Методичні рекомендації для самостійної роботи

Основи програмної інженерії

2023 |

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «**ФАХОВИЙ КОЛЕДЖ ПРОМИСЛОВОЇ АВТОМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

ЗАТВЕРДЖУЮ
Заступник директора з
навчально-методичної роботи
ФКПАІТ ОНАХТ
з навчально-методичної роботи
_____ Вікторія ОКСАНІЧЕНКО
_____._____.20___ року

Методичні рекомендації для самостійної роботи

2.01 **ОСНОВИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ**
(шифр за ОПП і назва навчальної дисципліни)

галузі знань _____12 «Інформаційні технології»_____
(шифр і назва галузі знань)

спеціальності_____121 «Інженерія програмного забезпечення»_____
(шифр і назва спеціальності)

освітня програма_____Інженерія програмного забезпечення_____
(назва освітньої програми)

Одеса 2023

Методичні рекомендації для самостійної роботи студентів з дисципліни «Основи програмної інженерії» для підготовки фахових молодших бакалаврів за спеціальністю 121 «Інженерія програмного забезпечення».

Укладач: викладач ФКПАІТ ОНАХТ Тетяна КОСТИРЕНКО

Конспект лекцій затверджено на засіданні циклової комісії «Комп'ютених наук та інженерія програмного забезпечення» ФКПАІТ ОНАХТ
Протокол від ___._____.20___ року, № ____

Голова циклової комісії
_____Тетяна КОСТИРЕНКО
(підпис) (власне ім'я та прізвище)
___._____.20___ року

Зміст **Стор.**

Блок змістових модулів 1: Інженерні основи програмного забезпечення

Змістовий модуль 1.1: Огляд інженерії програмного забезпечення

Тема 1 Технологія розробки програмного забезпечення

(2 години)

Мета: Ознайомитись з поняттям «технологія». Розглянути основні вимоги до технології розробки ПЗ, розглянути проблеми розробки складних ПЗ.

План:

1. Поняття «Технологія».
2. Вимоги до технології розробки ПЗ.
3. Проблеми розробки складних ПЗ

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект матеріалу.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю:

1. Що таке технологія?
2. Яким вимогам повинна задовольняти ТРПЗ?
3. Що ускладнює процес розробки ПЗ?

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

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

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

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

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

Під **технологією розробки програмного забезпечення** (ТРПЗ) розуміється сукупність узагальнених і систематизованих знань, або наука про оптимальні способи (прийоми) проведення процесу розробки програмного 12 забезпечення, що забезпечує в заданих умовах отримання програмної продукції із заданими властивостями.

Технологія розробки програмного забезпечення являє собою інженерний підхід до розробки програмних засобів ЕОМ, що охоплює методологію програмування,

проблеми забезпечення надійності програм, оцінки робочих характеристик і якості проектів.

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

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

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

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

1. Необхідна стандартизація мов проектування програм, оформлення та випробування програмних модулів, а також гарантії їх якості. Це дозволить значно скоротити дублюючі розробки, впровадити складальне програмування і вести накопичення на підприємствах і в країні високоякісного програмного продукту для його багаторазового використання як типових комплектуючих виробів.

2. Вести постійний контроль і забезпечення якості програм.

3. Програми не повинні містити неперевірених шляхів і ситуацій функціонування, які призводять до несподіваних результатів.

4. Користувачеві або покупцю програм необхідно дати чітке уявлення про можливості даної програми і технологічних умовах експлуатації, при яких гарантуються певні функції і якості.

5. Технологія розробки програмного забезпечення повинна забезпечувати відторгнення програмного виробу від його розробника, тобто людський фактор у програмуванні має бути зведений до мінімуму.

6. Технологія розробки програмного забезпечення та засоби її підтримки (автоматизації) повинні забезпечувати цілеспрямовану роботу, насамперед колективу програмістів, а не окремих особистостей. Вона повинна спонукати колектив працювати тільки правильно і злагоджено; повинна автоматично блокувати будь-які не санкціоновані технологією дії.

7. Необхідно вести акуратне документування всіх етапів розробки. Документація повинна також заноситися і зберігатися на магнітних носіях. Доступ до цієї інформації має бути відкритим, простим і автоматизованим.

8. Робота користувача повинна забезпечуватися розвиненою інформаційнодовідковою системою.

9. Засоби автоматизації технології повинні охоплювати всі етапи роботи колективу програмістів.

10.Технологія розробки програмного забезпечення повинна бути простою в освоєнні, із засобами підказки, що автоматично включаються.

11.Технологія розробки програмного забезпечення повинна мати засоби автоматичної фіксації в хронологічному порядку всіх дій, які виконуються в процесі колективного виготовлення програмного виробу - повинні вестися і зберігатися в системі журнали (протоколи, щоденники) розробки. Ці засоби повинні дозволяти відновлювати будь-який стан процесу розробки на будь-якому інтервалі виготовлення програмного продукту.

Проблеми розробки складних програмних систем

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

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

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

− складність формального визначення вимог до програмних систем;

− відсутність задовільних засобів опису поведінки дискретних систем із великою кількістю станів при недетермінованій послідовності вхідних впливів;

− колективна розробка;

− необхідність збільшення ступеня повторюваності кодів.

Складність визначення вимог до програмних систем.

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

Відсутність задовільних засобів формального опису поведінки дискретних систем.

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

Колективна розробка.

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

Необхідність збільшення ступеня повторюваності кодів.

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

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

Блок змістових модулів №2 Основи моделювання програмного забезпечення

Змістовий модуль 2.1 Структурний підхід до проектування ПЗ

Тема 2 Функціональні вимоги до ПЗ

(4 години)

Мета: Ознайомити студентів з поняттям функціональних вимог.

План:

1. Функціональні вимоги.
2. Типи функціональних вимог
3. Створення функціональних вимог.

Домашнє завдання:

1. Ознайомитись з теоретичним матеріалом.
2. Зробити опорний конспект лекції.
3. Відповісти на контрольні питання.

Питання для самоконтролю:

1. Що таке вимоги до ПЗ?
2. Наведіть приклад функціональних вимог до ПЗ.
3. Як документуються функціональні вимоги до ПЗ?
4. Які типи функціональних вимог ви запам'ятали?
5. Чим відрізняються функціональні та не функціональні вимоги?

Функціональні вимоги (functional requirements) визначають функціональність ПЗ, яку розробники повинні забезпечити, щоб користувачі змогли виконати свої завдання в межах бізнес-вимог.

Інколи їх називають вимоги поведінки (behavioral requirements), вони містять положення з традиційним «повинна». Наприклад, «Система повинна по електронній пошті відправляти користувачу підтвердження замовлення». Функціональні вимоги описують, що розробнику необхідно реалізувати.

Функціональні вимоги **документуються** в специфікації вимог до ПЗ (software requirement specification, SRS), де описується очікувана поведінка системи. Це може бути документ, база даних, таблиця з вимогами, сховище даних в комерційному інструменті управління вимогами, або навіть набір карток для невеликого проекту.

Специфікація вимог до ПЗ використовується при розробці, тестуванні, гарантії якості продукту, управлінні проектом і зв'язаних з проектом функціях. Окрім функціональних в специфікації містяться нефунціональні вимоги, де описані цілі і атрибути якості.

Типи функціональних вимог

Ось найпоширеніші типи функціональних вимог:

- Регламент підприємницької діяльності

- Вимоги до сертифікації
- Вимоги до звітності
- Адміністративні функції
- Рівні авторизації
- Відстеження аудиту
- Зовнішні інтерфейси
- Управління даними
- Законодавчі та нормативні вимоги

Створення функціональних вимог:

Створюючи функціональні вимоги, важливо мати на увазі, що вони повинні бути конкретними, вимірними, досяжними, релевантними та обмеженими у часі (SMART). Іншими словами, ваші функціональні вимоги повинні:

- Конкретизуйте, що повинна робити система
- Будьте вимірними, щоб ви могли визначити, чи система це робить
- Будьте досяжними протягом встановленого вами терміну
- Будьте відповідними вашим бізнес-цілям
- Прив’яжіть себе до часу, щоб відстежувати прогрес

Дотримуючись цих вказівок, ви можете бути впевнені, що ваші функціональні вимоги чіткі та допоможуть вашій команді розробників створити правильний продукт.

