification) шляхом трансформації в

кінцевий результат, близький до кінцевого програмного продукту, що виконується;

*– методи верифікації/доведення* (verification/proving properties), що

використовують твердження (теореми), перед- і постумови, формально описуються

і застосовуються для встановлення правильності специфікації програм.

Методи доведення застосовувалися в основному в теоретичних

експериментах. Понад 25 років їх застосування було обмежено через

трудомісткість і економічну невигідність. У 2005 р. проблема верифікації знову

набула актуальності у запропонованому новому міжнародному проекті «Цілісний

автоматизований набір інструментів для перевірки коректності ПС» (Т. Хоар,

«Открытые системы», 2006, № 6), який поставив наступні перспективні задачі:

– розробка єдиної теорії побудови й аналізу програм;

– побудова багатостороннього інтегрованого набору інструментів верифікації

на усіх виробничих процесах – розроблення формальних специфікацій, їх

доведення і перевірка правильності, генерація програм і тестових прикладів,

уточнення, аналіз і оцінка;

– створення репозитарію формальних специфікацій, верифікованих

програмних об'єктів різних типів і видів.

Формальні методи верифікації будуть охоплювати всі аспекти створення і

перевірки правильності програм. Це приведе до створення потужної верифікованої

виробничої основи і сприятиме значному зменшенню помилок у ПЗ (стосовно

доведення і верифікації див. розділ 6).

*Методи прототипування (Prototyping Methods) засновані на використанні*

*прототипу ПЗ для моделювання на ньому завдань нової системи і базуються на:*

*– стилях прототипування, що уособлюють тривалість використання*

*прототипів, наприклад, стиль створення тимчасово використовуваних прототипів*

*(throw away),*

*– моделях еволюційного прототипування – перетворення прототипу в*

*кінцевий продукт і розроблення специфікацій, відповідно до якої він виконується;*

*– техніках оцінки/дослідження (evaluation) результатів прототипування.*

***5. Сутність SWEBOK***

Спеціально створений комітет фахівців з інформатики при ACM і IEEE

Computer Society сформував базове ядро знань SWEBOK (Software Engineering body

of Knowledge – 2001р.), у якому в концентрованому вигляді подав концептуальний

зміст десятьох базових областей (knowledge areas) та дефініції програмної інженерії

(ПІ), зокрема [1-3]. У ядрі знань SWEBOK наведене таке визначення програмної

інженерії (воно відповідає глосарію IEEE)

**Ядро знань SWEBOK** – стислий опис концептуальних основ програмної

інженерії. Структурно ділиться на 10 розділів (knowledge areas), які умовно можна

розкласти за двома категоріями: проектування продукту і інженерна діяльність.

Перша категорія – це методи і засоби розробки (формування вимог, проектування,

конструювання, тестування, супровід), друга категорія – методи керування

проектом, конфігурацією і якістю та базовим процесом організації-розробника

(детальніше див. у п.1.2).

Методи ядра знань програмної інженерії менеджер проекту зіставляє з

відповідними стандартними процесами ЖЦ, виконання яких забезпечує послідовне

розроблення програмного продукту через наповлення базового процесу програмної

інженерії методами з ядра знань SWEBOK, а також задачами і діями стандартного

ЖЦ, що обумовлює його застосовність до потреб конкретної організації-

розробника щодо певної регламентованої послідовності розробки і супроводу

програмного продукту. Все це створює технологічний базис інженерії виготовлення

конкретного продукту (або низки однотипних продуктів) в організації. На

початкових стадіях розробки виконуються процеси визначення вимог до продукту,

вироблення проектних рішень і каркасу (абстрактної архітектури) майбутнього

продукту. На основі вимог і каркасу розробляються або вибираються готові прості

об’єкти для «наповлення» цього каркасу змістом для подальшого його доведення

до стану готового продукту.

***6. Поняття базового процесу інженерії програмного забезпечення.***

**Базовий процес** (БП) *–* є метарівнем для забезпечення «процесного

продукування» продукту. Він містить у собі опис понять щодо оснастки, організаційної структури колективу розробників та методології оцінки,

вимірювання, керування змінами та вдосконалення самого процесу. В цілому

базовий процес складається з множини логічно пов’язаних видів інженерної

діяльності організації-розробника та набору засобів і інструментів щодо

виготовлення програмного продукту.

***7. Інструменти інженерії програмного забезпечення.***

**Інструменти інженерії ПЗ** забезпечують автоматизовану підтримку процесів

розроблення ПЗ і містять у собі множину інструментів, що охоплюють усі процеси

ЖЦ.

*Інструменти роботи з вимогами (Software Requirements Tools)* – це:

– інструменти розробки (Requirement Development) і керування вимогами

(Requirement Management), орієнтовані на аналіз, збирання, специфікування і

перевірку вимог;

– інструменти трасування вимог (Requirement traceability tools) є невід'ємною

частиною роботи з вимогами, їх функціональний зміст залежить від складності

проектів і рівня зрілості процесів.

*Інструменти проектування (Software Design Tools) –* це інструменти для

створення ПЗ із застосуванням базових нотацій (структурної SADT/IDEF,

моделювання UML і т.п.).

*Інструменти конструювання ПЗ (Software Construction Tools) –* це

інструменти для трансляції і об’єднання програм. До них належать:

– редактори програм (program editors) і програми редагування загального

призначення;

– компілятори і генератори коду (compilers and code generators) як самостійні

засоби об'єднання програмних компонентів в інтегрованому середовищі для

одержання вихідного продукту з використанням препроцесорів, складальників,

завантажників і ін.;

– інтерпретатори (interpreters), які забезпечують контрольоване виконання

програм за їх описом. Намітилася тенденція злиття інтерпретаторів і компіляторів

(наприклад, Java, в .NET);

– відлагоджувачі (debuggers), призначені для перевірки правильності опису

вихідних програм і усунення помилок;

– інтегроване середовище розробки (IDE – integrated development

environment) та бібліотеки компонентів (libraries components), що є утворюють

середовище виконання процесу розроблення ПС;

– програмні платформи (Java, J2EE і Microsoft .NET) і платформи для

розподілених обчислень (CORBA і WebServices, тощо).

*Інструменти тестування (Software Testing Tools) –* це:

– генератори тестів (test generators), що допомагають у розробці сценаріїв

тестування;

– засоби виконання тестів (test execution frameworks), які забезпечують

виконання тестових сценаріїв і відслідковують поведінку об'єктів тестування;

– інструменти оцінки тестів (test evaluation tools), які підтримують

оцінювання результатів виконання тестів і ступеня відповідності поведінки

тестованого об'єкта очікуваній поведінки;

– засоби керування тестами (test management tools), які забезпечують

інженерне керування процесом тестування ПЗ;

– інструменти аналізу продуктивності (performance analysis tools), кількісної

її оцінки та оцінки поводження програм у процесі виконання.

*Інструменти супроводу (Software Maintenance Tools)* містять у собі:

– інструменти полегшення розуміння (comprehension tools) програм,

наприклад, різні засоби візуалізації;

– інструменти реінженерії (reengineering tools) підтримують діяльність з

перетворення програм і зворотної інженерії (reverse engineering) для відновлення

(артефактів, специфікації, архітектури) застарілого ПЗ або генерації нового

продукту.

*Інструменти конфігураційного керування (Software Configuration Management*

*Tools)* – це:

– інструменти відстеження (tracking) дефектів;

– інструменти керування версіями;

– інструменти керування складанням, випуском версії (конфігурації)

продукту та його інсталяції.

*Інструменти керування інженерною діяльністю (Software Engineering*

*Management Tools)* підрозділяються на:

– інструменти планування і відстеження ходу проектів, кількісної оцінки

зусиль і вартості робіт у проекті (наприклад, Microsoft Project 2003);

– інструменти керування ризиками, які використовуються для ідентифікації,

моніторингу ризиків і оцінки нанесеного ушкодження;

– інструменти кількісної оцінки властивостей ПЗ шляхом вимірювань і

розрахунків остаточного значення надійності і якості.

*Інструменти підтримки процесів (Software Engineering Process Tools)*

розділені на:

– інструменти моделювання та опису моделей ПЗ (наприклад, UML і його

інструменти);

– інструменти керування програмними проектами (наприклад, Microsoft

Project);

– інструменти керування конфігурацією для підтримки версій і всіх

артефактів проекту.

*Інструменти забезпечення якості (Software Quality Tools)* діляться на дві

категорій:

– інструменти інспектування для підтримки перегляду (review) і аудиту;

– інструменти статичного аналізу артефактів, даних, потоків робіт і перевірки

їх властивостей на відповідність показникам.

*Додаткові аспекти інструментального забезпечення (Miscellaneous Tool*

*Issues)* стосуються:

– техніки інтеграції інструментів (платформ, представлень, процесів, даних)

для їх природного сполучення в інтегрованому середовищі;

– метаінструментів для генерації інших інструментів для ПЗ;

– оцінки інструментів при їх еволюції

***8. Інженерія програмного забезпечення, як економічна дисципліна.***

*економічна дисципліна* сформульована як сукупність методів експертного,

якісного і кількісного оцінювання проміжних об’єктів ЖЦ, а також економічних

методів розрахунків часу, обсягу і вартості виготовлення програмних продуктів, що

поставляються на ринок;

Економіка ПІ є самостійною дисципліною зі своєю теорією і практикою

оцінювання вартісних, часових і експертних *показників* щодо складання контрактів

на створення ПП, прийняття проектних рішень, подання вимог, розроблення

архітектури тощо, визначення *ризиків* проектування за заданими ресурсами,

проведення розрахунків за роботи виконавців та отриману якість ПП. Ця

дисципліна є найбільш розвинутою з точки зору методів економічних розрахунків у

ПІ, а саме, наявних методологій прогнозування розміру ПП (FРA– Function Points

Analyses, Feature Points, Mark-H Function Points, 3D Function Points тощо),

оцінювання витрат на розроблення ПП за допомогою сімейства моделей COCOMO

або інших математичних моделей (Angel, Slim, Seer тощо) [4].

При формуванні цієї дисципліни необхідно використати фундаментальні

економічні методи, пов’язані з принципами розподілу робіт у складних системах,

методи розрахунків вартості окремих частин систем залежно від розміру і системи

у цілому, існуючі стандарти з оцінювання ПП тощо. Систематизований і науково

обґрунтований курс економічної дисципліни ПІ компенсує відсутність відповідних

посібників і підручників для навчання спеціалістів, зайнятих у виробництві ПП.

**Таким чином**, наукові, інженерні, виробничі напрями ПІ, дисципліни

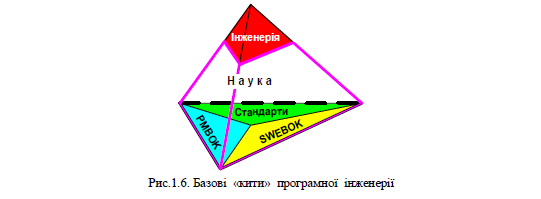
керування і економіки, а також SWEBOK, СТАНДАРТИ, PMBOK є головними

складовими програмної інженерії. Вони зв'язані між собою процесами ЖЦ,

методами проектування і керування розробленням програмних проектів. Ключеві

моменти з питань вироблення програмних продуктів на процесній і інженерній

основі відображено змістовним багатогранником фундаменту ПІ (рис.1.6).

******

науки, інженерії, економіки та керування виробництвом ПП. Надано необхідні

аргументи і обґрунтування визначень як цільових об’єктів проектування, так і

програмної інженерії.

***9. Інженерія програмного забезпечення, як виробнича дисципліна.***

*виробнича дисципліна* – це сучасні промислові технологічні прийоми

виробництва прикладних систем, сімейств систем з застосуванням готових

програмних ресурсів, включаючи компоненти повторного використання,

накопичених у сучасних інформаційних сховищах, одиночні готові програми

розв’язку деяких задач, сервісні, агентні артефакти тощо. Для забезпечення їх

правильності виконуються методи верифікації, тестування і оцінки за отриманими

на них показниками якості програмного продукту.

**Програмна інженерія як виробнича дисципліна**

Загальне призначення програмної інженерії – практичне виготовлення

комп'ютерних програм, систем і інструментів із застосуванням теоретичних і

інженерних методів ПІ.

Головна особливість практики у майбутньому – це використання

розроблених готових програм і інформаційних ресурсів Інтернету. Доступ до них

може здійснити будь-який користувач і одержувати безкоштовно або на

комерційній основі готовий програмний ресурс як сервіс. Він може бути

одноразово використаний для розв’язання відповідної задачі, або як окрема

програма постійного і багаторазового застосування в деякому домені. Сьогодні

сформувалися три інженерні підходи до застосування таких готових ресурсів:

reusing engineering, application engineering, domain engineering. Вони

використовують як готові ресурси повторно використовувані компоненти, засоби і

системи. Застосування готових ресурсів, як багаторазово використаного готового

продукту, дає значну економію при виробництві з них нових програмних систем і

сімейств систем. Усі види компонентів, а саме, КПВ зберігаються в сховищах

проекту – репозитарії [8, 9].

**Інженерія КПВ** – це систематична і цілеспрямована діяльність з вибору

реалізованих і поданих у репозитарії КПВ. Система проектується знизу вгору.

Спочатку створюється загальна структура – каркас продукту, далі дається його

опис і за цим описом готові компоненти і КПВ інтегруються в систему.

**Інженерія застосувань** також базується на багаторазовому використанні

КПВ і готових програм. Проектування одиночних, тобто унікальних програмних

застосувань – це інженерія програмування з готових КПВ. Процес побудови

починається з аналізу предметної області, визначення концептуальної моделі,

розроблення проектних рішень і проведення композиції компонентів з

використанням шаблонів або каркасів.

**Інженерія ПрО** – набір засобів, інструментів, готових ресурсів і базового

процесу побудови з них систем сімейства (домену) на основі моделі, що містить у

собі загальні і змінювані характеристики представників сімейства. Вибрані КПВ

або одиночні для застосування вбудовуються в модель домену. Технологія

розробки сімейства планується для створення як моделі цього домену, так і членів

сімейства. Вона містить у собі процеси аналізу, проектування і вбудовування КПВ

в окремі члени сімейства. За своїм характером ця інженерія наближена до

конвеєрного виробництва продуктів з готових ресурсів. Керування нею складається

з планування і розподілу робіт за кожним учасником процесу створення домену і

контролю їх виконання в заданий строк і з встановленим рівнем якості.

**Лінійки продуктів.** Технологічні прийоми виробництва програмних

продуктів з готових компонентів і програмних систем втілені в так звані *лінійки*

*продуктів,* що відповідають конвеєрному виробництву програмних продуктів на

ринковій основі. Практичні інструменти цього виробництва – SWEBOK, PMBOK істандарти, а також сучасні інструментально-технологічні системи і середовища з набором необхідних інструментів та засобів.