Приклади:

Щоб краще зрозуміти функціональні вимоги, розглянемо кілька прикладів.

Приклад №1

: Користувач повинен мати можливість увійти в систему, використовуючи своє ім’я користувача та пароль.

У цьому прикладі функція — «вхід», а поведінка — «Система дозволить користувачеві ввійти за допомогою свого імені користувача та пароля».

Приклад №2

: система розраховує податок із продажу для покупки користувача.

У цьому прикладі функція — «обчислити податок із продажу», а поведінка — «Система обчислить податок із продажу, помноживши ціну покупки на ставку податку».

Приклад №3

: система надішле користувачеві електронний лист із підтвердженням після того, як він успішно розмістить замовлення.

У цьому прикладі функція – «надіслати електронний лист із підтвердженням», а поведінка – «Система надішле електронний лист із підтвердженням користувачеві після того, як він успішно розмістить замовлення».

Як бачите, функціональні вимоги — це конкретні твердження про те, що повинна робити система. Вони відрізняються від нефункціональних вимог, які

визначають, як система працює всередині (наприклад, продуктивність, безпека тощо).

Створюючи функціональні вимоги, важливо мати на увазі, що вони повинні бути конкретними, вимірними, досяжними, релевантними та обмеженими у часі (SMART). Дотримуючись цих вказівок, ви можете бути впевнені, що ваші функціональні вимоги чіткі та допоможуть вашій команді розробників створити правильний продукт.

Чим функціональні вимоги відрізняються від нефункціональних?

Функціональні вимоги, як випливає з назви, описують функції системи, яка буде розроблена. Це опис того, якою буде система і як вона функціонуватиме для задоволення потреб користувачів. Вони забезпечують чіткий опис того, як система має реагувати на конкретну команду, функції та те, що очікують користувачі.

Нефункціональні вимоги пояснюють обмеження та обмеження системи, що проектується. Ці вимоги не впливають на функціональність програми. Крім того, існує звичайна практика підкласифікації нефункціональних вимог на різні категорії, наприклад:

- Інтерфейс користувача
- Надійність безпеки
- Продуктивність
- Технічне обслуговування
- Стандарти

Підкласифікація нефункціональних вимог є хорошою практикою. Це допомагає під час створення контрольного списку вимог, які мають відповідати створеній системі.

Нефункціональні вимоги такі ж важливі, як і функціональні. Якщо функціональні вимоги визначають, що повинна робити система, то нефункціональні вимоги описують, як вона буде це робити. Наприклад, нова програма надасть нам остаточний список усіх підключених користувачів. Це частина функціональних вимог. Якщо у вимозі зазначено, що система працюватиме лише в системах Windows і Linux, це буде частиною нефункціональних вимог.

Єдина відмінність між ними полягає в тому, що система не може функціонувати без задоволення всіх функціональних вимог. З іншого боку, система дасть вам бажаний результат, навіть якщо вона не задовольняє нефункціональним вимогам.

Тема 3 Нефункціональні вимоги до програмного забезпечення

(4 години)

Мета: Ознайомитись з поняттям не функціональні вимоги.

План:

1. Нефункціональні вимоги
2. Категорії нефункціональних вимог
3. Переваги не функціональних вимог
4. Недоліки не функціональних вимог
5. Приклади не функціональних вимог

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю

1. Що таке нефункціональні вимоги?
2. Які категорії нефункціональних вимог виділяють?
3. Які є переваги нефункціональних вимог?
4. Які є недоліки нефункціональних вимог?
5. Що таке збір нефункціональних вимог?

Нефункціональні вимоги (NFR) — це обмеження, накладені на систему, які визначають її атрибути якості. Зазвичай вони позначаються такими прикметниками, як безпека, продуктивність і масштабованість. Нефункціональні вимоги важливі, оскільки вони допомагають забезпечити відповідність системи потребам користувача.

Категорії нефункціональних вимог

Нефункціональні вимоги можна розділити на дві категорії:

1. **Атрибути якості**: Це характеристики системи, які визначають її загальну якість. Приклади атрибутів якості включають безпеку, продуктивність і зручність використання.
2. **Обмеження**: Це обмеження, накладені на систему. Приклади обмежень включають час, ресурси та середовище.

Переваги нефункціональних вимог

Є кілька переваг нефункціональних вимог:

1. Вони допомагають переконатися, що система відповідає потребам користувача.
2. Вони допомагають переконатися, що система відповідає меті.
3. Вони допомагають гарантувати, що система є масштабованою, безпечною та надійною.
4. Вони допомагають забезпечити зручність використання та обслуговування системи.

Недоліки нефункціональних вимог

Є кілька недоліків нефункціональних вимог:

1. Їх важко зрозуміти та реалізувати.
2. Вони можуть бути трудомісткими та дорогими для перевірки.
3. Вони можуть вплинути на функціональність системи, якщо їх не впровадити належним чином.

Нефункціональні вимоги проти функціональних вимог

Функціональні вимоги, як випливає з назви, описують функції системи, яку необхідно проектувати. Це опис того, якою буде система і як вона буде функціонувати для задоволення потреб користувачів. Вони надають чіткий опис того, як система повинна реагувати на певну команду, функції та те, що очікують користувачі.

Нефункціональні вимоги пояснюють обмеження системи, що проектується. Ці вимоги не впливають на функціональність програми. Крім того, існує звичайна практика підкласифікації нефункціональних вимог на різні категорії:

- Інтерфейс користувача
- Надійність безпеки
- Продуктивність
- Технічне обслуговування
- Стандарти

Підкласифікація нефункціональних вимог є хорошою практикою. Це допомагає під час створення контрольного списку вимог, які мають відповідати створеній системі.

Нефункціональні вимоги такі ж важливі, як і функціональні. Якщо функціональні вимоги визначають, що повинна робити система, то нефункціональні вимоги описують, як вона буде це робити. Наприклад, нова програма надасть нам остаточний список усіх підключених користувачів. Це частина функціональних вимог. Якщо у вимозі зазначено, що система працюватиме лише в системах Windows і Linux, це буде частиною нефункціональних вимог.

Єдина відмінність між ними полягає в тому, що система не може функціонувати без задоволення всіх функціональних вимог. З іншого боку, система дасть вам бажаний результат, навіть якщо вона не задовольняє нефункціональним вимогам.

Приклади нефункціональних вимог

Ось кілька прикладів нефункціональних вимог:

1. **Безпека:** Система повинна бути захищена від несанкціонованого доступу.
2. **Продуктивність**: Система повинна бути здатна обслуговувати необхідну кількість користувачів без будь-якого зниження продуктивності.
3. **Масштабованість**: Система повинна мати можливість масштабування за потреби.
4. **наявність**: Система повинна бути доступною, коли це необхідно.

5. **Технічне обслуговування**: Система повинна бути простою в обслуговуванні та оновленні.
6. **Портативність**: Система повинна мати можливість працювати на різних платформах з мінімальними змінами.
7. **Надійність**: Система повинна бути надійною і відповідати вимогам користувача.
8. **Usability**: Система повинна бути простою у використанні та зрозумілою.
9. **Сумісність**: Система повинна бути сумісна з іншими системами.
10. **Відповідність**: Система має відповідати всім чинним законам і нормам.

Нефункціональні вимоги є важливими для будь-якої системи. Вони допомагають переконатися, що система відповідає потребам користувача та здатна функціонувати за призначенням. Перед проектуванням і розробкою системи важливо ретельно розглянути всі нефункціональні вимоги.

Що таке збір нефункціональних вимог?

Збір нефункціональних вимог — це процес ідентифікації та документування нефункціональних вимог до системи. Це можна зробити за допомогою інтерв'ю, опитування, фокус-груп або іншими методами. Після того, як нефункціональні вимоги зібрано, їх можна проаналізувати та визначити пріоритети.

Процес збору нефункціональних вимог є важливою частиною розробки системи. Це допомагає переконатися, що всі необхідні вимоги визначені та що їм приділено відповідний рівень уваги. Без ретельного процесу збору нефункціональних вимог було б важко розробити систему, яка б відповідала потребам користувача.

Що таке методи виявлення нефункціональних вимог?

Методи виявлення нефункціональних вимог використовуються для ідентифікації та документування нефункціональних вимог до системи. Існує багато різних методів, які можна використовувати, наприклад, інтерв'ю, опитування, фокус-групи чи інші методи. Після того, як нефункціональні вимоги зібрано, їх можна проаналізувати та визначити пріоритети.

Процес виявлення нефункціональних вимог є важливою частиною розробки системи. Це допомагає переконатися, що всі необхідні вимоги визначені та що їм приділено відповідний рівень уваги. Без ретельного процесу визначення нефункціональних вимог було б важко розробити систему, яка б відповідала потребам користувача.

Що таке нефункціональний аналіз вимог?

Аналіз нефункціональних вимог — це процес аналізу нефункціональних вимог до системи. Це можна зробити, переглянувши вимоги, оцінивши їх і розставивши пріоритети. Мета аналізу нефункціональних вимог полягає в тому, щоб переконатися, що всі необхідні вимоги визначені та що їм приділено відповідний рівень уваги.

Аналіз нефункціональних вимог є важливою частиною розробки системи. Це допомагає переконатися, що всі необхідні вимоги визначені та що їм приділено відповідний рівень уваги. Без ретельного аналізу нефункціональних вимог було б важко розробити систему, яка б відповідала потребам користувача.

Найкращі практики для написання нефункціональних вимог

Під час написання нефункціональних вимог слід дотримуватися кількох найкращих практик. До них належать:

1. Переконайтеся, що вимоги чіткі та лаконічні.
2. Будьте конкретні щодо того, що потрібно.
3. Уникайте використання жаргону.
4. Використовуйте просту мову.
5. Переконайтеся, що вимоги досяжні.
6. Будьте реалістами щодо того, чого можна досягти.
7. Розставте вимоги за пріоритетністю.
8. Зберігайте вимоги гнучкими.
9. За потреби перегляньте та відредагуйте вимоги.
10. Отримайте відгук від зацікавлених сторін щодо вимог.

Нефункціональні вимоги є важливою частиною будь-якого проекту розробки системи. Дотримуючись цих найкращих практик, ви можете переконатися, що ваші нефункціональні вимоги є чіткими, лаконічними та досяжними.

Тема 4 Документування функціональних та не функціональних вимог до ПЗ

(4 години)

Мета: Ознайомитись зі способами документування функціональних та не функціональних вимог.

План:

1. Що таке специфікація вимог?
2. Стандарти для написання вимог?
3. Типи специфікацій вимог.
4. Характеристики документа специфікації вимог до програмного забезпечення.
5. Основні компоненти SRS.

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю

1. Що таке системні вимоги?
2. Що таке функціональні та нефункціональні вимоги?
3. Які переваги наявності специфікації вимог?

4. Стандарти для написання вимог?
5. Які характеристики документа специфікації вимог до програмного забезпечення ви занєте?
6. Перелічіть основні компоненти SRS.

Що таке специфікація вимог?

Специфікація вимог, також відома як документація, — це процес запису всіх вимог системи та користувача у вигляді документа. Ці вимоги мають бути чіткими, повними, вичерпними та послідовними.

Під час захоплення ми збираємо всі вимоги з різних джерел. Під час аналізу та переговорів ми аналізуємо та розуміємо ці вимоги. Тепер ми повинні підготувати офіційний документ, що пояснює ці вимоги. Ось що таке специфікація вимог. Якщо бути точним, то це процес чіткого та точного документування всіх потреб і обмежень користувача та системи.

Що таке системні вимоги?

Системні вимоги можна назвати розширеною версією вимог користувача. Системні вимоги служать початковою точкою для будь-якої нової системи. Ці вимоги є детальним описом вимог користувача, яким має задовольняти система.

Що таке вимога до користувача?

Вимоги користувача - це комбінація функціональних і нефункціональних вимог. Ці вимоги до користувача мають бути розроблені таким чином, щоб вони були легко зрозумілі користувачам, які не мають жодних технічних знань. Отже, вони повинні бути написані природною мовою з використанням простих таблиць, форм і діаграм. Крім того, переконайтеся, що в документі немає подробиць про дизайн системи, програмне забезпечення чи офіційні позначки.

Що таке функціональні та нефункціональні вимоги?

Функціональні вимоги, як випливає з назви, описують функції системи, що проектується. Це опис того, якою буде система і як вона функціонуватиме для задоволення потреб користувачів. Вони забезпечують чіткий опис того, як система має реагувати на конкретну команду, функції та те, що очікують користувачі.

Нефункціональні вимоги пояснюють обмеження та обмеження системи, що проектується. Ці вимоги не впливають на функціональність програми. Крім того, існує поширена практика підкласифікації нефункціональних вимог на різні категорії, наприклад

1. Інтерфейс користувача
2. Надійність безпеки
3. Продуктивність
4. Технічне обслуговування
5. Стандарти

Підкласифікація нефункціональних вимог є хорошою практикою. Це допомагає під час створення контрольного списку вимог, які мають відповідати створеній системі.

Нефункціональні вимоги настільки ж важливі, як і функціональні вимоги. Якщо функціональні вимоги визначають, що повинна робити система, то нефункціональні вимоги описують, як система це буде робити. Наприклад, нова програма надасть нам остаточний список усіх підключених користувачів. Це частина функціональних вимог. Якщо вимога говорить, що система працюватиме лише в системах Windows і Linux, це буде частиною нефункціональних вимог.

Єдина відмінність між ними полягає в тому, що система не може функціонувати без задоволення всіх функціональних вимог. З іншого боку, система дасть вам бажаний результат, навіть якщо вона не задовольняє нефункціональним вимогам.

Які переваги наявності специфікації вимог?

Є багато переваг наявності специфікації вимог. Деякі з них наведено нижче:

Допомагає переконатися, що всі зацікавлені сторони мають спільне розуміння системи, яка має бути розроблена. Це дозволяє уникнути непорозумінь на наступних етапах розробки.

Служить орієнтиром для всіх зацікавлених сторін у процесі розробки.

Допомагає виявити будь-які прогалини у вимогах на ранній стадії.

Зменшує загальну вартість і час розробки, оскільки дозволяє уникнути переробки через зміни вимог.

Стандарти для написання вимог?

EARS тут буде ефективною методологією. Це означає простий підхід до синтаксису вимог. У цьому методі ми пишемо чіткою, короткою та зрозумілою мовою. Це покращує весь робочий процес розробки вимог і спрощує роботу, роблячи речі досить легкими для розуміння.

Щоб досягти цього, ось деякі принципи, які слід пам’ятати під час написання вимог. Вони включають:

- Кожна вимога має бути у формі повного речення. Не слід використовувати маркери, скорочення, абревіатури чи модні слова. Намагайтеся складати короткі, прямі та повні речення.
- Переконайтеся, що кожна вимога має відповідний підмет, присудок і дієслово. Темою буде тип користувача або система, про яку ми говоримо. Предикатом будуть умови, дії або бажані результати, які ми очікуємо. Ми повинні використовувати такі слова, як «буде», «буде» та «повинен», щоб висловити якусь необхідність, і слова на кшталт «можеться», щоб висловити необов’язковість у вимозі.
- Кожна вимога повинна ефективно пояснювати кінцевий результат, якого ми бажаємо від системи.