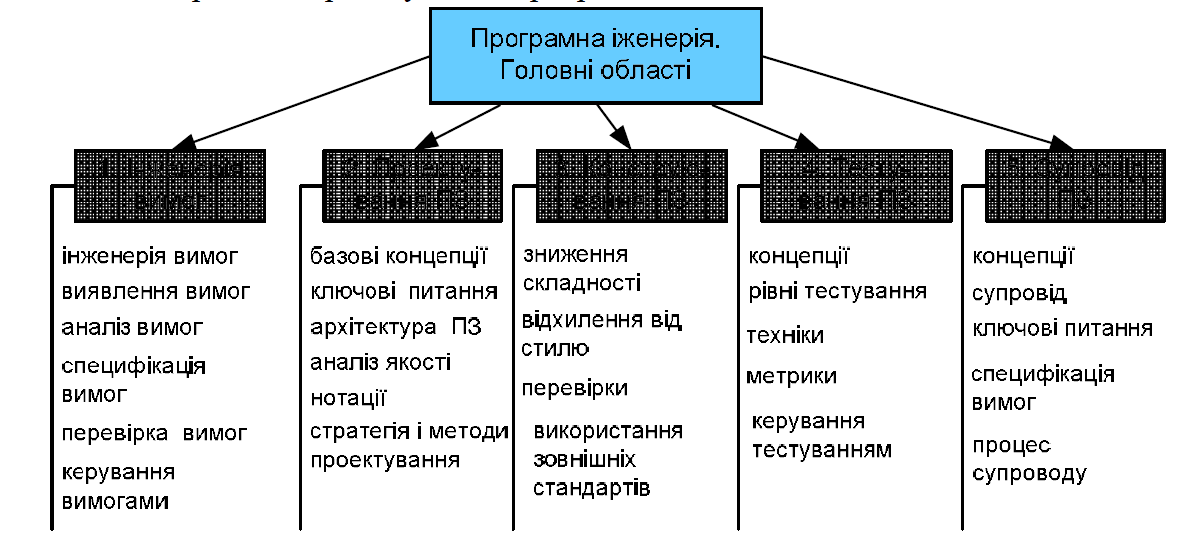
***10. Головні галузі інженерії програмного забезпечення***

Для подання понятійного апарату областей знань SWEBOK проведемо

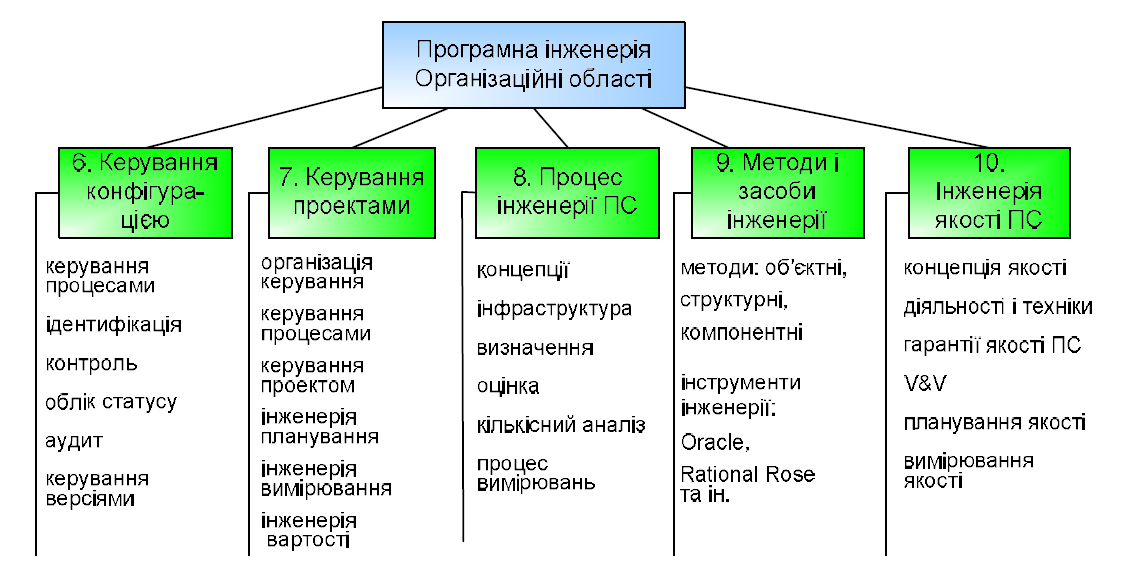
умовне розділення областей на *головні* (п'ять областей для розроблення ПС, рис.

1.7) і допоміжні *організаційні області* (п'ять областей, що забезпечують інженерію

керування розробкою ПС, рис.1.8).

******

***11.Організаційні галузі інженерії програмного забезпечення.***

******

***12 Основні задачі інженерії вимог***

*Вимоги* до ПЗ – сукупність властивостей, які повинно мати ПЗ. Призначені

для адекватного визначення функцій, умов і обмежень виконання ПЗ, а також

обсягів даних, технічного забезпечення і середовища його виконання.

Вимоги відбивають потреби людей (замовників, користувачів, розробників),

зацікавлених у створенні ПЗ. Замовник і розробник спільно виявляють вимоги,

аналізують, переглядають, визначають необхідні обмеження і умови, а також

описують їх. Розрізняють вимоги до продукту і до процесу, а також функціональні,

не функціональні і системні вимоги. Вимоги до продукту і до процесу визначають

умови виконання і режими роботи ПЗ в операційному середовищі, обмеження на

структуру і пам'ять комп'ютерів та принципи взаємодії програм.

Функціональні вимоги визначають призначення і функції системи, а не

функціональні – умови стосовно виконання ПЗ, його переносності і доступу до

даних. Системні вимоги описують вимоги до програмної системи, яка складається з

взаємозалежних програмних і апаратних підсистем і різних застосувань. Вимоги

можуть бути кількісні (наприклад, кількість оброблених запитів на секунду,

середній показник помилок і т.п.). Значна частина вимог стосується атрибутів

якості: безвідмовність, надійність і ін., а також захисту і безпеки як ПЗ, так і даних.\_

***13. Види діяльності в інженерії вимог.***

Область знань «Вимоги до ПЗ (Software Requirements)» складається з таких

розділів:

– інженерія вимог (Requirement Engineering),

– виявлення вимог (Requirement Elicitation),

– аналіз вимог (Requirement Analysis),

– специфікація вимог (Requirement Specification).

– валідація вимог (Requirement validation),

– керування вимогами (Requirement Management).

***14. Основні задачі проектування програмного забезпечення***

**Проектування ПЗ** – це процес визначення архітектури, набору компонентів,

їх інтерфейсів, інших характеристик системи і кінцевого складу програмного

продукту.

Область знань «Проектування ПЗ (Software Design)» складається з таких

розділів:

– базові концепції проектування ПЗ (Software Design Basic Concepts),

– ключові питання проектування ПЗ (Key Issue in Software Design),

– структура й архітектура ПЗ (Software Structure and Architecture),

– аналіз і оцінка якості проектування ПЗ (Software Design Quality Analysis and

Evaluation),

– нотації проектування ПЗ (Software Design Notations),

– стратегія і методи проектування ПЗ (Software Design Strategies and

Methods).

***15.Види діяльності в проектуванні програмного забезпечення***

**Базова концепція проектування ПЗ** – це методологія проектування

архітектури за допомогою різних методів (об'єктного, компонентного й ін.),

процеси ЖЦ (стандарт ISO/IEC 12207) і техніки – декомпозиція, абстракція,

інкапсуляція й ін. На початкових стадіях проектування предметна область

декомпозується на окремі об'єкти (при об’єктно-орієнтованому проектуванні) або

на компоненти (при компонентному проектуванні). Для подання архітектури

програмного забезпечення вибираються відповідні артефакти (нотації, діаграми,

блок-схеми і методи).

**Ключові питання проектування –** це декомпозиція програм на

функціональні компоненти для незалежного і одночасного їхнього виконання,

розподіл компонентів у середовищі функціонування і їх взаємодія між собою,

забезпечення якості і живучості системи й ін.

**Проектування архітектури ПЗ** проводиться архітектурним стилем,

заснованим на визначенні основних елементів структури – підсистем, компонентів,

об'єктів і зв'язків між ними.

*Архітектура проекту* – високорівневе подання структури системи і

специфікація її компонентів. Архітектура визначає логіку системи через окремі

компоненти системи настільки детально, наскільки це необхідно для написання

коду, а також визначає зв'язки між компонентами. Існують і інші види подання

структур, засновані на проектуванні зразків, шаблонів, сімейств програм і каркасів

програм.\_\_

**Аналіз і оцінка якості проектування ПЗ** – це заходи щодо аналізу

сформульованих у вимогах атрибутів якості, функцій, структури ПЗ, з перевірки

якості результатів проектування за допомогою метрик (функціональних,

структурних і ін.) і методів моделювання і прототипування.

**Нотації проектування** дозволяють представити опис об'єкта (елемента) ПЗ і

його структуру, а також поведінку системи за цим об’єктом. Існує два типи

нотацій: структурна, поведінкова, та множина їх різних представлень.

**Стратегія і методи проектування ПЗ.** До стратегій відносять: проектування

вгору, вниз, абстрагування, використання каркасів і ін. Методи є функціонально-

орієнтовані, структурні, які базуються на структурному аналізі, структурних

картах, діаграмах потоків даних й ін. Вони орієнтовані на ідентифікацію функцій і

їх уточнення знизу-вгору, після цього уточнюються діаграми потоків даних і

проводиться опис процесів.

В об’єктно-орієнтованому проектуванні ключову роль відіграє спадкування,

поліморфізм й інкапсуляція, а також абстрактні структури даних і відображення

об'єктів. Підходи, орієнтовані на структури даних, базуються на методі Джексона і

використовуються для подання вхідних і вихідних даних структурними діаграмами.

Метод UML призначений для опису сценаріїв роботи проекту у наочному

діаграмному вигляді. Компонентне проектування ґрунтується на використанні

готових компонентів (reuse) з визначеними інтерфейсами і їх інтеграції в

конфігурацію, як основи розгортання компонентної системи для її функціонування

в операційному середовищі.

***16.Основні задачі конструювання програмного забезпечення.***

Термін конструювання програмного забезпечення (software construction) описує детальне створення робочої програмної системи за допомогою комбінації кодування, верифікації (перевірки), модульного тестування (unit testing), інтеграційного тестування та відлагодження.

Деякі конкретні задачі, що виникають в процесі розробки ПЗ, пов’язані з конструюванням:

* перевірка виконання умов, необхідних для успішного конструювання;
* визначення способів подальшого тестування коду;
* проектування та написання класів та методів;
* створення та присвоєння імен змінних та іменованих констант;
* вибір управляючих структур та організації блоків команд;
* блочне тестування, інтеграційне тестування і відлагодження власного коду;
* взаємний огляд коду та низькорівневих програмних структур членами групи;
* "шліфування" коду шляхом його ретельного форматування та коментування;
* інтеграція програмних компонентів, створених окремо;
* оптимізація коду, яка направлена на підвищення його швидкодії і зниження міри використання ресурсів.

***17. Види діяльності в конструюванні програмного забезпечення***

– зниження складності (Reduction in Complexity),

– попередження відхилень від стилю (Anticipation of Diversity

– структуризація перевірок (Structuring for Validation),

– використання стандартів (Use of External Standards).

**Зниження складності** – це мінімізація, зменшення і локалізація складності конструювання.

*Мінімізація складності* – це використання у процесі конструювання простих елементів, а також рекомендацій стандартів.

*Зменшення складності* в конструюванні ПЗ досягається шляхом створення простого коду, що легко читається і спрощує тестування.

*Локалізація складності* –призначена для внесення змін, пов'язаних з виявленими помилками в коді, або коли джерелом помилок є середовище, у якому виконується код.

**Попередження відхилень від стилю.**

*Лінгвістичний стиль* призначений для конструювання нескладних конструкцій і приводиться до вигляду традиційних функцій і процедур.

*Формальний стиль* використовується для точного й однозначного визначення компонентів системи, мінімальної кількості помилок.

*Візуальний* стиль – дозволяє представляти елемент конструювання у наочному вигляді. Візуальна мова проектування UML надає розробнику набір діаграм для подання статичної і динамічної структур ПЗ.

**Структуризація перевірок** припускає, що побудова ПС структурована таким чином, що спрощується пошук помилок, дефектів і різних збоїв у процесі перевірок як на стадії незалежного тестування, так і в процесі експлуатації. Структуризації перевірок сприяють огляд, інспектування, спільний перегляд, модульне тестування із застосуванням автоматизованих засобів тестування й ін.

**Використання зовнішніх стандартів**. При конструюванні має бути визначений достатній набір стандартів . До таких стандартів відносять: мови програмування (Java, Ада 95, С++ і ін.), інтерфейси мов програмування (МП) і прикладні інтерфейси платформ Windows (COM, DCOM), CORBA і ін. При конструюванні використовують стандарти мов опису даних (XML, SQL і ін.), засобів комунікації (COM, CORBA і ін.), інтерфейсних мов (POSIX, IDL, APL), UML і ін.

***18. Основні задачі тестування програмного забезпечення***

*Тестування ПЗ* – це процес перевірки готової програми в статиці (перегляди, інспекції, налагодження вихідного коду) і в динаміці (прогін на наборі тестових даних) з метою перевірки різних шляхів виконання програми і порівняння отриманих результатів із заздалегідь заданими. Існує дві форми перевірки коду – модульна й інтеграційна. Спочатку використовують стандарти (IEEE 829:1996 і IEEE 1008:1987) з перевірки і тестування модулів. Потім проводиться інтеграційне тестування модулів системи і їх інтерфейсів у динаміці виконання. Під час різних видів перевірок збираються дані про помилки, дефекти, відмови тощо і оформляється відповідна документація (таблиці типів помилок, частоти і часу виявлення відмов і ін.). Зібрані дані використовуються при оцінюванні характеристик якості готового ПЗ, наприклад, надійності.

Тестування визначається як процес перевірки правильності програми в динаміці її виконання за тестовими даними. При тестуванні виявляються недоліки: відмови (faults) і дефекти (defects) як причини порушення роботи програми, збої (failures) як небажані ситуації, помилки (errors) як наслідки збоїв і ін. Базове поняття тестування – тест, що виконується в заданих умовах і за наборами даних. Тестування вважається успішним, якщо знайдено дефект або помилка, і вони відразу усуваються. Ступінь тестованості визначається критерієм покриття системи тестами, перевірки всіх можливих шляхів виконання програм і імовірності припущення стосовно того, що може з'явитися збій або помилкова ситуація в системі.