- Крім того, вимога повинна описувати якість, яку ми очікуємо від системи. Це допомагає, коли ми оцінюємо кінцевий результат і перевіряємо, чи правильно реалізована вимога чи ні.

Типи специфікацій вимог:

Існує багато різновидів специфікацій вимог. Вони включають специфікації функціональних вимог (FRS), специфікацію вимог до продуктивності (PRS), специфікацію вимог до конфігурацій (CRF), специфікацію бізнес-вимог (BRS), специфікацію вимог до надійності (RRF), специфікацію вимог до сумісності (CRF) і специфікацію вимог до програмного забезпечення (SRS).).

Характеристики функціональних вимог: Специфікація функціональних вимог (FRS) — це документ, який фіксує функції, які повинна виконувати система. Він містить усі функції, приміщення, заходи безпеки та іншу відповідну інформацію. Простіше кажучи, FRS — це документ, який містить усе, що повинна робити певна система.

Специфікації вимог до продуктивності: Специфікація вимог до продуктивності (PRS) — це документ, який фіксує всі пов’язані з продуктивністю аспекти системи. Це включає в себе час відгуку, пропускну здатність даних, ефективність, масштабованість тощо. В основному все, що можна кількісно оцінити та покращити, підпадає під категорію PRS.

Специфікація вимог до конфігурацій: Специфікація вимог до конфігурації (CRS) — це документ, який фіксує всю інформацію, пов’язану з конфігурацією системи. Це включає такі деталі, як підтримувані платформи, залежності програмного/апаратного забезпечення, мінімальні системні вимоги тощо.

Специфікації бізнес-вимог: Специфікація бізнес-вимог (BRS) — це документ, який охоплює всі бізнес-аспекти системи. Це включає такі функції, як керування користувачами, безпека, цілісність даних тощо. Загалом, усе, що впливає на бізнес-операції системи, підпадає під категорію BRS.

Специфікації вимог до надійності: Специфікація вимог до надійності (RRF) — це документ, який фіксує всю інформацію, пов’язану з надійністю системи. Це включає такі аспекти, як час безвідмовної роботи, час відновлення, рівень помилок тощо.

Специфікації вимог до сумісності: Специфікація вимог щодо сумісності (CRF) — це документ, який фіксує всю інформацію, пов’язану із сумісністю системи. Це включає такі аспекти, як підтримувані платформи, залежності програмного/апаратного забезпечення, мінімальні системні вимоги тощо.

Специфікації вимог до програмного забезпечення: Специфікація вимог до програмного забезпечення (SRS) — це документ, який охоплює всі пов’язані з програмним забезпеченням аспекти системи. Це включає такі аспекти, як функціональність, продуктивність, масштабованість тощо. В основному все, що впливає на роботу програмного забезпечення системи, підпадає під категорію SRS.

Специфікація вимог до програмного забезпечення проти специфікації бізнес-вимог:

Люди іноді змішують поняття програмного забезпечення та специфікації бізнес-вимог. Насправді вони обидва досить різні.

Основна відмінність між специфікацією вимог до програмного забезпечення та специфікацією бізнес-вимог полягає в тому, що перша фіксує всю інформацію, пов'язану з програмним забезпеченням, тоді як остання фіксує всю інформацію, пов'язану з бізнесом.

Специфікація бізнес-вимог (BRS)	**Специфікація вимог до програмного забезпечення (SRS)**
Це офіційний документ, який описує різні вимоги, надані клієнтом/зацікавленими сторонами.	Він визначає функціональні та нефункціональні вимоги до програмного забезпечення.
Він випливає з вимог і взаємодії клієнта.	Він походить від специфікації бізнес-вимог (BRS).
Його створює бізнес-аналітик.	Його створює системний аналітик, або системний архітектор, або бізнес-аналітик.
Він описує функціональні характеристики програмного забезпечення на дуже високому рівні.	Він також на високому рівні описує як технічні, так і функціональні характеристики програмного забезпечення.
Він стосується вимог бізнесу.	Він має справу з ресурсами, які надає компанія.
Він визначає потреби клієнта. Документ використовується від початку до кінця проекту.	Він описує, як функціонує бізнес під час використання програмного забезпечення чи програми.
Таблиці та випадки використання не включені.	Таблиці та випадки використання включені.

Характеристики документа специфікації вимог до програмного забезпечення:

Точний – Мета документа ЄСВ – бути простим для розуміння. Нічого не повинно бути незрозуміло, щоб не було суперечок між зацікавленими сторонами.

Вимірний – Вимоги у вашому документі SRS мають бути вимірними, тому готовий продукт може бути валідований і сертифікований відповідно до вимог.

Цілковита – Документ SRS має містити достатньо інформації для вашої команди розробників і тестувальників, щоб завершити продукт і переконатися, що він відповідає вимогам користувача без помилок.

Здійсненно – Вимоги мають відображати фактичний стан справ, включаючи вартість, терміни та технологію. Вони не повинні залежати від майбутніх технологічних досягнень.

гнучкий – Оскільки обставини на робочому місці можуть змінюватися, ваш документ ЄСВ має бути достатньо адаптованим, щоб відповідати змінам. Не включайте зайвий матеріал у кілька розділів, які доведеться оновлювати щоразу, коли буде зміна.

Перевіряється – Кожен член команди розробників повинен мати доступ до документа, щоб мати змогу звертатися до нього так часто, як це буде потрібно. Оскільки вимоги мають бути чіткими, членам команди не потрібна додаткова інформація. Усі вони мають бути доступні в документі ЄСВ.

Послідовний – Вимоги мають бути сумісними. Не повинно бути суперечностей між частинами вашого документу вимоги. Документ має бути структурований послідовно, а термінологія повинна використовуватися однаковим чином.

Відсутність обмежень щодо впровадження – Загалом документ ЄСВ має бути достатньо детальним, щоб виконати роботу, але не настільки конкретним, щоб зупинити розробку. Багато розробок базується на сторонніх службах, які розробники не контролюють.

точний – Вимоги, зазначені в документах, мають бути дуже точними, щоб уникнути будь-якої плутанини. У них не повинно бути жодних лазівок, двозначностей, суб'єктивності, відмінних ступенів чи порівнянь.

Основні компоненти SRS:

Основні розділи специфікації вимог до програмного забезпечення:

Бізнес драйвери – У цьому розділі описано причини, чому клієнт хоче побудувати систему. Цей розділ також містить проблеми, з якими клієнт стикається з поточною системою, і можливості, які надасть нова система.

Бізнес-модель – У цьому розділі обговорюється бізнес-модель, яку система має підтримувати. Він також містить різні інші деталі, як-от організаційний і бізнес-контекст, основні бізнес-функції та діаграми процесів системи.

Функціональні та системні вимоги – У цьому розділі зазвичай детально описуються вимоги, організовані в ієрархічній структурі. Функціональні вимоги знаходяться на верхньому рівні, а детальні системні вимоги перераховані як підпункти.

Випадки використання системи – Цей розділ містить діаграму варіантів використання Уніфікованої мови моделювання (UML), яка пояснює всі ключові зовнішні об'єкти, які взаємодіятимуть із системою, і різні варіанти використання, які вони повинні виконувати.

Технічні вимоги – У цьому розділі обговорюються всі нефункціональні вимоги, які складають технічне середовище, і технічні обмеження, в яких працюватиме програмне забезпечення.

Системні якості – У цьому розділі визначаються численні якості системи, такі як надійність, придатність до обслуговування, безпека, масштабованість, доступність і зручність обслуговування.

Обмеження та припущення – У цьому розділі описані всі обмеження, накладені на дизайн системи з точки зору клієнта. Тут також обговорюються різні припущення команди інженерів щодо того, чого очікувати під час розробки.

Критерії прийняття – Детально про всі умови, які повинні бути виконані перед передачею системи кінцевим споживачам, обговорюються в цьому розділі.

Структура СГД:

1. Введення -

У вступі пояснюється загальне значення SRS, її сфера застосування для вашої команди та її структура.

1.1. Мета

Тут поясніть мету та структуру програмної документації SRS: типи вимог, які будуть розглянуті, а також персонал, який використовуватиме його.

Нехай цей розділ буде коротким: достатньо 1-2 абзаців.

1.2. Цільова аудиторія

Ви можете глибоко заглибитися та пояснити, як зацікавлені сторони та команди працюватимуть із SRS, а також взяти участь у її розробці. Зазвичай це власники продуктів, інвестори, бізнес-аналітики, розробники, іноді тестувальники та робочий персонал. Уся структура визначається вашим підходом до розробки програмного забезпечення та організаційною структурою команди.

1.3. Цільове використання

Опишіть, у яких ситуаціях ваша команда використовуватиме SRS. Зазвичай він використовується в таких випадках:

проектування та мозковий штурм нових функцій

планування тривалості проекту, спринтів, оцінка витрат

оцінка ризиків

моніторинг та вимірювання успішності команди

конфліктні ситуації, коли залучені сторони мають різні бачення добре виконаного продукту.

1.4. Область застосування

Ця частина охоплює сферу застосування продукту, тому вам потрібно дати короткий огляд системи – її основне призначення, функції та положення. Це можна порівняти з тим, як ви пояснюєте продукт на зустрічі зацікавлених сторін, за винятком того, що дозволяється глибше заглиблюватися в технічні особливості.

Цей розділ повинен описати:

Усі типи користувачів, від яких очікується взаємодія з системою

Усі важливі частини архітектури

1.5 Визначення та акроніми

Вищезазначені компоненти складають визначення. Визначення надають інформацію про функцію, базові технології, цільові персони, суб'єктів господарювання (користувачів, клієнтів, посередників) і зацікавлених сторін. Ви можете використовувати абревіатуру, щоб швидше написати свій ЄСВ, якщо захочете це зробити. Документ можна буде прочитати, доки він міститься в таблиці визначень.

У вашому документі команда часто використовує певні слова. Усунути будь-які потенційні непорозуміння, дозволити новим розробникам приєднатися та вирішити конфліктні ситуації – усе це стане легше, якщо ви з'ясуєте значення цих слів.

2. Загальний опис

У другій частині ви описуєте читачам основні функції продукту, цільових користувачів і масштаби системи. Цей опис зосереджується лише на ключових функціях і архітектурі програмного забезпечення, не вдаючись до особливостей додаткових компонентів і з'єднань.

2.1 Потреби користувача

Ця частина є питанням вибору, тому деякі організації вирішують не включати її до своєї технічної документації SRS. Ми вважаємо, що краще зараз перерахувати проблеми, які ви хочете вирішити за допомогою своїх функцій. Це стане в нагоді пізніше під час мозкового штурму та функцій моніторингу. Ви можете повернутися до цього розділу в будь-який час під час процесу розробки продукту та перевірити, чи не збилася команда з розробки користувацького досвіду з наміченого шляху.

Потреби стосуються питань, які користувачі зможуть вирішити за допомогою системи. Ви можете розділити ці потреби на підкатегорії, якщо ви маєте справу з дуже сегментованою аудиторією. Намагайтеся не вдаватися в подробиці потреб кожного користувача. Ви повинні залишити певний простір для тлумачення на випадок, якщо проблема виявиться більш суттєвою, ніж ви спочатку думали.

2.2 Припущення та залежності

Припущення — це припущення команди щодо продукту та його можливостей, які будуть правильними в 99% ситуацій. Природно припустити, наприклад, що платформа, яка допомагає водіям орієнтуватися вночі, буде використовуватися переважно в нічному режимі.

Яке значення мають припущення? Вони дозволяють вам спершу зосередитися на найважливіших функціях програми. Це припущення допомагає зрозуміти, що дизайнери повинні розробити інтерфейс, придатний для бачення в темряві для асистента водіння вночі. Деякі користувачі, звичайно, можуть відкрити програму протягом дня, але це довго, тому вам не потрібно відразу включати пов'язані елементи в прототип.

3. Системні функції та вимоги

У цій частині детально описано характеристики продукту та критерії виконання. Оскільки попередні два розділи стосуються продукту в цілому, тут ви знайдете більш повний опис.

3.1 Функціональні вимоги

Функціональні вимоги викладені в списку функцій, які будуть виконуватися в системі. Ці критерії стосуються того, «що буде створено?» а не «як» і «коли».

Функціональні вимоги починаються з опису функціональних можливостей, необхідних на основі того, наскільки вони важливі для програми. Якщо ви хочете спочатку попрацювати над цим, ви можете почати з дизайну, але потім вам слід перейти до розробки. Функціональні вимоги не вдаються в деталі технологічних стеків, оскільки вони можуть змінюватися в ході проекту. Замість того, щоб зосереджуватися на внутрішній логіці, функціональні вимоги зосереджуються на функціях кінцевого користувача.

3.2 Вимоги до зовнішнього інтерфейсу

Функціональні вимоги є значною частиною специфікації системних вимог. Щоб охопити всі необхідні функції системи, вам знадобиться 4-5 сторінок інформації. Деякі команди розбивають їх за темами, щоб полегшити читання документа.

Зазвичай компоненти дизайну SRS називають окремими від серверної частини та бізнес-логіки. Це має сенс, оскільки дизайнери, а не розробники займаються більшістю цієї сфери, а також тому, що саме тут починається процес розробки продукту.

Залежно від проекту вимоги до зовнішнього інтерфейсу можуть складатися з чотирьох типів:

Користувацький інтерфейс

Програмний інтерфейс

Апаратний інтерфейс

Інтерфейс зв'язку

Вимоги до зовнішнього інтерфейсу описують елементи сторінки, які будуть видимі кінцевому клієнту. Вони можуть включати список сторінок, елементи дизайну, ключові стилістичні теми, навіть художні елементи тощо, якщо вони важливі для продукту.

3.3 Системні вимоги

Системні вимоги продукту визначають умови, за яких його можна використовувати. Зазвичай вони стосуються специфікацій і функцій обладнання. Вимоги до обладнання SRS часто визначаються мінімальними та максимальними діапазонами, а також оптимальним порогом продуктивності продукту.

Створення системних вимог перед початком створення продукту може здатися важким, але це важливо. Розробники повинні дотримуватися вимог до апаратного забезпечення, щоб пізніше їм не довелося перезапускати проект. Мобільні програми (з багатьма змінними, які слід враховувати) і програми, які

потребують високої реактивності (ігри, будь-які продукти з VR/AR або IoT), особливо вразливі.

3.4 Нефункціональні вимоги

Для багатьох організацій ця частина ЄСВ є найскладнішою. Якщо функціональні вимоги стосуються питання, що створити, то нефункціональні стандарти визначають, як. Вони встановлюють критерії відповідно до того, наскільки ефективно повинна працювати система. Порогові значення продуктивності, безпеки та зручності використання включені в цю область.

Справжня цінність – це те, що ускладнює визначення нефункціональних вимог. Визначити такі фрази як «паралелізм» або «переносність» важко, оскільки вони можуть мати різні тлумачення для всіх залучених сторін. Тому ми виступаємо за оцінку кожної нефункціональної вимоги. Ви можете будь-коли переглянути вимоги свого проекту, щоб перевірити, чи відповідає поточна система початковим очікуванням.