***19.Види діяльності в тестуванні програмного забезпечення***

– рівні тестування (Test Levels),

– техніки тестування (Test Techniques),

– метрики тестування (Test Related Measures),

– керування процесом тестування (Managing the Test Process).

**Рівні тестування**:

*– тестування окремих елементів*

*– інтеграційне тестування* орієнтоване на перевірку зв'язків і взаємодії компонентів

*– тестування системи* – це перевірка правильності функціонування системи.

**Техніки тестування** базуються на певних теоретичних і практичних

положеннях щодо проектування , а також на таких даних:

– інформація про структуру ПЗ або системи в документації («біла скринька»*)*;

– підбір тестових наборів даних для перевірки правильності роботи

компонентів і системи в цілому без знання їх структури («чорна скринька»);

– аналіз граничних значень, таблиць прийняття рішень, потоків даних,

статистики відмов і ін.;

– блок-схеми побудови програм і складання наборів тестів для покриття

системи цими тестами;

– виявлені і зафіксовані в таблицях системи дефекти, перед- і постумови

виконання, структурні характеристики системи (кількість модулів, обсяг даних

тощо).

**Метрики тестування**. Метрики тестування – це вимірювання процесу планування, проектування і тестування, а також результатів тестування на основі таксономії відмов і дефектів, покриття границь тестування, перевірки потоків даних і ін.

**Керування тестуванням**:

– планування процесу тестування (складання планів, тестів, наборів даних) і оцінювання показників якості готового продукту;

– проведення тестування компонентів повторного використання і патернів як основних об'єктів складання ПЗ;

– генерація необхідних тестових сценаріїв, що відповідають середовищу виконання ПЗ;

– верифікація правильності реалізації системи і валідація реалізації вимог до ПЗ;

– збирання даних про відмови, помилки і виявлені непередбачені ситуації при виконанні програмного продукту;

– підготовка звітів за результатами тестування й оцінка характеристик системи. Відповідно до стандарту ISO/IEC 12207 тестування ПЗ розглядається як невід'ємна частина ЖЦ.

***20.Основні задачі супроводу програмного забезпечення***

**Супровід програмного забезпечення** - процес покращення, оптимізації та усунення дефектів [програмного забезпечення](http://znaimo.com.ua/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) (ПО) після передачі в експлуатацію. Супровід ПЗ - це одна з фаз життєвого циклу програмного забезпечення, наступна за фазою передавання ПО в експлуатацію. У ході супроводження в програму вносяться зміни, з тим, щоб виправити виявлені в процесі використання дефекти і недоробки, а також для додавання нової функціональності, з метою підвищити зручність використання ( [юзабіліті](http://znaimo.com.ua/%D0%AE%D0%B7%D0%B0%D0%B1%D1%96%D0%BB%D1%96%D1%82%D1%96)) та застосовність ПО.

В [моделі водоспаду](http://znaimo.com.ua/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B2%D0%BE%D0%B4%D0%BE%D1%81%D0%BF%D0%B0%D0%B4%D1%83), супровід ПЗ виділяється в окрему фазу циклу розробки. У спіральної моделі, що виникла в ході розвитку [об'єктно-орієнтованого програмування](http://znaimo.com.ua/%D0%9E%D0%B1%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), супровід не виділяється як окремий етап. Тим не менш, ця діяльність займає значне місце, враховуючи той факт, що зазвичай близько 2/3 життєвого циклу програмних систем займає супровід.

**Процес супроводження** містить у собі моделі процесу супроводу і планування діяльності людей, що проводять запуск ПЗ, перевірку правильності його виконання і внесення в нього змін. Цей процес згідно з стандартом ISO/IEC 14764 проводиться шляхом:

– коригування, тобто зміни продукту для усунення виявлених помилок або нереалізованих задач;

– адаптації, тобто настроювання продукту в умовах експлуатації, що змінилися, або в новому середовищі виконання;

– поліпшення, тобто еволюційної зміни продукту для підвищення продуктивності або рівня супроводу;

– перевірки ПЗ, пошуку і виправлення помилок при експлуатації системи.

***21.Види діяльності при супроводі програмного забезпечення***

**Техніка супроводу** (цей розділ називають також еволюцією ПЗ). Відомий фахівець в області ПЗ Дж. Леман (1970 р.) запропонував розглядати супровід як еволюційну розробку програмних систем, оскільки здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Внаслідок змін система стає більш складною і погано керованою. У зв'язку з цим виникає проблема зменшення її складності. До технологій еволюції ПЗ відносять реінженерію, реверсну інженерію і рефакторинг.

*Реінженерія –* це удосконалення застарілого ПЗ шляхом його реорганізації або реструктуризації, а також перепрограмування окремих елементів або настроювання параметрів на іншу платформу, середовище виконання зі збереженням зручності його супроводу.

*Реверсна інженерія* полягає у відновленні специфікації (графів викликів, потоків даних і ін.) за отриманим кодом системи для її аналізу на більш високому рівні. Відновлюється ідентифікація компонентів і зв'язків між ними для забезпечення перепрограмування системи на нову платформу. Найчастіше реверсна інженерія застосовується після того, як у код ПЗ було внесено багато змін і воно стало некерованим або змінилася платформа комп'ютера.

*Рефакторинг –* це реорганізація коду для поліпшення характеристик і показників якості об’єктно-орієнтованих і компонентних програм без зміни їх поведінки. Цей процес реалізується шляхом поступової зміни окремих операцій над текстами, інтерфейсами, середовищем програмування і виконання ПЗ, а також настроювання або внесення змін в інструментальні засоби підтримки ПЗ. Якщо при зміні зберігається формат існуючої системи, то рефакторинг – один з варіантів реверсної інженерії.

***22.Основні задачі керування конфігурацією програмного забезпечення.***

Керування конфігурацією – це ідентифікація компонентів системи, визначення функціональних, фізичних характеристик системи, апаратного і програмного забезпечення для контролю виконання, внесення змін і трасування конфігурації. Процес керування визначено як один з допоміжних процесів ЖЦ (ISO/IEC 12207), виконуваний технічним і адміністративним менеджментом проекту. При цьому складаються звіти про зміни, внесені у конфігурацію, і ступінь їхньої реалізації, а також проводиться перевірка відповідності внесених змін заданим вимогам.

*Конфігурація системи* – це склад функцій, програмного і технічного забезпечення системи, можливі їх комбінації залежно від наявності устаткування, загальносистемних засобів і вимог до продукту.

*Конфігурація ПЗ* складається з набору функціональних і технічних характеристик ПЗ, заданих у технічній документації і реалізованих у готовому продукті. Це сполучення різних елементів продукту з заданими процедурами збирання компонентів і настроювання на середовище. Вхідними елементами конфігурації є графік розробки, проектна документація, вихідний виконуваний код, бібліотека компонентів, інструкції з установки і розгортання системи.

***23.Основні задачі керування інженерією програмного забезпечення***

Керування інженерією ПЗ (Software Engineering Management) – керування роботами команди розробників ПЗ у процесі виконання плану проекту, визначення критеріїв ефективності роботи цієї команди й оцінка процесів і продуктів проекту з використанням загальних методів планування і контролю робіт. Як будь-яке керування, воно полягає у плануванні, координації, контролі, вимірі й обліку виконаних робіт у процесі розроблення програмного проекту. Координацію людських, фінансових і технічних ресурсів виконує менеджер проекту аналогічно до того, як це робиться в технічних проектах. У його обов'язки входить дотримання запланованих бюджетних і часових характеристик і обмежень, стандартів і сформульованих вимог. Загальні питання керування проектом містяться в ядрі знань РMBOK [12], а також у стандарті ISO/IEC 12207 – Software life cycle processes, де керування проектом розглядається як організаційний процес ЖЦ.

***24.Види діяльності в базовому процесі інженерії програмного забезпечення.***

Процес інженерії – є метарівнем, що визначає основні поняття, способи реалізації, оцінювання, вимірювання, дії з керування змінами й удосконалення самого процесу.

**Визначення процесу**  метою процесу є підвищення якості одержуваного продукту, поліпшення різних аспектів програмної інженерії, автоматизація і удосконалення процесів.

**Оцінка процесу** проводиться з використанням відповідних моделей і методів оцінки. Наприклад, оцінка потенційної здатності фахівця до розроблення і виконання відповідного процесу, а також оцінювання зрілості процесу, згідно за яким проводиться розроблення ПЗ.

**Якісний аналіз процесу** полягає в ідентифікації і пошуку «слабких місць» у процесі створення ПЗ на початку його функціонування і після експлуатації. Розглядається такі техніки аналізу: огляд даних і порівняння процесу з основними положеннями стандарту ISO/IEC 12207, збирання даних про якість процесів; аналіз головних причин відмов у функціонуванні ПЗ, відкіт назад від точки виникнення відхилення до точки правильної роботи системи для з'ясування причин зміни процесу. На якість результатів проекту і процесу впливають застосовувані інструменти і досвід фахівців.

**Виконання і зміна процесу**. Оцінка вдосконалення процесу проводиться шляхом встановлення кількісних характеристик процесу і продуктів. Після процесу розгортання ПЗ виконуються обчислення функцій і аналіз отриманих результатів, які можуть застосовуватися при оцінюванні якості, продуктивності, трудовитрат та ін. Якщо результати не задовольняють користувача ПЗ, проводять обговорення і приймають рішення щодо необхідності виправлення ситуації шляхом або внесення зміни у процес, або вдосконалення процесу, а також організаційну структуру і деякі інструменти керування змінами.

***25.Основні складові базового процесу інженерії програмного забезпечення.***

Область знань «Процес програмної інженерії (Software Engineering Process)»

складається з таких розділів:

– концепції процесу інженерії ПЗ (Software Engineering Process Concepts),

– інфраструктура процесу (Process Infrastructure),

– визначення процесу (Process Definition),

– оцінки процесу (Process Assessments),

– якісний аналіз процесу (Qualitative Process Analysis),

– виконання і змінювання процесу (Process Implementation and Change).

**Концепції процесу інженерії ПЗ –** задачі і дії, що зв'язані з керуванням, реалізацією, оцінкою, змінами й удосконаленням процесу і/або ПЗ. Ціль керування процесом – це створення інфраструктури процесу, виділення необхідних ресурсів, планування реалізації і зміни процесу з метою впровадження його у практику і, нарешті, оцінка переваг від його впровадження у практику проектування ПЗ.

**Інфраструктура процесу** містить у собі ресурси, стандарти, методики керування якістю, проектом і структуру колективу розробників ПЗ типу: команда, бригада, експериментальна фабрика, каркас виробництва на лінії продуктів і ін. До основних задач інфраструктури належатькерування і комунікації в колективі, інженерні методи виробництва програмного продукту й удосконалення процесу з накопиченим досвідом розробки ПЗ.

**Визначення процесу** ґрунтується на: типах процесів і моделей (каскадна,спіральна, ітераційна й ін.);

**Оцінка процесу** проводиться з використанням відповідних моделей і методів оцінки.

**Якісний аналіз процесу** полягає в ідентифікації і пошуку «слабких місць» у процесі створення ПЗ на початку його функціонування і після експлуатації.

**Виконання і зміна процесу**. Існує ряд фундаментальних аспектів вимірювань в програмній інженерії, що покладені в основу детальних вимірювань процесу.

***26. Поняття якості програмного забезпечення.***

*Якість ПЗ* – набір властивостей продукту (сервісу або програм), що

характеризують його здатність задовольнити встановлені або передбачувані

потреби замовника. Поняття якості має різні інтерпретації залежно від конкретної

програмної системи і вимог до неї. Крім того, у різних джерелах таксономія

(класифікація) характеристик у моделі якості розрізняється.

Моделі мають різну кількість рівнів і повністю або частково збігаються щодо

набору характеристик якості. Наприклад, модель якості МакКолла на найвищому

рівні має три характеристики: функціональність, модифікованість і переносність, а

46 Розділ 1

на нижчих рівнях моделі – 11 підхарактеристик якості і 18 критеріїв (атрибутів)

якості.

Стандарт ISO 9126:2001 регламентує *зовнішні і внутрішні характеристики*

якості. Перші відображають вимоги до функціонування програмного продукту. Для

кількісного встановлення критеріїв якості, за якими буде здійснюватися перевірка і

підтвердження відповідності ПЗ заданим вимогам, визначаються відповідні

зовнішні вимірювані властивості (зовнішні атрибути*)* ПЗ, метрики (наприклад, час

виконання окремих компонентів), діапазони зміни значень і моделі їх оцінки.

Метрики використовуються на стадії тестування або функціонування і називаються

*зовнішніми метриками.* Вони являють собою моделі оцінки атрибутів.

*Внутрішні характеристики* якості і внутрішні атрибути ПЗ

використовуються для складання плану досягнення необхідних зовнішніх

характеристик якості продукту. Для квантифікації внутрішніх характеристик якості

застосовують внутрішні метрики, як інструмент перевірки відповідності проміжних

продуктів внутрішнім вимогам до якості, які формулюються на процесах, що

передують тестуванню.

Зовнішні і внутрішні характеристики якості відображають властивості самого

ПЗ (працюючого або не працюючого), а також погляд замовника і розробника на

таке ПЗ. Безпосереднього кінцевого користувача ПЗ цікавить експлуатаційна якість

ПЗ – сукупний ефект від досягнення характеристик якості, що виміряється строком

результату, а не властивістю самого ПЗ. Це поняття ширше, ніж будь-яка окрема

характеристика (наприклад, зручність використання або надійність).

Остаточна оцінка якості проводиться відповідно до стандарту ISO/IEC 14598.

Якість може підвищуватися за рахунок постійного поліпшення використовуваного

продукту виявленням, усуненням дефектів у ПЗ і їх запобіганням.

Область знань «Якість ПЗ (Software Quality)» складається з наступних

розділів:

– концепції якості ПЗ (Software Quality Concepts);

– визначення і планування якості (Definition & Planning for Quality);

– техніки й види діяльності, що забезпечують гарантію якості, валідацію і

верифікацію (Activities and Techniques for Software Quality Assurance, Validation &

Verification –V&V);

– вимірювання при аналізі якості ПЗ (Measurement in Software Quality

Analysis).

**Концепції якості ПЗ** – це зовнішні і внутрішні характеристики якості, їхні

метрики, а також моделі якості, визначені на множині цих характеристик, що

наведені в стандартах з якості і в [8, 9] – це шість характеристик і кожна з них має

кілька атрибутів. До характеристик якості належать:

– функціональність (functionality);

– надійність (realibility);

– зручність застосування (usability);

– ефективність (efficiency);

– супровід (maitainnability);

– переносність (portability).

Базова модель якості містить у собі ці характеристики і вони притаманні

будь-якому типу програмних продуктів. При розробці вимог замовник формулює

Розділ 1 47

такі вимоги до якості, які найбільшою мірою підходять для програмного продукту,

який замовляється.

**Визначення і планування якості ПЗ** ґрунтується на положеннях стандартів

у цій області, складанні планів і графіків робіт, процедурах перевірки і ін. План

забезпечення якості містить у собі набір дій для перевірки процесів забезпечення

якості (верифікація, валідація і ін.) і формування документа з керування якістю.

Планування якості призначено для підтримки керування процесами

досягнення якості продуктів проекту (зокрема проміжних робочих) і ресурсів –

програмних, технічних, виконавських і ін. Воно також передбачає керування

вимогами до процесів і продуктів і полягає в наступному:

– визначення продукту термінами заданими характеристиками якості;

– планування процесів для гарантії одержання необхідної якості;

– вибір методів оцінки запланованих характеристик якості і встановлення

відповідності продукту сформульованим вимогам.

У стандарті ISO/IEC 12207 визначені спеціальні процеси забезпечення якості

– верифікація, валідація (атестація), спільний аналіз і аудит.

**Види діяльності і техніки гарантії якості** містять у собі, зокрема:

інспекцію, верифікацію і валідацію ПЗ.

*Інспекція ПЗ* – аналіз і перевірка різних видів подання системи і ПЗ

(специфікації, архітектурної схеми, діаграм, початкового коду тощо). Виконується

на всіх процесах ЖЦ розробки ПЗ.

*Верифікація ПЗ* – процес забезпечення правильної реалізації ПЗ відповідно до

специфікацій, виконується протягом усього життєвого циклу. Верифікація дає

відповідь на питання, чи правильно створюється система.

*Валідація* – процес перевірки відповідності ПЗ функціональним і

нефункціональним вимог і очікуваним потребам замовника.

Верифікація і валідація (V&V) можуть виконуватися, починаючи з ранніх

стадій ЖЦ. Вони орієнтовані на отримання правильних функцій ПЗ, плануються і

забезпечуються визначеними ресурсами з чітким розподілом ролей. Перевірка

ґрунтується на використанні відповідних технік тестування для виявлення тих або

інших дефектів і збирання статистики. Після зібрання даних оцінюється

правильність реалізації вимог і роботи ПЗ у заданих умовах.

**Вимірювання при аналізі якості ПЗ** ґрунтується на метриках продукту і

даних, зібраних у процесі створення продукту при заданих ресурсах: оцінок

процесів, ПЗ і його моделей, і передбачає документування вимірів. Оцінювання

якості продукту полягає у вимірюванні і оцінюванні якісних показників за

допомогою даних про різні типи помилок і відмов під час тестування ПЗ і

виконання коду на тестових даних. Ці дані аналізуються, перевіряються і

використовуються при якісній і кількісній оцінки ПЗ.

Для імітації роботи системи в режимі тестування розробляються тести з

реальними вхідними даними для перевірки правильності роботи ПЗ на різних

фрагментах програми і шляхах проходження в них операторів. У процесі

тестування ПЗ виявляються різного роду помилки (відмови, дефекти, помилки

тощо), кількість яких значною мірою може вплинути на одержання правильного і

якісного результату.

З урахуванням типів виявлених помилок можна встановити наявність (або

відсутність) відповідності реалізованих і нереалізованих функцій, заданих у

48 Розділ 1

вимогах до системи, а також оцінити способи реалізації нефункціональних вимог

(продуктивності, надійності та ін.). Оцінюються процеси керування планами,

інспекціями, прогонами і т.п. За цими оцінками приймаються рішення про

завершення розробки продукту проекту і передачі його замовнику в експлуатацію,

під час якої можуть бути внесені зміни щодо усунення помилок, визначення

адекватності планів і вимог, оцінки ризиків перероблення ПЗ тощо.

Мета інспекцій – виявлення різних аномальних станів у ПЗ незалежними

фахівцями команди експертів із залученням авторів проміжного або кінцевого

продукту. Експерти інспектують виконання вимог, інтерфейси, вхідні дані і т.п., а

потім документують виявлені відхилення в проекті.

Призначення аудита – незалежна оцінка продуктів і процесів на відповідність

регулюючим і регламентуючим документам (планам, стандартам і ін.),

формулювання звіту про випадки невідповідності і пропозицій для їх коригування.

Таким чином, розгляд розділів SWEBOK свідчить по те, що це ядро містить

весь необхідний набір знань з програмної інженерії, який повинні мати фахівці

різних профілів (аналітики, інженери, програмісти, оцінювачі, контролери тощо),

що виробляють програмний продукт.

Зазначимо, що ядро знань SWEBOK не позбавлено недоліків тактичного

характеру. Так, між областями знань у цьому ядрі існують перетини за методами,

концепціями і стратегіями, а деякі важливі напрями програмної інженерії взагалі

не відбиті у ньому наявними областями знань. Це стосується, наприклад, методів

доведення правильності програм, еволюції програм, розподілених і неоднорідних

середовищ, взаємодії систем, таких методів програмування, як аспектне, агентне,

сервісне й інші, а також аспектів захисту, безпеки тощо.

***27. Види діяльності при забезпеченні якості програмного забезпечення.***

Види діяльності і техніки гарантії якості містять у собі, зокрема:

інспекцію, верифікацію і валідацію ПЗ.

Інспекція ПЗ – аналіз і перевірка різних видів подання системи і ПЗ

(специфікації, архітектурної схеми, діаграм, початкового коду тощо). Виконується

на всіх процесах ЖЦ розробки ПЗ.

Верифікація ПЗ – процес забезпечення правильної реалізації ПЗ відповідно до

специфікацій, виконується протягом усього життєвого циклу. Верифікація дає

відповідь на питання, чи правильно створюється система.

Валідація – процес перевірки відповідності ПЗ функціональним і

нефункціональним вимог і очікуваним потребам замовника.

Верифікація і валідація (V&V) можуть виконуватися, починаючи з ранніх

стадій ЖЦ. Вони орієнтовані на отримання правильних функцій ПЗ, плануються і

забезпечуються визначеними ресурсами з чітким розподілом ролей. Перевірка

ґрунтується на використанні відповідних технік тестування для виявлення тих або

інших дефектів і збирання статистики. Після зібрання даних оцінюється

правильність реалізації вимог і роботи ПЗ у заданих умовах.

***28. Поняття життєвого циклу програмного забезпечення.***

Життєвий цикл – схема виконання робіт із проектування системи,

починаючи з моменту прийняття рішення про необхідність її побудови і

закінчуючи моментом її повного вилучення з експлуатації.

**Загальні поняття**

В загальному випадку, [життєвий цикл](http://uk.wikipedia.org/wiki/%D0%96%D0%B8%D1%82%D1%82%D1%94%D0%B2%D0%B8%D0%B9_%D1%86%D0%B8%D0%BA%D0%BB_%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%86%D1%96%D1%97) визначається моделлю й описується у формі методології (методу). Модель або парадигма життєвого циклу визначає загальну організацію і, як правило, основні його фази та принципи переходу між ними. Методологія (метод) визначає комплекс робіт, їх детальний зміст і рольову відповідальність спеціалістів на всіх етапах вибраної моделі.

Життєвий цикл програмного забезпечення супроводжується розробленням, обігом та використанням [програмної документації](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B0_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D1%96%D1%8F).

*Програмна документація* — сукупність документів, що містять відомості, необхідні для розробки, виготовлення, супроводу та експлуатації [програм](http://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0)[[2]](http://uk.wikipedia.org/wiki/%D0%96%D0%B8%D1%82%D1%82%D1%94%D0%B2%D0%B8%D0%B9_%D1%86%D0%B8%D0%BA%D0%BB_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F#cite_note-GOST19101-2). Програмна документація є одним з видів [технічної документації](http://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D1%96%D1%87%D0%BD%D0%B0_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D1%96%D1%8F).

Комплекс державних стандартів, що встановлюють взаємопов'язані правила розробки, оформлення та обігу програм і програмної документації називається [«Єдина система програмної документації»](http://uk.wikipedia.org/wiki/%D0%84%D0%B4%D0%B8%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D1%97_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D1%96%D1%97) (ЄСПД)[[3]](http://uk.wikipedia.org/wiki/%D0%96%D0%B8%D1%82%D1%82%D1%94%D0%B2%D0%B8%D0%B9_%D1%86%D0%B8%D0%BA%D0%BB_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F#cite_note-GOST19001-3).

Кожна ПС протягом свого існування проходить визначену послідовність

процесів (процесів), починаючи від постановки задачі до її втілення в готову

54 Розділ 2

програму, наступної експлуатації і остаточного виведення з експлуатації та

списання. Така послідовність процесів називається життєвим циклом розробки

ПС. На кожному процесі ЖЦ виконується визначена сукупність процесів і/або

підпроцесів, кожний з яких породжує відповідний проміжний продукт,

використовуючи при цьому результати попереднього процесу і доробок.

***29. Основні процеси життєвого циклу програмного забезпечення.***

В стандарті до основних процесів належать:

– процес придбання, який ініціює ЖЦ ПС і визначає дії організації-покупця

(або замовника), що отримує автоматизовану систему, програмний продукт або

сервіс. Цей процес містить у собі такі види діяльності: ініціювання і підготовка

запиту, оформлення контракту і його актуалізація; моніторинг користувачів,

приймання і завершення;

– процес постачання, який визначає дії з передачі покупцю програмного

продукту або сервісу і містить у собі такі види діяльності: підготовку пропозицій

(відповідей на запити); оформлення контракту; планування, виконання і контроль

продукту, що постачається; аналіз і оцінку продукту; постачання і завершення

робіт з постачання. Процес постачання починається тоді, коли встановлені

договірні відношення між замовником і постачальником. Залежно від умов

договору процес постачання може містити у собі процес розроблення ПЗ, процес

експлуатації і супроводження для виправлення і поліпшення ПС;

52 Розділ 2

– процес розроблення, який визначає дії підприємства-розробника

програмного продукту: аналіз вимог до системи; проектування архітектури

системи, детальне проектування компонентів ПС, кодування і тестування ПС,

інтеграцію системи, кваліфікаційне тестування, установку ПС і забезпечення

приймання ПС (рис.2.1);

Рис. 2.1. Схема основних процесів ЖЦ ПС

– процес експлуатації, який визначає дії підприємства-оператора, що

забезпечує обслуговування системи в ході її експлуатації користувачами

(консультування користувачів, вивчення потреб операторів, задоволеності

споживачів системою тощо). Цей процес регламентує задачі і дії з функціонального

тестування, перевірки правильності експлуатації системи; дотримання інструкцій і

настанов з її запуску;

– процес супроводження, який визначає дії організації, що виконує супровід

програмного продукту (керування модифікаціями, підтримку поточного стану і

функціональної придатності, інсталяцію програмного продукту на обчислювальній

системі користувача та її вилучення при списанні). Даний процес містить у собі

завдання і дії щодо аналізу питань супроводження і модифікації, розробки планів і

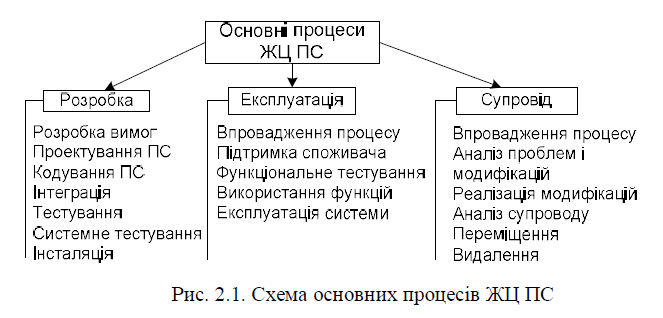
реалізації модифікації системи, аналізу результатів супроводження після змін

системи, міграції ПС в інше середовище або її виведення з експлуатації.

До категорії основних процесів належать також «первинні» процеси, що

визначають порядок підготовки договору на розробку ПС, моніторинг діяльності

постачальників ПС тощо (див. табл.2.1).



***30. Допоміжні процеси життєвого циклу програмного забезпечення.***

Стандарт містить у собі опис допоміжних процесів, що регламентують

додаткові дії з перевірки продукту, керування проектом та його якістю (рис.2.2).

До процесів підтримки розроблення ПС належать: документування,

керування версіями, верифікація і валідація, перегляди, аудити, оцінювання

продукту та ін. Процес керування версіями за змістом відповідає керуванню

конфігурацією системи, що так само, як і продукти процесів, повинні перевірятися

на правильність реалізації цілей проекту і відповідність вимогам замовника.

Завдання з перевірки рекомендується виконувати спеціальним контролерам, які

знаються на методах і процесах проектування ПС.

До організаційних процесів належать: керування проектом (менеджмент

розробки), якістю, ризиками тощо.

Розділ 2 53

Рис. 2.2. Схема допоміжних процесів ЖЦ ПС

Ці процеси виконуються спеціальними службами, що здійснюють

планування робіт у проекті, контроль процесів, визначення метрик для

вимірювання продуктів, перевірку показників якості, дотримання стандартних

положень та ін.

Процеси, дії і задачі наведені в стандарті в найбільш загальній природній

послідовності. Залежно від цілей конкретного проекту головний розробник і

менеджер вибирають процеси, дії і задачі, вибудовують визначену схему ЖЦ для

застосування в цьому проекті.

Процеси керування в стандарті структуровані за рівнями і напрямами, вони

жодним чином не зв’язані з існуючими методами і засобами програмної інженерії з

розроблення ПС. Це дає можливість при їх виборі для застосування у ЖЦ

зіставляти з ними звичні парадигми і методи розроблення (об'єктні, компонентні,

сервісні й ін.) та засоби ядра знань SWEBOK.

Таким чином, між стандартом ISO/IEC 12207 і ядром знань SWEBOK існує

зв'язок: вони взаємодоповнюють та збагачують один одного, більше у розробці

відповідних документів брали участь одні ті самі висококваліфіковані фахівці в

галузі програмування й інформатики. Інженерна дисципліна проектування ПС

використовує теоретичні, прикладні методи і засоби розробки ПС і стандарти

(ISO/IEC 12207, ISO/IEC 15404, ISO/IEC 9126 та ін.), а також рекомендації і

методики керування розробкою, до яких відносять методи оцінки на процесах ЖЦ,

якості ПС, витрачених ресурсів і вартості виконаних робіт. При цьому ядро знань

SWEBOK, а також численні монографії і статті рекомендують до застосування

методи і засоби програмної інженерії, а стандарт дає настанови до побудови

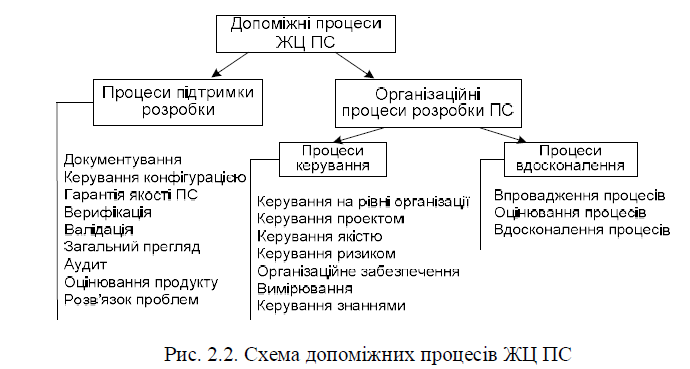
процесів на стандартизованій інженерній основі.