Тема 5 Системні вимоги

(2 години)

Мета: Ознайомитись детально з поняттям системних вимог, розглянути способи запису системних вимог.

План:

1. Системні вимоги
2. Способи запису системних вимог
   a. Структурована мова специфікацій
   b. Створення специфікацій за допомогою PDL
   c. Специфікація інтерфейсів

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю

1. Що таке системні вимоги?
2. Перелічіть способи запису системних вимог.
3. Наведіть приклади запису системних вимог різними способами.

Системні вимоги – це більш деталізований опис користувацьких вимог. Вони зазвичай є основою для укладення контракту на розробку програмної системи і тому повинні представляти максимально повну специфікацію системи в цілому. Системні вимоги також використовуються в якості відправної точки на етапі проектування системи.

Специфікація системних вимог може будуватися на основі різних системних моделей, таких, як об'єктна модель або модель потоків даних. Різні моделі, використовувані при розробці специфікації системних вимог.

У принципі системні вимоги визначають, що повинна робити система, не показуючи при цьому механізму її реалізації. Але, з іншого боку, для повного опису системи потрібна деталізована інформація про неї, яка по можливості повинна включати всю інформацію про системну архітектуру. На те існує ряд причин.

Первісна архітектура системи допомагає структурувати специфікацію вимог. Системні вимоги повинні описувати підсистеми, з яких складається розроблювальна система.

У більшості випадків розроблювальна система повинна взаємодіяти із уже існуючими системами. Це накладає певні обмеження на архітектуру нової системи.

У якості зовнішньої системної вимоги може виступати умова використання для розроблювальної системи спеціальної архітектури.

Специфікації системних вимог часто пишуться природньою мовою. Використання природньої мови може породити певні проблеми при написанні деталізованої специфікації. Застосування природньої мови має на увазі, що ті, хто пише специфікацію, і ті, хто її читає, ті самі слова і вираження розуміють однаково. Однак насправді це не так, оскільки природній мові властива певна розмитість понять. Внаслідок цього та сама вимога може трактуватися різними людьми по-різному.

Щоб уникнути подібних проблем, розроблені методи опису вимог, які структурують специфікацію і зменшують розмитість визначень. Ці методи представлені в табл. 1. Крім цього, розроблені інші підходи, наприклад спеціальні мови опису вимог, які використовуються відносно рідко.

Таблиця 1. Способи запису специфікацій вимог

Система запису	Опис
Структурована природня мова	Використання стандартних форм і шаблонів для написання специфікації
Мови опису програм	Використання спеціальних структурованих мов, подібних мовам програмування, де специфікація вимог будується на основі обраної операційної моделі системи
Графічні нотації	Графічна мова, що використовує для опису функціональних вимог діаграми і блок-схеми, доповнені текстовими поясненнями. Найбільш відомий приклад такої графічної мови – діаграми структурного аналізу і проектування ПЗ (SADT).
Математичні специфікації	Це системи нотацій, засновані на математичних концепціях, таких, як теорія кінцевих автоматів

	або теорія множин. Це формалізований однозначно і позбавлений двозначності запис системних вимог. Однак багато замовників ПЗ не розуміють формальних специфікацій, внаслідок чого виникають певні проблеми при укладення контрактів на розробку програмних продуктів.

Структурована мова специфікацій

Це скорочена форма природної мови, призначена для написання специфікації вимог. Перевагою такого підходу до написання специфікацій є те, що він зберігає виразність і зрозумілість природної мови і разом з тим формалізує опис вимог. Структурованість мови проявляється у використанні спеціальної термінології, а також шаблонів для опису системних вимог. Структурована мова може включати мовні конструкції, узяті з мов програмування.

Для написання специфікації вимог для програмної системи керування польотами розроблені спеціальні форми, що допомагають описати вхідні і вихідні дані і функції системи.

Для опису системних вимог часто розробляються спеціальні форми і шаблони. Вони повинні враховувати, на основі чого будується специфікація: на основі об'єктів, керованих системою, на основі функцій, виконуваних системою, або на основі подій, оброблюваних системою. Приклад форми для специфікації показано в прикладі 1.

Приклад 1. Специфікація системної вимоги, що використовує стандартну форму

Екліпс/АРМ/Засоби/DE/RD/3.5.1

Функція. Додавання структурних елементів у схему.

Опис. Додавання структурних елементів в існуючу схему системної архітектури. Користувач вибирає тип структурного елемента і його місце розташування. Після вставки в схему структурний елемент стає виділеним (поточним структурним елементом). Користувач визначає місце розташування елемента шляхом переміщення курсору по області схеми.

Вхідні дані. Тип елемента, позиція елемента, ідентифікатор схеми.

Джерела вхідних даних. Тип елемента і позиція елемента задаються користувачем, ідентифікатор схеми отриманий з бази даних проекту.

Вихідні дані. Ідентифікатор схеми.

Пункт призначення. База даних проекту. Ідентифікатор схеми міститься в базу даних проекту по завершенню виконання даної функції.

Для виконання функції потрібна схема, визначена вхідним ідентифікатором схеми. Передумова. Схема відкрита і відображається на екрані користувача.

Постумови. Схема, за винятком вставки нового структурного елемента, не змінюється.

Побічні ефекти. Немає.

Специфікація: Єкліпс/АРМ/Засоби/DE/RD/3.5.1

Стандартні форми, використовувані для специфікування функціональних вимог, повинні містити наступну інформацію.

Опис функції або об'єкта.

Опис вхідних даних і їх джерела.

Опис вихідних даних із вказівкою пункту їх призначення.

Вказівка, що необхідна для виконання функції.

Якщо це специфікація функції, необхідний опис попередніх умов (передумов), які повинні виконуватися перед викликом функції, і опис заключної умови (постумов), яка має бути виконане після завершення виконання функції.

Опис побічних ефектів (якщо вони є).

Використання структурованої мови знімає деякі проблеми, властиві специфікаціям, написаною природньою мовою, оскільки знижує "варіабельність" специфікації і більш ефективно її структурує. Разом з тим деяка "розмитість" визначень і описів у специфікації залишається. Альтернативою використанню структурованої природньої мови може служити спеціальна мова опису специфікацій, яка повністю знімає проблему нечіткості опису вимог. Але з іншого боку, неспеціаліст знайде таку специфікацію важкою для читання і розуміння.

Створення специфікацій за допомогою PDL

Для зменшення властивої природній мові нечіткості понять при описі системних вимог використовується спеціальна мова опису програм (program description language – PDL). Ця мова подібна таким мовам програмування, як Java і Ada, але більш абстрактна. Перевагою застосування PDL для створення специфікації є те, що в такій специфікації можна перевірити синтаксис і семантику існуючими програмними засобами. Ці перевірки дозволяють вилучити помилки і непогодженість в описі вимог.

Застосування PDL приводить до дуже докладних і деталізованих специфікацій, які іноді просто неможливо ввести в заключний документ із описом системних вимог. Рекомендовано використовувати PDL у наступних ситуаціях.

Якщо описувана операція складається з послідовності простих дій і важливий порядок їх виконання. Описати такі послідовності дій природньою мовою часом важко, оскільки їх виконання може супроводжуватися вкладеними умовами або вони можуть повторюватися циклічно. Цієї ситуації відповідає специфікація системи обслуговування банкоматів, наведена в лістингу 1.

Якщо необхідно специфікувати апаратні або програмні інтерфейси, тому що практично у всіх специфікаціях системних вимог доводиться описувати інтерфейси.

Лістинг 1. Специфікація системи керування банкоматами

```
class ATM {
```

```
// декларативна частина
public static void main (String args[]) throws Invalidcard{
try {
thiscard.read ();
//може породити виняткову ситуацію Invalidcard
pin = Keypad.readpin ();
attempts = 1;
while (!thiscard.pin.eguals (pin) & attempts < 4)
{ pin = Keypad.readpin();
Attempts = attempts + 1;
}
if ( !thiscard.pin.eguals (pin))
throws new Invalidcard ("Неправильний PIN-код") ;
thisbalance = thiscard.getbalance();
do ( Screen.promt ("Будь ласка, виберіть сервіс");
service = Screen.touchkey ();
switch (service) {
CASE Services.withdrawalwithreceipt:
receiptrequired = true;
CASE Services.withdrawalnoreceipt:
amount = Keypad.readamount ();
if (amount > thisbalance)
{ Screen.printmsg ("Рахунок перевищений");
break;
}
Dispenser.deliver (amount);
newbalance = thisbalance - amount;
if (receiptrequired)
Receipt.print (amount, newbalance);
break;
//далі опис інших сервісів
default: break ;
}
}
while (service != Services.quit);
thiscard.returntouser ("Будь ласка, заберіть картку");
}
catch (Invalidcard e )
{ Screen.printmsg ("Недійсна картка або PIN-код") ;
}
//далі обробка інших виключень
}//main ()
```

}//ATM

Разом з тим підхід до побудови специфікацій, заснований на PDL, має свої недоліки.

1. PDL, використовуваний для написання специфікації, може не мати достатніх засобів для опису всіх системних функцій.
2. Специфікації, створені за допомогою PDL, зрозумілі тільки людям, що мають певні знання мов програмування.
3. Системна архітектура, отримана на основі такої специфікації, не є системною моделлю, яка допомогла б користувачеві розібратися в структурі системи.

Найбільш ефективним способом розробки специфікацій є комбінація підходу, заснованого на PDL, з використанням структурованої природньої мови. У цьому випадку формалізовані записи природньою мовою використовуються для опису системи в цілому, а PDL – для опису послідовностей керуючих дій або для деталізованого опису інтерфейсів.

Специфікація інтерфейсів

Переважна більшість розроблювальних програмних систем повинні взаємодіяти з іншими системами, що вже існують. Для того щоб нова система могла працювати разом з іншими системами, необхідно специфікувати інтерфейси цих систем. Такі специфікації створюються на ранньому етапі розробки систем і включаються (зазвичай у вигляді додатка) у специфікацію системних вимог.

Розрізняють три типи специфіцікуємих інтерфейсів.

1. Процедурні інтерфейси, коли існуючі підсистеми пропонують набір сервісів, доступних за допомогою інтерфейсної процедури, що викликається.
2. Структури (інтерфейсні формати) даних, які пересилаються від однієї підсистеми до іншої. У цьому випадку може використовуватися PDL, заснований мовою програмування Java, який дозволяє описати класи даних з відповідними полями, що формують структуру даних.
3. Спеціальні представлення даних, наприклад у вигляді впорядкованої послідовності двійкових розрядів. Мова Java не підтримує такі детальні описи даних не рекомендується в такому разі використовувати PDL, заснований мовою Java.

Формальні нотації дозволяють описати інтерфейси чітко і недвозначно. Однак такі специфікації зрозумілі тільки фахівцям, що володіють певними знаннями. Формальні нотації рідко використовуються на практиці, хоча, вони ідеально підходять для створення специфікацій інтерфейсів. Менш формальні специфікації інтерфейсів, отримані за допомогою PDL, є компромісом між зрозумілістю і точністю опису інтерфейсів.

У лістингу 2 представлений приклад опису інтерфейсу першого типу (з перерахованих вище). Цей опис процедурного інтерфейсу сервера друку. Він

управляє чергами на друк, що сдійснюється декількома принтерами. Користувачі можуть перевіряти чергу, відповідну до будь-якого принтера, і видаляти свої файли із черги. Вони також можуть перемикати свої файли з одного принтера на інший.

Лістинг 2. Опис інтерфейсу сервера друку за допомогою PDL

```
interface Printserver {
// визначення абстрактного сервера друку
// потрібно: інтерфейс Printer, інтерфейс Printdoc
// надає функції: initialize (ініціалізація),
// print (друк),
// displayprintqueue (відображення черги на друк),
// cancelprintjob (видалення файлу із черги),
// switchprinter (перемикання між принтерами)
void initialize (Printer p);
void switchprinter (Printer printrinter p2, Printdoc d);
}//Printserver
```

Специфікація, наведена в лістингу 2 – абстрактна модель сервера друку без деталізації інтерфейсних частковостей. Інтерфейсні функції можна описати за допомогою структурованої природної мови, за допомогою PDL, заснованого мовою програмування Java (лістинг 1), або за допомогою формальних нотацій.

Тема 6 Документування системних вимог

(2 години)

Мета: Ознайомитись детально з поняттям системних вимог, розглянути способи запису системних вимог.

План:

1. Документування системних вимог
2. Стандарт IEEE/ANSI 830-1993
3. Завдання

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю

1. Що таке системні вимоги?
2. Хто читає системні вимоги?
3. Яким умовам повинна відповідати специфікація програми?

Документ, що містить вимоги, також називаний специфікацією системних вимог – це офіційне приписання для розробників програмної системи. Він містить користувацькі вимоги і деталізований опис системних вимог. У деяких випадках користувацькі і системні вимоги можуть не різнитися, виступаючи спільно у

вигляді однорідного опису системи. В інших випадках користувацькі вимоги приводяться у введенні документа-специфікації. Якщо загальна кількість вимог велика, деталізовані системні вимоги можуть бути представлені у вигляді окремого документа.

Системну специфікацію читає безліч різних людей, починаючи від вищого керівництва компанії – замовника системи і закінчуючи рядовим розробником системи.

Хенінгер (Heninger) сформулював шість **умов**, яким повинна відповідати специфікація програмної системи.

1. Описувати тільки зовнішню поведінка системи.
2. Указувати обмеження, що накладаються на процес реалізації системи.
3. Передбачати можливість внесення змін у специфікацію.
4. Служити довідковим засобом у процесі супроводу системи.
5. Відображати весь життєвий цикл системи.
6. Передбачати реакцію системи і групи супроводу на непередбачені (позаштатні) ситуації.

Хоча із часу написання цих умов пройшло більш 20 років, вони не втратили своєї актуальності і є гарною підмогою при розробці специфікацій. Разом з тим часом складно або навіть неможливо описати систему тільки в термінах зовнішньої поведінки, тобто тільки за допомогою функцій, які вона повинна виконувати, оскільки в специфікації також повинні бути відбиті обмеження, що накладаються оточенням уже існуючих систем. Інша умова – охват усього життєвого циклу системи – приймається всіма розробниками, але рідко виконується на практиці.

Багато організацій, такі, як Міністерство оборони США і Інститут інженерів по електротехніці і радіоелектроніці IEEE, розробили власні стандарти документування специфікацій. Найбільш відомий стандарт розроблений IEEE і називається IEEE/ANSI 830-1993 [IEEE, 1993]. Даний стандарт передбачає наступну структуру специфікації.

1. Введення
   - Мети документа
   - Призначення програмного продукту
   - Визначення, акроніми і абревіатури
   - Список літератури і інших джерел
   - Огляд специфікації
2. Загальний опис
   - Опис програмного продукту
   - Функції програмного продукту
   - Користувацькі характеристики
   - Загальні обмеження
   - Обґрунтування, припущення і допущення

3. Специфікація вимог
4. Додатки
5. Покажчики

Специфікація вимог охоплює функціональні, нефункціональні і інтерфейсні вимоги. Це найбільш значима частина документа, але внаслідок украй широкого діапазону можливих вимог, пропонованих програмним системам, у стандарті не визначена структура цього розділу. Тут можуть бути документовані зовнішні інтерфейси, описані функціональні можливості системи, наведені вимоги, що визначають логічну структуру баз даних, обмеження, що накладаються на структуру системи, описані інтеграційні властивості системи або її якісні характеристики.