Природно, що в невеликих програмних проектах завжди можна буде

застосовувати творчі і неформальні підходи, запропоновані фахівцями для

створення різного роду унікальних продуктів, процес розроблення яких не завжди

відповідає загальному стандарту.



***31. Моделі життєвого циклу програмного забезпечення.***

**Модель життєвого циклу** - це структура, що складається із процесів, робіт та задач, які включають в себе розробку, експлуатацію і супровід програмного продукту; охоплює життя системи від визначення вимог до неї до припинення її використання. На сьогодні найбільшого розповсюдження набули дві моделі:

* каскадна модель;
* спіральна модель.

Модель життєвого циклу – це схема виконання робіт і задач у рамках

процесів, що забезпечують розробку, експлуатацію і супровід програмного

продукту. Ця схема відображає еволюцію ПС, починаючи від формулювання вимог

і закінчуючи припиненням користування нею [1– 6].

Історично така схема робіт містить у собі:

– розробку вимог або технічного завдання;

– розробку ескізного або технічного проекту;

– програмування або робоче проектування;

– пробну експлуатацію;

– супровід і поліпшення;

– зняття з експлуатації.

Основне призначення моделей ЖЦ є таким:

– планування і розподіл робіт і ресурсів між розробниками, а також

керування програмним проектом;

– забезпечення взаємодії між розробниками проекту і замовником;

– спостереження і контроль робіт, оцінка проміжних продуктів ЖЦ на

дотримання специфікацій вимог, правильне їх виконання, оцінка продукту і

реальних витрат, у тому числі і щодо застосовуваних програмних засобів і

інструментів;

– узгодження проміжних результатів із замовником;

– перевірка правильності кінцевого продукту шляхом його тестування на

запланованих і погоджених із замовником наборах тестів;

– оцінка відповідності характеристик якості отриманого продукту заданим

вимогам;

– обговорення використовуваних процесів ЖЦ з метою оцінки їх потенційних

можливостей і недоліків, що виявлялися при їх застосуванні, а також визначення

напрямів удосконалення або модернізації ЖЦ.

Процеси, що включені в модель ЖЦ, призначені для реалізації стандартних

задач процесів ЖЦ, вони можуть залучати інші процеси, що пов'язані із

забезпеченням захисту даних. Інтерфейси (входи і виходи) будь-яких двох процесів

ЖЦ повинні бути мінімальними і кожний з них повинен відповідати таким

правилам:

– якщо процес А викликається процесом В і тільки процесом В, то А

належить В;

– якщо функція викликається більше ніж одним процесом, то вона стає

окремим процесом;

АВТОРСЬКЕ ПРАВО НАЛЕЖИТЬ ГРУПІ КІ-35

– перевірка будь-якої функції в ЖЦ є обов'язковою.

Іншими словами, якщо вирішення певної задачі потребує більше ніж один

процес, то вона може сама набути статусу процесу, що використовується одно- або

багаторазово протягом ЖЦ конкретної системи. Кожен процес повинен мати

внутрішню структуру, встановлену відповідно до особливостей його виконання.

Процеси конкретної моделі ЖЦ орієнтовані безпосередньо на розробника

даної системи. Розробник може виконувати один або кілька процесів і процес, у

свою чергу, може бути виконаний одним або кількома розробниками. При цьому

один з розробників має відповідати за один процес або за всі процеси моделі, навіть

якщо окремі роботи виконує інший розробник.

Створювана модель ЖЦ узгоджується з конкретними методиками розробки

систем і відповідними стандартами в області програмної інженерії, які існують або

розробляються самостійно для проекту з урахуванням можливостей і особливостей

ПС. Іншими словами, кожен процес ЖЦ підкріплюється вибраними для реалізації

ПС засобами і методами програмування, а також методикою їх застосування і

виконання.

При формуванні моделі ЖЦ важливу роль відіграють організаційні аспекти:

– структура колективу і зв’язків між ними;

– планування послідовності робіт і термінів їх виконання;

– підбір і підготовка ресурсів (людських, програмних і технічних) для

виконання робіт;

– оцінка можливостей реалізації проекту в заданий термін, вартість і ресурси.

56 Розділ 2

Впровадження моделі ЖЦ у практичну діяльність зі створення програмного

продукту дозволить впорядкувати взаємини між суб'єктами процесу розроблення

ПС і враховувати динамічні модифікації вимог до проекту і до системи.

***32 .Типи моделей життєвого циклу програмного забезпечення.***

Розглянуті підходи щодо побудови різних видів моделей ЖЦ базуються на

процесному підході до виконання програмних проектів. Вони використовувалися

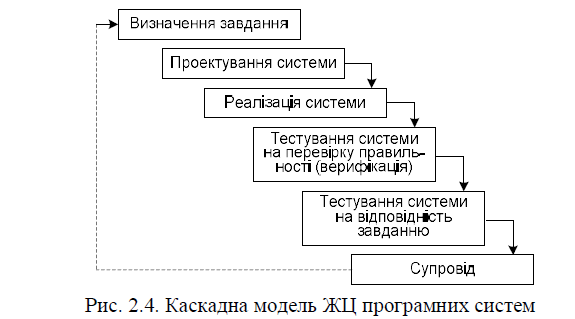
на практиці під час створення різних типів моделей ЖЦ, до яких належать такі

моделі: каскадна, спіральна, інкрементна, еволюційна та ін.

***33. Особливості каскадної моделі життєвого циклу програмного забезпечення.***

Однією з перших почала застосовуватися каскадна модель, де кожна робота

виконується один раз і в такому порядку, який подано на рис.2.4.



Тобто вважається, що кожна робота має бути виконана настільки ретельно,

що після її закінчення і переходу до наступного етапу, повертатися до

попереднього не буде потреби. Розробник перевіряє проміжний результат відомими

методами верифікації і фіксує його як готовий еталон для наступного процесу.

Згідно з даною моделлю роботи і завдання процесу розроблення зазвичай

виконуються послідовно, як це наведено у схемі.

Проте допоміжні і організаційні процеси (контроль вимог, керування якістю і

ін.), як правило, виконуються разом з процесами розробки ПЗ. У даній моделі

повернення до початкового процесу передбачається після супроводження і

виправлення помилок.

Особливість такої моделі полягає у фіксації послідовних процесів

розроблення програмного продукту. В її основу покладена модель фабрики, де

продукт проходить стадії від задуму до виробництва, потім його передають

замовнику у вигляді готового виробу, де заміна не передбачена, хоча можна

подати інший подібний виріб. Недоліки цієї моделі такі:

– процес створення ПС не завжди вкладається в таку жорстку форму і

послідовність дій;

– не враховуються змінювані потреби користувачів, нестабільні умови

зовнішнього середовища, які впливають на зміни вимог до ПС під час ї

розроблення;

– значний розрив між часом внесення помилки (наприклад, на процесі

проектування) і часом її виявлення (при супроводі), що призводить до суттєвої

переробки ПС.

Розділ 2 59

При застосуванні каскадної моделі можуть спостерігатися такі чинники

ризику:

– вимоги до ПС недостатньо чітко сформульовані, або не враховують

перспективи розвитку ОС, середовищ і т.п.;

– громіздка система, що не допускає компонентної декомпозиції, може

викликати проблеми щодо розміщення її в пам'яті або на платформах, не

передбачених у вимогах;

– внесення швидких змін до технології і у вимоги може погіршити процес

розроблення окремих частин системи або системи в цілому;

– обмеження на ресурси (людські, програмні, технічні і ін.) в ході розробки

можуть звузити окремі можливості реалізації системи;

– отриманий продукт може виявитися не придатним для застосування

внаслідок нерозуміння розробниками вимог або функцій системи або недостатньо

проведеного тестування.

Переваги реалізації системи за допомогою каскадної моделі такі:

– всі завдання підсистем і системи реалізуються одночасно, завдяки чому не

можна забути жодного завдання, а це сприяє встановленню стабільних зв'язків між

ними;

– повністю розроблену систему з документацією на неї легше

супроводжувати, тестувати, фіксувати помилки і вносити зміни не хаотично, а

цілеспрямовано, починаючи з вимог, наприклад, додавати або замінювати деякі

функції і повторювати процес.

Каскадну модель можна розглядати як модель ЖЦ, придатну для створення

першої версії ПЗ з метою перевірки реалізованих в ній функцій. При супроводі і

експлуатації можуть бути виявлені різного роду помилки, виправлення яких

спричинить повторне виконання всіх процесів, починаючи з уточнення вимог.

***34. Особливості інкрементної моделі життєвого циклу програмного забезпечення.***

Цю модель (incremental) ще називають моделлю з нарощуванням або з

приростом. Її суть полягає в розробці продукту ітераціями, і кожна ітерація

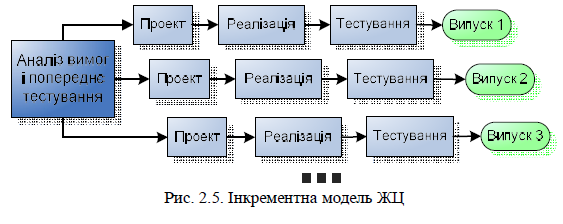
закінчується випуском працездатної версії. У кожній новій версії додаються деякі

функціональні можливості. Розробка системи починається з визначення набору всіх

вимог до ПС і ділення процесу розроблення на ітерації. Кожна ітерація реалізується

послідовно з використанням процесів ЖЦ і фіксації робочої версії системи,

системи, що поступово наближається до остаточної версії (рис. 2.5).



Перша проміжна версія системи, що створюється (випуск 1), реалізує частину

вимог, у подальшу версію (випуск 2) додають додаткові вимоги до тих пір, поки не

будуть остаточно виконані всі вимоги і розв’язані задачі розробки системи.

Для кожної проміжної версії на процесах ЖЦ виконуються необхідні

процеси, роботи і завдання, зокрема, аналіз вимог і створення нової архітектури, які

можуть бути виконані одночасно.

Процеси розроблення технічного проекту програмної системи, її

програмування і тестування, збирання і кваліфікаційні випробування ПС

виконуються при створенні кожної подальшої версії.

Відповідно до даної моделі ЖЦ, процеси якої практично такі самі, як і в

каскадній моделі, наголос робиться на побудову деякої закінченої проміжної

версії, а завдання процесу розроблення виконуються послідовно або частково

паралельно для ряду окремих проміжних структур версії.

Роботи і завдання процесу розроблення наступної версії системи з

додатковими вимогами або функціями можуть виконуватися неодноразово в одній

тій же послідовності для всіх проміжних версій системи. Процеси супроводження і

експлуатації можуть бути реалізовані разом з процесом розроблення версії шляхом

перевірки частково реалізованих вимог в кожній проміжній версії і так до

отримання кінцевого варіанта системи. Допоміжні і організаційні процеси ЖЦ

зазвичай виконуються разом з процесом розроблення версії системи і до кінця

розробки збираються дані, на підставі яких можна встановити рівень завершеності і

якості виготовленої системи.

При застосуванні даної моделі необхідно враховувати такі чинники ризику:

– вимоги складені з урахуванням можливості їх зміни при реалізації

продукту;

– всі можливості системи потрібно реалізувати одразу;

– швидка зміна технології і вимог до системи може призвести до порушення

отриманої структури системи;

– обмеження в ресурсному забезпеченні (виконавці, фінанси) можуть

призвести до несвоєчасного введення системи в експлуатацію.

Дану модель ЖЦ доцільно використовувати, у випадках, коли:

– бажано реалізувати деякі можливості системи швидко за рахунок створення

проміжної версії продукту;

– система декомпозується на окремі складові частини, які можна

реалізовувати як деякі самостійні проміжні або готові продукти;

– можливе збільшення фінансування на розробку окремих частин системи.

***35. Особливості спіральної моделі життєвого циклу програмного забезпечення.***

Виходячи з можливості внесення змін, як в процес, так і в проміжний

продукт було створено спіральну модель ЖЦ (рис.2.6) [6].

Внесення змін орієнтоване на задоволення потреби користувачів одразу, як

тільки буде встановлено, що створені артефакти або елементи документації не

відповідають дійсному стану розробки.

Дана модель ЖЦ допускає аналіз продукту на витку розробки, його

перевірку, оцінку правильності та прийняття рішення про перехід на наступний

виток або повернення на попередній виток для доопрацювання на ньому

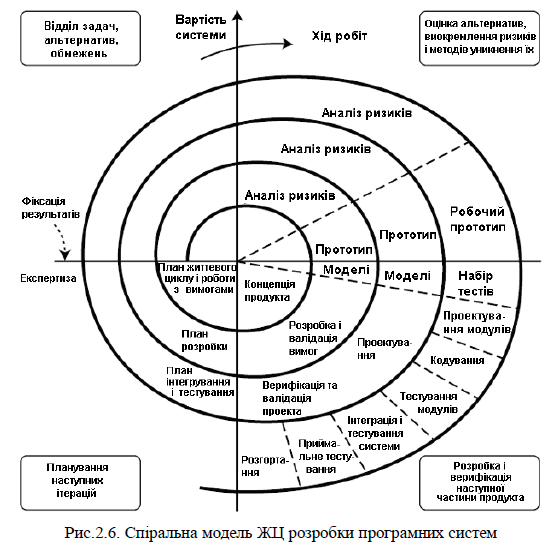
проміжного продукту.

Розділ 2 61

Відмінність цієї моделі від каскадної полягає в можливості багато разів

повертатися до процесу формулювання вимог і до повторної розробки версії

системи з будь-якого процесу моделі.



Для програмного продукту така модель не дуже підходить з декількох

причин. По-перше, висловлення вимог замовником носить суб'єктивний характер,

вимоги можуть багаторазово уточнюватися протягом розробки ПС i навіть після

завершення та випробовування, і часом може з'ясуватися, що замовник «хотів

зовсім інше». По-друге, змінюються обставини та умови використання системи,

тому загальновизнаним законом програмної інженерії є закон еволюції, який

сформулюємо так: кожна діюча ПС з часом потребує внесення змін або виводиться

з експлуатації.

При необхідності внесення змін до системи на кожному витку з метою

отримання нової версії системи обов'язково вносяться зміни в заздалегідь

зафіксовані вимоги, після чого повертаються на попередній виток спіралі для

продовження реалізації нової версії системи з урахуванням усіх змін.

***36 Особливості еволюційної моделі життєвого циклу програмного забезпечення.***

Розвиткомспіральноїмоделі є модель еволюційногопрототипування в рамках

усього ЖЦ розробки ПС.

У літературі вона часто називаєтьсямоделлюшвидкоїрозробкипрограм

RAD (RapidApplicationDevelopment).

У даніймоделінаведенідії, якіпов'язані з аналізомїїзастосовності для

конкретного виду системи, а такожобстеженнямзамовника для визначення потреб

користувача при розробці плану створення прототипу.

У моделі є двіголовніітераціїрозробкифункціонального прототипу,

проектування і реалізаціїсистеми з метою перевірки, чизадовольняє вона всі

функціональні і нефункціональнівимоги. Основною ідеєюцієїмоделі є

моделюванняокремихфункційсистеми в прототипі і поступовееволюційнейого

доопрацювання до виконаннявсіхзаданихфункціональнихвимог.

Ітерацій з отриманняпроміжнихваріантів прототипу може бути декілька, в

кожній з якихдодаєтьсяфункція і повторно моделюється робота прототипу. І так

до тих пір, поки не будутьпромодельованівсіфункції, задані у вимогах до системи.

Післяцьоговиконуєтьсяще одна ітерація – остаточнепрограмування для

отриманняготовоїсистеми.

Ця модель застосовується для систем, в якихнайбільшважливими є

функціональніможливості, і якінеобхідношвидкопродемонструвати на CASE-

засобах.

Оскількипроміжніпрототиписистемивідповідаютьреалізаціїдеяких

функціональнихвимог, їхможнаперевіряти і під час супроводу і експлуатації,

тобто разом з процесомрозробкичерговихпрототипівсистеми. При цьому

допоміжні і організаційніпроцесиможутьвиконуватися разом з процесом

розроблення і накопичувативідомості за данимикількісних і якіснихоцінокна

процесахрозроблення.

При цьомувраховуютьсятакічинникиризику:

– реалізаціявсіхфункційсистемиодночасноможепризвестидогроміздкості;

– обмеженілюдськіресурсизайнятірозробкоюпротягомтривалого часу.

Перевагизастосуванняданоїмоделі ЖЦ такі:

– швидкареалізаціядеякихфункціональнихможливостейсистеми і їх

апробація;

– використанняпроміжного продукту в наступномупрототипі;

– виділенняокремихфункціональнихчастиндляреалізаціїїх у вигляді

прототипу;

– можливістьзбільшенняфінансуваннясистеми;

– зворотнийзв'язокіззамовником для уточненняфункціональнихвимог;

– спрощеннявнесеннязмін у зв'язкуіззаміноюокремихфункції.

Модель розвивається у напрямудодаваннянефункціональнихвимогдо

системи, пов'язанихіззахистом і безпекоюданих, несанкціонованим доступом до

них і ін.

***37. Поняття вимог до програмного забезпечення.***

У загальномувипадкупідвимогами до ПС розуміютьвластивості, які

повинна мати система для виконаннязапропонованихзамовникомфункцій.

Прикладами такихфункційможуть бути бізнес-функції, документообіг, керування

даними і структурою інформації, щонеобхідна для прийняттясистемнихрішень, та

ін.

Найпоширенішийінструментінженеріївимог – це UML, у якомувимоги

визначаються й уточнюються через поданняможливихваріантіввикористання ПС

(usecase), щотрансформуються у проектнірішення і архітектурусистеми.

Визначеннявимог є нетривіальною задачею і проводиться, як правило,

шляхом обговоренняпоглядівзамовника на систему з майбутнімиїїрозробниками.

Замовниквисловлюєсвої потреби і представляє погляди щодоавтоматизації

функцій і задач системи, якідалінабуваютьформулювання у вигляді

різноплановихвимог до ПС.

***38. Типи вимог до програмного забезпечення.***

Вимоги до ПЗ є такі: системні, функціональні і нефункціональні вимоги.

Системнівимоги (systemrequirements) визначаютьзовнішніумовивиконання

системнихфункцій і обмежень на створення продукту, а такожвимоги до опису

програмно-апаратнихпідсистем. Системнівимогинакладаютьобмеження на

архітектурусистеми, засобиїївізуальногоподання і функціонування. Для опису

системнихвимогвикористовуютьспеціальнішаблони і форми, щодопомагають

уявитивхідні і вихіднідані й автоматизуватицівимоги.

Функціональнівимоги – цеперелікфункційабосервісів, якіповиннанадавати

система, а такожобмежень на дані і поводженнясистеми при їхньомувиконанні.

Специфікаціяфункціональнихвимог (software requirements specification) – опис

функцій та їхніхвластивостей, які не містять у собіпротиріч і виключень.

Нефункціональнівимогивизначаютьумовивиконанняфункцій (наприклад,

захистінформації у БД, аутентифікація доступу до ПС тощо) у середовищі, що

безпосередньо не пов'язані з функціями, а відбивають потреби користувачівщодо

їхвиконання. Цівимогихарактеризуютьпринципивзаємодіїізсередовищамиабо

іншими системами, а такожвизначаютьпоказники часу роботи, захистуданих і

досягненняякості з урахуваннямрекомендаційвикористовуваного стандарту. Вони

можутьвстановлюватися як числовізначення (наприклад, час чеканнявідповіді,

кількістьклієнтів, щообслуговуються і ін.) у різниходиницяхвиміру, включаючи,

наприклад, ймовірність (значенняймовірностібезвідмовноїроботисистеми –

показникаїїнадійності).

***39. Діяльність при збиранні вимог до програмного забезпечення.***

Джереламивідомостей для збираннявимог є:

– мета і задачі проекту, щоформулюєзамовникмайбутньоїсистеми, і які

повинніосмислюватисярозробниками;

– колектив, якийвиконуєреалізаціюфункційсистеми.

Вивчення і фіксаціяреалізованихфункціональнихможливостей у діючій

системі є підґрунтям для накопиченнядосвіду для формулюванняновихвимог до

неї. При цьомунеобхідновідокремлюватиновівимоги до системивідстарих

вимог, щоб не повторитиневдалірозв’язкищодостароїсистемив новому її

виконанні.

Вимоги до системиформулюютьсязамовником у термінах понять його

предметноїобласті з урахуваннямвідомихсловників, стандартів, існуючих умов

середовищафункціонуваннямайбутньоїсистеми, а такожтрудових і фінансових

ресурсів, виділених на розробкусистеми.

Методизбираннявимогтакі:

– інтерв'ю з виразникамиінтересівзамовникасистеми;

– вивченняприкладівможливихваріантіввиконанняфункцій, ролей

відповідальнихосіб, якіпропонуютьціваріанти, або тих, щовзаємодіютьіз

системою при їїфункціонуванні;

– спостереження за роботоюдіючоїсистеми для відокремленнявластивостей,

щообумовленікадровими аспектами.

Зовнішні і внутрішніаспективимогпов’язують з характеристиками якості і

відносяться до властивостействорюваного продукту, а саме, функційсистеми, її

призначення і виконання в заданомусередовищі. На прикінцікористувачочікує

досягненнямаксимальногоефектувідзастосуваннявихідного продукту та

орієнтується на йогокінцевуексплуатаційнуякість.

Отриманнязовнішніх і внутрішніх характеристик якостідосягається

спеціальнорозробленими методами з виконанняпроцесів ЖЦ. Внутрішні

характеристики, якідосягаються в ході ЖЦ, позначаються на зовнішніхпоказниках

і використовуються при оцінюванніробочих та кінцевихпродуктів ПС.

Остаточно сформульованівимоги – основа для підпису контракту між

замовником і розробникомсистеми.

***40. Діяльність в інженерії вимог до програмного забезпечення*.**

Інженернадисциплінааналізу і документуваннявимогпередбачаєпланування

і перетвореннязапропонованихзамовникомвимог до системи на описвимог до ПЗ,

їхспецифікацію і валідацію. Вона базується на моделях процесіврозроблення

вимог, діяхакторів і керуванніпоступовимперетвореннямвимог до проектних

рішень і описукомпонентів у мовіпрограмування.

Модель процесувизначеннявимог – це схема процесів ЖЦ, щовиконуються

від початку проекту і доти, поки не будутьвизначені і погодженівимоги.

Керуваннявимогами до ПЗ полягає в плануванні і контроліформування

вимог, заданих на основіпроектнихрішень, у перетворенніїх на специфікації

компонентівсистеми.

Якість і процесполіпшеннявимог – цеметодидосягнення і перевірки

характеристик і атрибутівякості (надійність, реактивність та ін.), які повинна мати

система і ПЗ, у процесах ЖЦ і під час закінченнярозробки продукту.

Керуваннявимогами до системи – цепланування і керуванняформуванням

вимог на всіхпроцесах ЖЦ, а саме, керуваннязмінамивимог, відновленняїхнього

джерела й уточненнявимог. Невід'ємнаскладовапроцесукерування – трасування

вимог, щополягаєувідстеженніправильностізавдання і реалізаціївимог до

системи і ПЗ на процесах ЖЦ і зворотногопроцесузвіряння ПЗ іззаданими

вимогами.

Основнізадачікеруваннявимогамице:

– розробленняатрибутіввимог,

– керуванняваріантамивимог,

– керуванняризиками, щовиникають при неточному визначеннівимог,

– контроль статусу вимог, вимірюваннязусиль при формуваннівимог;

– реалізаціявимог на процесах ЖЦ.

Розроблення і керуваннявимогамизв'язана з іншими областями знань.

Плануванняробіт на проектістосуєтьсяпитаньорганізаціїінтеграції

компонентів, керуванняризиками, версіямисистеми, наяківпливаютьзадані

вимоги та їхнізміни.

***41.Призначення фіксації вимог до програмного забезпечення.***

**Фіксація вимог**

Початковий процес розроблення ПС – збирання вимог, який завершується

формуванням списку вимог до системи, що називається у вітчизняній практиці

технічним завданням. Фіксація вимог (Requirement Capturing) у технічному

завданні обумовлена потребою замовника зафіксувати і одержати задані ним

властивості у реалізованій системі. При цьому передбачається специфікація,

верифікація і валідація вимог на правильність, відповідність і повноту.

*Специфікація вимог до ПЗ* **–** це формалізований опис функціональних,

нефункціональних і технічних вимог, вимог до характеристик якості, до структури

ПЗ і принципів взаємодії з його компонентами.

**Приклад.** Скласти вимоги до облікової і статистичної функцій ПЗ системи

обробки даних.

Згідно з стандартом ДСТУ 34.601–92 («Розробка АС») функціональні вимоги

до ПЗ даної системи можливо представити так:

– система повинна мати 12 функцій, з них 8 облікових та 4 статистичних;

– кожна функція повинна бути ретельно реалізована, бути коректною і

повинна давати точні результати;

– дані для функцій подаються в табличному вигляді і зберігаються в БД

СКБД Oracle, їхній обсяг – 10000 записів на рік;

– дані повинні контролюватися і бути захищені від несанкціонованого

доступу до БД тощо.

Дані вимоги відносять до характеристик функціональності системи. Після її

реалізації повинні бути перевірені функції на відповідність установленим вимогам,

діючим нормам і стандартам. Оцінка функцій виконується після їхньої валідації,

верифікації та реалізації. Як оцінки використовують метрики (коректність,

точність, повнота тощо) для перевірки різних аспектів реалізації функцій. Ці

метрики наведені у першій, а їхні формули – у другій колонках табл.3.1. У ці

формули підставлені значення, яким надано кількісний опис у специфікації вимог.

На основі отриманих метрик розв’язується задача правильної реалізації вимог до

ПЗ або до всієї програмної системи.

***42.Призначення трасування вимог до програмного забезпечення.***

**Трасування вимог**

Одна з головних проблем збирання вимог – їхня зміна. Вимоги створюються

ітераційно шляхом постійного спілкування представників замовників з аналітиками

і розробниками майбутньої системи з метою виявлення необхідних потреб. Вимоги

змінюються в міру уточнення функцій і задач, умов їхнього визначення на процесі

укладання договору на створення системи і, зрештою, відповідають поглядам

замовника на систему [4].

Одним з інструментів установлення залежності між сформульованими

вимогами та їхніми змінами є *трасування*, тобто розвиток і обробка вимог із

простежуванням ідентифікованих зв'язків, що повинні бути зафіксовані за двома

напрямками – від потреб до робочих продуктів, і навпаки (рис. 3.3.). На процесі

розроблення вплив змін у вимогах поширюється в першому напрямку, від потреб

до робочих продуктів, а на процесі експлуатації – у зворотному напрямку. Під час

цього виявляють причини виникнення різних неточностей, а потім виносять

рішення про трасування в одному з наведених напрямків.

***43.Засоби та інструменти інженерії вимог до програмного забезпечення.***

**Об’єктно-орієнтована інженерія вимог**

В об’єктно-орієнтованих підходах і методах розробки програмних систем

головним є об'єкт. Для нього задаються вимоги за допомогою варіантів

використання (use case), сценаріїв або прецедентів.

Наведені сценаріями або прецедентами вимоги до системи в UML послідовно

трансформуються до інших сценаріїв, що наближають до логічної та виконуваної

структури системи. Головні їх елементи – сценарії і актори, що задають дії щодо

виконання сценаріїв системи.

***44.Основні задачі в проектуванні архітектури програмної системи.***

**Проектування архітектури ПЗ** проводиться архітектурним стилем,

заснованим на визначенні основних елементів структури – підсистем, компонентів,

об'єктів і зв'язків між ними.

*Архітектура проекту* – високорівневе подання структури системи і

специфікація її компонентів. Архітектура визначає логіку системи через окремі

компоненти системи настільки детально, наскільки це необхідно для написання

коду, а також визначає зв'язки між компонентами. Існують і інші види подання

структур, засновані на проектуванні зразків, шаблонів, сімейств програм і каркасів

програм.

Розділ 1 31

Один з інструментів проектування архітектури – *патерн (шаблон).* Це

типовий конструктивний елемент ПЗ, що задає взаємодію об'єктів (компонентів)

проектованої системи, а також ролі і відповідальності виконавців. Основна мова

опису – UML. Патерн може бути *структурним*, що містить у собі структуру

типової композиції з об'єктів і класів, об'єктів, зв'язків і ін.; *поведінковим*, що

визначає схеми взаємодії класів об'єктів і їх поведінку, задається діаграмами

діяльності, взаємодії, потоків керування й ін.; *погоджувальним*, що відображає

типові схеми розподілу ролей екземплярів об'єктів і способи динамічної генерації

структур об'єктів і класів.

***45.Основні архітектурні моделі, що розробляються для програмної системи.***

*Архітектура системи* – це структурна схема компонентів системи,

взаємодіючих між собою через інтерфейси. Компоненти можуть складатися з

послідовності більш дрібних компонентів та інтерфейсів. Розроблення архітектури

ґрунтується на загальних довідниках, класифікаторах та ін. У ній ідентифікуються

загальні частини системи, у тому числі готові програмні продукти і заново

розроблені компоненти, а також багаторазово використовувані в інших

застосуваннях.

Побудова архітектури системи здійснюється шляхом визначення цілей

системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми,

компоненти або модулі та розроблення її загальної структури.

Основні розв’язки щодо структури системи приймаються групою

архітекторів і аналітиків. Вони передають окремі частини системи для реалізації

невеликим групам розробників.

Архітектуру системи визначають також як множину представлень, кожне з

яких відбиває деякий аспект, що цікавить групу учасників проекту – аналітиків,

проектувальників, кінцевих користувачів та ін. Представлення фіксують проектні

рішення щодо структури і поділу ПС на окремі компоненти та визначення зв’язків

між ними. На ці рішення впливають вимоги до функцій і середовища.

Проектування архітектури системи може проводитися різними методами

(стандартизованим, об’єктно-орієнтованим, компонентним і ін.), кожний з яких

пропонує свій шлях побудови архітектури, а саме, визначення концептуальної,

об’єктної й інших моделей за допомогою відповідних конструктивних елементів

(блок-схем, графів, структурних діаграм тощо).

При застосуванні об’єктно-орієнтованого підходу компонентами є окремі

об’єкти, а процес конструювання об’єктної структури перетворюється в процес

виявлення наявних у ПрО об’єктів і визначення сценарію їхнього виконання і

взаємодії. Стандартизований і об’єктно-орієнтований підходи до проектування

використовують відповідні сформовані технології проектування програмних систем.

***46. Компроміси щодо забезпечення вимог, при виборі архітектури програмної системи***

Моделі архітектури можуть залежати від нефункціональних системних вимог:

  1. Продуктивність. Якщо критичним вимогою є продуктивність системи, слід розробити таку архітектуру, щоб за всі критичні операції відповідало якомога менше підсистем з максимально малим взаємодією між ними. Щоб зменшити взаємодію між компонентами, краще використовувати крупномодульние компоненти, а не дрібні структурні елементи.

   2.Захищеність. У цьому випадку архітектура повинна мати багаторівневу структуру, в якій найбільш критичні системні елементи захищені на внутрішніх рівнях, а перевірка безпеки цих рівнів здійснюється на більш високому рівні.

3.Безпека. У цьому випадку архітектуру слід спроектувати так, щоб за всі операції, що впливають на безпеку системи, відповідало якомога менше підсистем. Такий підхід дозволяє знизити вартість розробки і вирішує проблему перевірки надійності.

   4.Надійність. У цьому випадку слід розробити архітектуру з включенням надлишкових компонентів, щоб можна було замінювати і оновлювати їх, не перериваючи роботу системи.

   5. Зручність супроводу. У цьому випадку архітектуру системи слід проектувати на рівні дрібних структурних компонентів, які можна легко змінювати. Програми, що створюють дані, повинні бути відокремлені від програм, що використовують ці дані. Слід також уникати структури спільного використання даних.

***47. Основні задачі, що вирішуються побудовою структурної схеми системи.***

На першому етапі процесу проектування архітектури система розбивається на декілька взаємодіючих підсистем. На самому абстрактному рівні архітектуру системи можна зобразити графічно за допомогою блок-схеми, в якій окремі підсистеми представлені окремими блоками. Якщо підсистему також можна розбити на кілька частин, на діаграмі ці частини зображуються прямокутниками всередині великих блоків. Потоки даних і / або потоки управління між підсистемами позначається стрілками. Така блок-схема дає загальне уявлення про структуру системи.

Для того щоб підсистеми, складові систему, працювали ефективніше, між ними повинен йти обмін інформацією. Обмін можна організувати двома способами:

   -Всі спільно використовувані дані зберігаються в центральній базі даних, доступній всім підсистем. Модель системи, заснована на спільному використанні бази даних, часто називають моделлю репозиторію.

   -Кожна підсистема має власну базу даних. Взаємообмін даними між підсистемами відбувається за допомогою передачі повідомлень.

***48. Особливості побудови структури системи на основі моделі репозиторію.***

Модель репозиторію:

1.Спільне використання великих обсягів даних ефективно, оскільки не потрібно передавати дані з однієї підсистеми в інші.

 2. Підсистеми повинні бути погоджені з моделлю репозиторія даних. Це завжди призводить до необхідності компромісу між вимогами, що пред'являються до кожної підсистемі. Компромісне рішення може знизити їх продуктивність. Якщо формати даних нових підсистем не підходять під узгоджену модель представлення даних, інтегрувати такі підсистеми складно або неможливо.

  3. Підсистемам, в яких створюються дані, не потрібно знати, як ці дані використовуються в інших підсистемах.

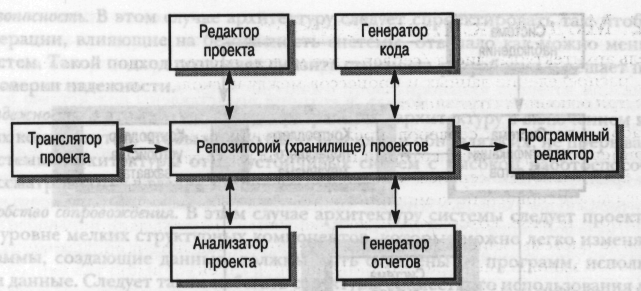
4. Оскільки відповідно до узгодженої моделі даних генеруються великі обсяги інформації, модернізація таких систем проблематична. Переклад системи на нову модель даних буде дорогим і складним, а часом навіть неможливим.

 5. В системах з репозиторієм такі засоби, як резервне копіювання, забезпечення безпеки, управління доступом і відновлення даних, централізовані, оскільки входять в систему управління репозиторієм.

  6. До різних підсистем пред'являються різні вимоги, що стосуються безпеки, відновлення та резервування даних. У моделі репозиторію до всіх підсистемах застосовується однакова політика.

7. Модель спільного використання репозиторію прозора: якщо нові підсистеми сумісні з узгодженою моделлю даних, їх можна безпосередньо інтегрувати в систему.

  8. Складно розмістити репозиторії на декількох машинах, оскільки можуть виникнути проблеми, пов'язані з надмірністю і порушенням цілісності даних.



***49. Особливості побудови структури системи на основі моделі клієнт/сервер.***

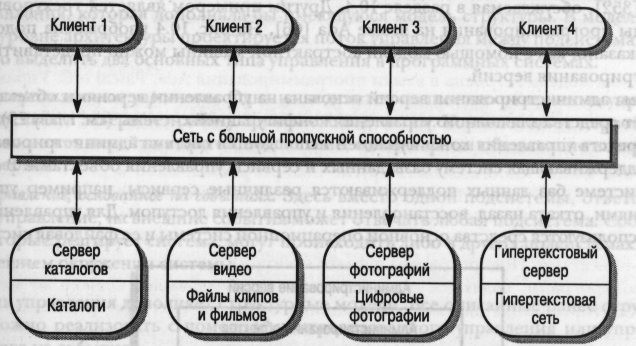
Модель архітектури клієнт / сервер - це модель розподіленої системи, в якій показано розподіл даних і процесів між декількома процесорами. Модель включає три основні компоненти:

 1. Набір автономних серверів, що надають сервіси іншим підсистемам.

  2.Набір клієнтів, які викликають сервіси, що надаються серверами. У контексті системи клієнти є звичайними підсистемами. Допускається паралельне виконання кількох екземплярів клієнтської програми.

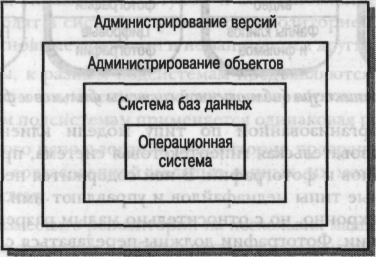
  3.Мережу, за допомогою якої клієнти отримують доступ до сервісів.

Найбільш важлива перевага моделі клієнт / сервер полягає в тому, що вона є розподіленою архітектурою. Її ефективно використовувати в мережевих системах з безліччю розподілених процесорів. У систему легко додати новий сервер і інтегрувати його з іншою частиною системи або ж оновити сервери, не впливаючи на інші частини системи.



***50.Oсобливості побудови структури системи на основі моделі абстрактної машини.***

Модель архітектури абстрактної машини (іноді звана багаторівневої моделлю) моделює взаємодію підсистем. Вона організовує систему у вигляді набору рівнів, кожен з яких надає свої сервіси. Кожен рівень визначає абстрактну машину, машинний мова якої (сервіси, що надаються рівнем) використовується для реалізації наступного рівня абстрактної машини.



***51. Основні моделі керування програмною системою.***

У структурних моделях немає (і не повинно бути) ніякої інформації по управлінню. Однак розробник архітектури повинен організувати підсистеми згідно деякої моделі управління, яка доповнювала б наявну модель структури. У моделях управління на рівні архітектури проектується потік управління між підсистемами.

Можна виділити два основних типи управління:

 1. Централізоване управління. Одна з підсистем повністю відповідає за управління, запускає і завершує роботу решти підсистем.

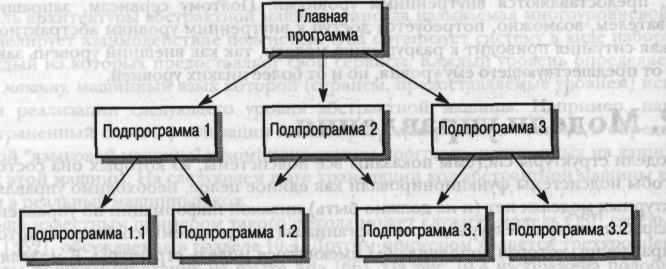
  2. Управління, засноване на подіях. Тут замість однієї підсистеми, відповідальною за управління, на зовнішні події може відповідати будь-яка підсистема. Події, на які реагує система, можуть відбуватися або в інших підсистемах, або в зовнішньому оточенні системи.

***52. Особливості моделі централізованого керування програмною системою.***

У моделі централізованого управління одна з систем призначається головною і управляє роботою інших підсистем. Такі моделі можна розбити на два класи:

Модель виклику-повернення. Це відома модель організації виклику програмних процедур "зверху вниз", в якій управління починається на вершині ієрархії процедур і через виклики передається на більш нижні рівні ієрархії. Дана модель застосовна тільки в послідовних системах.

Модель диспетчера. Застосовується в паралельних системах. Один системний компонент призначається диспетчером і управляє запуском, завершенням і координуванням інших процесів системи. Процес може протікати паралельно з іншими процесами. Модель такого типу застосовна також в послідовних системах, де керуюча програма викликає окремі підсистеми в залежності від значень деяких змінних стану.

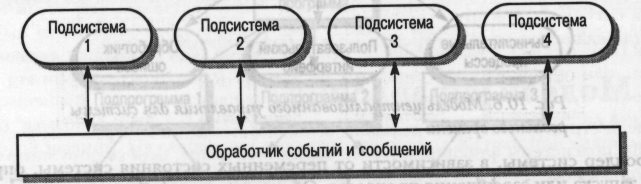




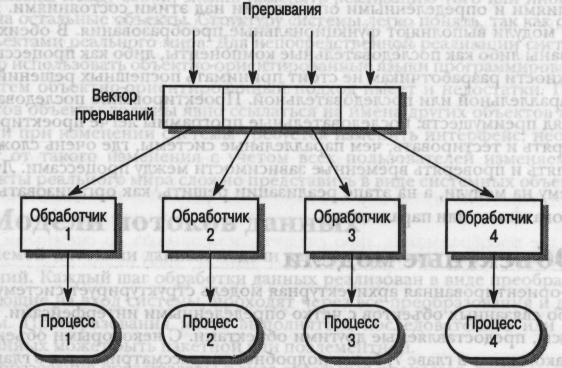
***53. Особливості моделі програмної системи, що керується подіями.***

У моделях централізованого управління, як правило, керування системою визначається значеннями деяких змінних її стану. На противагу таким моделям існують системи, управління якими засноване на зовнішніх подіях.

Моделі передачі повідомлень. У цих моделях подія являє собою передачу повідомлення всім підсистем. Будь підсистема, яка обробляє дану подію, відповідає на нього.

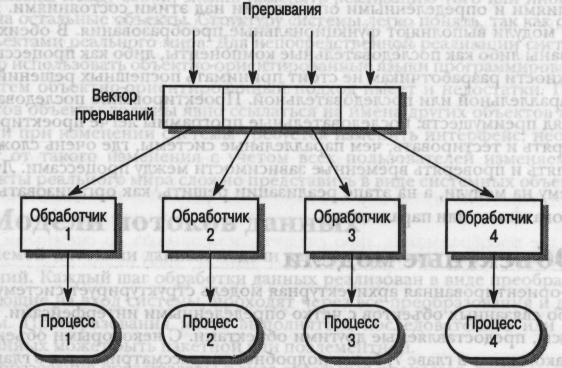


Моделі, керовані перериваннями. Такі моделі зазвичай використовуються в системах реального часу, де зовнішні переривання реєструються обробником переривань, а обробляються іншим системним компонентом.



***54. Особливості моделі програмної системи, що керується перериваннями.***

У моделях централізованого управління, як правило, керування системою визначається значеннями деяких змінних її стану. На противагу таким моделям існують системи, управління якими засноване на зовнішніх подіях.

Моделі, керовані перериваннями. Такі моделі зазвичай використовуються в системах реального часу, де зовнішні переривання реєструються обробником переривань, а обробляються іншим системним компонентом.

У моделі, керованій перериваннями , усі переривання розбиті на групи - типи, що утворять вектор переривань. Для кожного типу переривання є свій оброблювач. Кожен оброблювач реагує на свій тип переривання s запускає свій процес.

***55. Задачі модульної декомпозиції програмної системи***

Після етапу розробки системної структури в процесі проектування слід етап декомпозиції підсистем на модулі.

Поширені дві моделі, використовувані на етапі модульної декомпозиції підсистем:

  Об'єктно-орієнтована модель. Система складається з набору взаємодіючих об'єктів.

  Модель потоків даних. Система складається з функціональних модулів, які отримують на вході дані і перетворять їх деяким чином у вихідні дані. Такий підхід часто називається конвеєрним.

В об'єктно-орієнтованої моделі модулі являють собою об'єкти з власними станами і певними операціями над цими станами. У моделі потоків даних модулі виконують функціональні перетворення.

Згадаємо основні принципи, напрацьовані як рекомендації для декомпозиції складної системи на компоненти або модулі:

− для компоненти має бути чітко визначена мета, щоб можна було перевірити, чи вона її виконує;

− для компоненти має бути чітко визначено всі її входи та виходи;

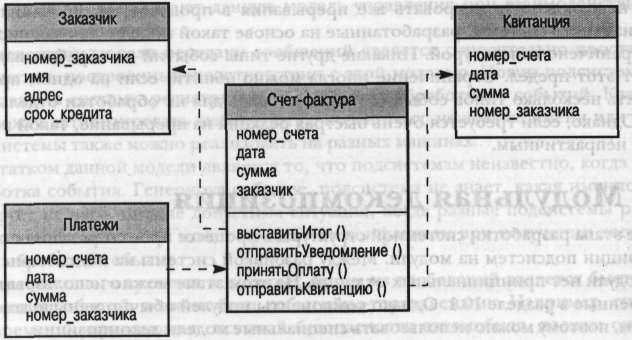
− компоненти мають утворювати ієрархію, кожний рівень якої відповідає рівню абстракції розгляду системи і дозволяє приховувати певні деталі, які буде відпрацьовано на наступних рівнях. Така покрокова деталізація прийняття рішень не стільки розподіляє вирішення складного завдання на кілька вирішень простіших завдань, скільки дозволяє відкласти детальні розв’язання проблем, щоб зосередитися на розв’язанні загальних рішень;

− робота над компонентами може вестися окремими членами команди із застосуванням кількох інструментальних засобів для кількох компонент, що суттєво впливає на ефективність роботи; але при цьому інтерфейси між компонентами мають бути прозорими й узгодженими, щоб інтеграція компонент в єдину структуру була можливою і базувалася на спільному розумінні проблеми. При цьому ключова якість об’єктного підходу − інкапсуляція внутрішніх дій і приховування всіх подробиць, які не стосуються правил використання компоненти − має діяти і для підсистеми як композиції об’єктів.

Враховуючи зазначене вище, можна прийти до висновку, що отримані нами сукупності об’єктів доцільно об’єднувати в підсистеми.

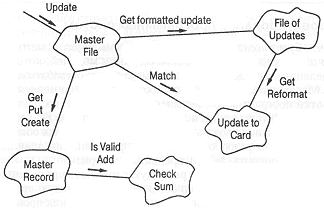
***56. Особливості об’єктної декомпозиції програмної системи***

Об'єктно-орієнтована архітектурна модель структурує систему у вигляді сукупності слабо пов'язаних об'єктів з чітко визначеними інтерфейсами. Об'єкти викликають сервіси, що надаються іншими об'єктами.



Об’єктно-орієнтовна декомпозиція. Можна розділили систему, вибравши як критерій декомпозиції приналежність її елементів до різних абстракцій даної проблемної області. До того як розділяти задачу на кроки типу  (Одержати зміни в відформатованому виді) і  (Додати до контрольної суми), ми повинні визначити такі об'єкти як (Основний файл) і (Контрольна сума), що запозичаються зі словника предметної області.

Досвід показує, що корисніше починати з об'єктної декомпозиції. Такий початок допоможе краще справитися з доданням організованості складності програмних систем. цей об'єктний підхід може використовуватись при описі таких несхожих систем, як комп'ютери, рослини, галактики і суспільні інститути. Об'єктна декомпозиція має трохи надзвичайно важливих переваг перед алгоритмічною. Об'єктна декомпозиція зменшує розмір програмних систем за рахунок повторного використання загальних механізмів, що приводить до істотної економії виразних засобів. Об’єктно-орієнтовані системи більш гнучкі і простіше еволюціонують згодом, тому що їхньої схеми базується на стійких проміжних формах. Дійсно, об'єктна декомпозиція істотно знижує ризик при створенні складної програмної системи, тому що вона розвивається з менших систем, у яких ми вже упевнені. Більш того, об'єктна декомпозиція допомагає нам розібратися в складній програмній системі, пропонуючи нам розумні рішення щодо вибору підпростору великого простору станів.



*Мал. 2. Об’єктно-орієнтована декомпозиція.*

***57. Особливості декомпозиції програмної системи за потоками даних.***

Дані проходять через послідовність перетворень. Кожен крок обробки даних реалізований у вигляді перетворення. Дані, що надходять на вхід системи, проходять через всі перетворення і досягають виходу системи. Перетворення можуть виконуватися послідовно або паралельно. Обробка даних може бути пакетної або поелементної.



Модель потоків даних (Data Flow Model – DFM) використовується для опису процесів обробки даних у системі й містить у собі:

– ієрархічний набір діаграм потоків даних (Data Flow Diagram – DFD);

– опис елементарних процесів, потоків даних, сховищ даних і зовнішніх сутностей.

Кожна DFD відбиває проходження даних через систему залежно від рівня та призначення діаграми. DFD перетворює вхідні потоки даних (входи) у вихідні потоки даних (виходи). Як правило, процеси, що виконують такі перетворення, створюють і використовують дані зі сховища даних.

До об'єктів моделювання системи належать:

1. Функції, які створюються на основі DFM і моделювання взаємозв'язків подій і сутностей для дослідження обробки даних у системі;

2. Події – деякі прикладні дії, які ініціюють процеси для занесення й відновлення даних системи. Подія приводить до виклику процесу і досліджується за допомогою моделювання її впливу на сутності;

3. Дані зображаються спочатку логічною моделлю, потім фізичною, яка відображається у реляційну або об’єктно-орієнтовану БД, залежно від вибраної для проекту СКБД.

У цьому методі програмна система розглядається як перетворювач вхідних потоків у вихідні. Метод потоків даних, як і структурний метод, з успіхом застосовувався при рішенні ряду складних задач, зокрема, у системах інформаційного забезпечення, де існують прямі зв'язки між вхідними і вихідними потоками системи і де не потрібно приділяти особливу увагу швидкодії.

***58 Особливості проблемно-залежних архітектур***

Поряд з основними моделями, використовуються архітектурні моделі, характерні для конкретної предметної області додатка. Ці моделі називаються проблемно-залежними архітектурами.

Можна виділити два типи проблемно-залежних архітектурних моделей:

1. Моделі класів систем. Відображають класи реальних систем, увібравши в себе основні характеристики цих класів. Як правило, архітектурні моделі класів зустрічаються в системах реального часу, наприклад у системах збору даних, моніторингу і т.д.

2. Базові моделі. Більш абстрактні і надають розробникам інформацію по загальній структурі якого типу систем.

Модель компілятора найбільш відомий приклад архітектурної моделі класу систем. Компоненти, з яких складається компілятор, можна організовувати у відповідності з різними архітектурними моделями. Можна застосувати архітектуру потоків даних, в якій таблиця ідентифікаторів служить сховищем спільно використовуваних даних.

Однак такі моделі виявляються менш ефективними, якщо компілятор інтегрований з іншими мовними засобами, наприклад системою редагування структур, інтерактивним відладчиком, програмою підготовки друкованих документів і т.п. У цьому випадку компоненти системи можна організувати у відповідності з моделлю репозиторію.

Базові моделі являють собою ідеалізовану архітектуру, в якій відображені особливості, властиві системам, працюючим в даній предметній області.

Прикладом базової архітектури може служити модель OSI.

Базові моделі зазвичай не розглядаються в якості методів реалізації. Їхнє основне призначення - служити еталоном для порівняння різних систем в якої предметної області, тобто базова модель є стандартом при оцінці різних систем.

***59. Особливості моделі класів систем.***

Моделі класів систем. Відображають класи реальних систем, увібравши в себе основні характеристики цих класів. Як правило, архітектурні моделі класів зустрічаються в системах реального часу, наприклад у системах збору даних, моніторингу і т.д.

Модель компілятора найбільш відомий приклад архітектурної моделі класу систем. Компоненти, з яких складається компілятор, можна організовувати у відповідності з різними архітектурними моделями. Можна застосувати архітектуру потоків даних, в якій таблиця ідентифікаторів служить сховищем спільно використовуваних даних.



Однак такі моделі виявляються менш ефективними, якщо компілятор інтегрований з іншими мовними засобами, наприклад системою редагування структур, інтерактивним відладчиком, програмою підготовки друкованих документів і т.п. У цьому випадку компоненти системи можна організувати у відповідності з моделлю репозиторію.

***60. Особливості моделі базових архітектур.***

Базові моделі являють собою ідеалізовану архітектуру, в якій відображені особливості, властиві системам, працюючим в даній предметній області.

Прикладом базової архітектури може служити модель OSI.

**Модель OSI** ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Open Systems Interconnection Reference Model* - **модель взаємодії**[**відкритих систем**](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D1%82%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_(%D0%86%D0%A2))) - абстрактна модель для мережних комунікацій і розробки [мережевих протоколів](http://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D0%B5%D0%B2%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB). Представляє [рівневий підхід](http://uk.wikipedia.org/wiki/%D0%A8%D0%B0%D1%80_%D0%B0%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D1%96%D1%97) до [мережі](http://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0). Кожен рівень обслуговує свою частину процесу взаємодії. Завдяки такій структурі спільна робота [мережного обладнання](http://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D0%BD%D0%B5_%D0%BE%D0%B1%D0%BB%D0%B0%D0%B4%D0%BD%D0%B0%D0%BD%D0%BD%D1%8F&action=edit&redlink=1) й [програмного забезпечення](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) стає набагато простішою й зрозумілішою.

Зрозуміло, у цей час основним використовуваним протоколом є TCP/IP, розробка якого не була пов'язана з моделлю OSI. **За увесь час існування моделі OSI вона не була реалізована, і, очевидно, не буде реалізована ніколи**.

Сьогодні використовується тільки деяка підмножина моделі OSI. Вважається, що модель занадто складна, а її реалізація займе занадто багато часу.

|  |  |
| --- | --- |
| **Модель OSI** | |
| Дані | Рівень |
| Дані | [**Прикладний**](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B8%D0%B9_%D1%80%D1%96%D0%B2%D0%B5%D0%BD%D1%8C) доступ до мережних служб |
| Дані | [**Представлення**](http://uk.wikipedia.org/w/index.php?title=%D0%A0%D1%96%D0%B2%D0%B5%D0%BD%D1%8C_%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F&action=edit&redlink=1) представлення і кодування даних |
| Дані | [**Сеансовий**](http://uk.wikipedia.org/w/index.php?title=%D0%A1%D0%B5%D0%B0%D0%BD%D1%81%D0%BE%D0%B2%D0%B8%D0%B9&action=edit&redlink=1) керування сеансом зв'язку |
| Блоки | [**Транспортний**](http://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D0%B8%D0%B9_%D1%80%D1%96%D0%B2%D0%B5%D0%BD%D1%8C_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%96_OSI) безпечне та надійне з'єднання «точка - точка» |
| Пакети | [**Мережний**](http://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D0%BD%D0%B8%D0%B9_%D1%80%D1%96%D0%B2%D0%B5%D0%BD%D1%8C&action=edit&redlink=1) визначення маршруту та IP (логічна адресація) |
| Кадри | [**Канальний**](http://uk.wikipedia.org/w/index.php?title=%D0%9A%D0%B0%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D1%80%D1%96%D0%B2%D0%B5%D1%8C&action=edit&redlink=1) MAC та LLC (фізична адресація) |
| Біти | [**Фізичний**](http://uk.wikipedia.org/w/index.php?title=%D0%A4%D1%96%D0%B7%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D1%80%D1%96%D0%B2%D0%B5%D0%BD%D1%8C&action=edit&redlink=1) кабель, сигнали, бінарна передача |

Базові моделі зазвичай не розглядаються в якості методів реалізації. Їхнє основне призначення - служити еталоном для порівняння різних систем в якої предметної області, тобто базова модель є стандартом при оцінці різних систем.

***Практичні:***

***1.Побудувати діаграму варіантів використання програмної системи …***

Діаграми варіантів використання (*usecase diagrams*) використовуються

для відображення сценаріїв використання системи (*usecases*) та

користувачів системи (*actors*), які використовують її функції.

Актори на діаграмі варіантів використання позначаються символом

людини, а варіанти використання – еліпсом. Актори та варіанти

використання поєднуються напрямленою асоціацією (unidirectional

*association*) – стрілкою, що спрямована від актора до варіанта

використання. Також актори можуть поєднуватися з використанням

зв’язків узагальнення. На рис. 1 показано фрагмент діаграми варіантів

використання для інтернет-магазину. Актор «Покупець» при цьому може

виконати сценарій «Замовити товар».

Варіанти використання можуть бути пов’язані між собою трьома

видами зв’язків: узагальненням (*generalization*), розширенням (*extend*

relationship) та включенням (*include* relationship).

Відношення узагальнення (generalization) показують відношення

між загальним і частковим. Наприклад на рис. 2 варіанти використання

«Search by category» та «Search by producer» є частковими випадками

загального варіанта «Search product», тому вони поєднані даним

відношенням. Також дане відношення може використовуватися для

поєднання акторів. Актор «Administrator» може виконувати всі функції

актора «Customer», тобто виступає частковим випадком покупця, але

може виконувати і специфічні операції (варіант використання «Check

db info»).



Відношення включення (include) відображає зв’язок «ціле – частина»,

тобто один варіант завжди в певний момент виконання повністю включає

інший. Для прикладу частиною варіанта використання «Order the product»

є сценарії «Search product» та «Select payment method», оскільки для

того, щоб замовити товар покупець завжди має відшукати його в каталозі

та обрати метод оплати.

Відношення розширення (extend) визначає такий тип відношення,

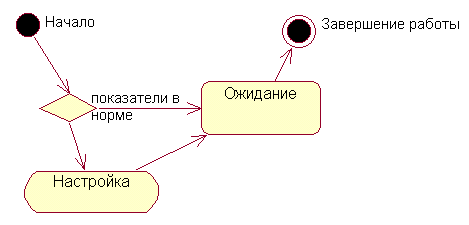
коли один варіант за певних умов повністю використовує інший

(розширює його). Так, наприклад, оператор Інтернет магазину може

видалити товар («Delete product»), знаючи його ідентифікатор, або

провівши попередньо пошук товару («Search product»).

***1.Побудувати діаграму активності клієнта (сервера) програмної системи …***

Activity diagram (діаграми активності)  
  
Це подальший розвиток діаграми станів. Фактично даний тип діаграм може використовуватися й для відбиття станів моделюємого об'єкта, однак, основне призначення Activity diagram у тому, щоб відбивати бізнес-процеси об'єкта. Цей тип діаграм дозволяє показати не тільки послідовність процесів, але й розгалуження й навіть синхронізацію процесів.  
  
Цей тип діаграм дозволяє проектувати алгоритми поведінки об'єктів будь-якої складності, у тому числі може використовуватися для складання блок-схем.  
  
  
Рис. 23.4. Приклад діаграми активності (Activity diagram)