Хоча стандарт IEEE не ідеальний, він може служити відправною точкою при написанні специфікації. Зазвичай, при її написанні необхідно також ураховувати стандарти, прийняті в організації – розробники ПЗ. У табл. 1 описані можливі розділи специфікації, побудованої на підставі стандарту IEEE.

Розділ	Опис
Передмова	Тут визначається коло осіб, на яких розрахований даний документ. Описуються попередні версії розроблювального програмного продукту, а також зміни, внесені в кожну версію. Дається обґрунтування для створення нової версії продукту
Введення	Тут більш розгорнуто обґрунтовується необхідність створення системи. Коротко перелічуються системні функції і пояснюється, як система буде працювати разом з іншими системами. Повинно бути показано, як розробка системи “вписується” у загальну бізнес-стратегію компанії, що замовляє програмний продукт
Глосарій	Дається опис технічних термінів, використовуваних у документі. Тут не робиться яких-небудь припущень про рівень знань або практичному досвіді читача документа
Користувацькі вимоги	Описуються сервіси, надавані користувачам, і нефункціональні системні вимоги. Цей опис може бути зроблений природньою мовою з використанням діаграм, блок-схем і інших форм запису, зрозумілих замовникові програмної системи. Тут також повинні бути наведені стандарти на програмний продукт і процес його розробки
Системна	Тут приводиться високорівневе представлення

архітектура	можливої системної архітектури із вказівкою, як розподілені системні функції по компонентах системи. Обов'язково повинні бути виділені повторно використовувані (тобто вже існуючі) компоненти
Системні вимоги	Докладно описуються функціональні і нефункціональні вимоги. Якщо необхідно, нефункціональні вимоги доповнюються описом інтерфейсів інших систем
Системні моделі	Тут представлено кілька системних моделей, що показують взаємовідношення між системними компонентами і між системою і її оточенням. Це можуть бути об'єктні моделі, моделі потоків даних або моделі даних
Еволюція системи	Приводяться основні припущення і допущення, на яких базується система, а також очікувані (прогнозовані) зміни в апаратних засобах, у потребах користувачів і т.п.
Додатки	Тут приводиться спеціалізована інформація, що відноситься до розроблювальної системи, наприклад опис апаратних засобів або бази даних, з якими повинна працювати система. При описі апаратних засобів необхідно показати мінімальну і оптимальну конфігурації, при яких може працювати програмна система. Опис бази даних повинен відображати логічну структуру даних, з якими буде працювати система, і відносини між ними
Покажчики	У документі можливе використання різних покажчиків. Це може бути звичайний алфавітний покажчик, покажчик діаграм або покажчик системних функцій

Зазвичай, інформація, що включається в специфікацію, залежить від типу розроблювального програмного забезпечення і від обраної технології розробки. Наприклад, якщо при розробці буде використовуватися еволюційний підхід, багато розділів специфікації, перераховані в табл. 1, будуть зайві. Якщо ж розроблювальне ПЗ є тільки частиною великої системи, що полягає із взаємодіючих апаратних і програмних систем, тоді по необхідності вимоги будуть дуже докладними і деталізованими. У цьому випадку специфікація може мати значний обсяг і буде містити більше розділів, чим перераховане в табл. 1. Для більших документів необхідно робити зміст і докладні покажчики для пошуку необхідної інформації.

Завдання:

1. Автоматизована система продажу залізничних квитків. Користувач указує пункт призначення, вставляє кредитну картку і вводить особистий ідентифікаційний номер. Система видає проїзний квиток і знімає із кредитної картки суму, рівну вартості проїзду в зазначений пункт. Коли користувач натискає початкову кнопку, відображається меню можливих пунктів призначення із вказівкою вартості проїзду до кожного пункту. Після вибору пункту призначення користувачеві пропонується вставити кредитну картку. Потім перевіряється кредитна картка, після чого користувачеві пропонується ввести особистий ідентифікаційний номер. По закінченню операцій із кредитною карткою видається проїзний квиток.
2. Перепишіть вимогу попереднього пункту, використовуючи структурний підхід.
3. Запишіть системні вимоги для системи продажу квитків за допомогою мови, заснованого мовою програмування Java. Зверніть увагу на обробку помилок користувача.

Тема 7 Стандартизація розробки програмного забезпечення

(4 години)

Мета: Ознайомитись з поняттям стандартизації розробки ПЗ, розглянути основні стандарти.

План:

1. Стандарт BSP
2. Міжнародні стандарти ISO

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю

1. Що таке галузеві стандарти?
2. Які кроки передбачав процес BSP?
3. На які групи розбиваються процеси ЖЦ ПЗ за стандартом ISO 12207?

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

Для полегшення взаємодії підприємств і зручності споживачів розробляються галузеві стандарти. Розвиток кожного виду господарської

діяльності призводить до потреби в державних засобах забезпечення якості продукції або процесу, тому розробляються і затверджуються державні стандарти.

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

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

Процес BSP передбачав виділення в ході розробки ІС таких кроків:

− отримання підтримки керівництва;

− визначення процесів підприємства;

− визначення класів даних;

− проведення інтерв'ю;

− обробка та організація результатів інтерв'ю.

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

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

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

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

Міжнародні стандарти ISO

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

Рисунок 1 – Процеси ЖЦ ІС відповідно до стандарту ISO 12207

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

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

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

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

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

- Керівництво для ISO / IEC 12207 (ISO / IEC TR 24748-3: 2011 Systems and software engineering - Life cycle management - Part 3: Guide to the application of ISO / IEC 12207 (Software life cycle processes))

- Керівництво по використанню ISO / IEC 12207 в управлінні проектами (ISO / IEC TR 16326: 2009 Systems and software engineering - Life cycle processes - Project management).

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

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

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

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

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

Таблиця 2. - Стадії створення систем (ISO / IEC 15288)

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

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

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

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

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

ISO 9000-3. Управління якістю і гарантії якості. Частина 3.

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

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

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

Тема 8 Стандарти організації IEEE та CMM

(6 години)

Мета: Розглянути склад Керівництва до зведення знань з програмної інженерії SWEBOK, методологію CMM та ознаки зрілості організації.

План:

1. Стандарт IEEE
2. Методологія CMM
3. Ознаки незрілої організації

4. Ознаки зрілої організації

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю

1. Що визначає CMM?
2. Назвіть ознаки злості та незрілості організації.
3. Назвіть переваги та недоліки моделі CMM.
4. Які галузі входять у SWEBOK?

Стандарти організації IEEE

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

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

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

Software requirements – програмні вимоги;

Software design – дизайн (архітектура);

Software construction – конструювання програмного забезпечення;

Software testing – тестування;

Software maintenance – експлуатація (підтримка) програмного забезпечення;

Software configuration management – управління конфігураціями;

Software engineering management – управління з програмної інженерії;

Software engineering process – процеси програмної інженерії;

Software engineering tools and methods – інструменти та методи;

Software quality–- якість програмного забезпечення.

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

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

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

Говорячи про стандартизацію процесів підприємства потрібно розглянути модель зрілості технологічних процесів організації Capability Maturity Model

(CMM), яка розроблена Інститутом інженерів програмного забезпечення (Software Engineering Institute, SEI) і корпорацією Mitre під керівництвом Воттса Хамфрі (Watts Humphrey).

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

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

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

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

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

Незрілою вважається організація, в якій:

• відсутнє довгострокове і проектне планування;

• процес розробки програмного забезпечення та його ключові моменти не ідентифіковані, реалізація процесу залежить від поточних умов, конкретних менеджерів і виконавців;

• методи і процедури не стандартизовані і не задокументовані;

• результат не визначений реальними критеріями, які встановлюються запланованими показниками із застосуванням стандартних технологій і розроблених метрик;

• процес вироблення рішення відбувається стихійно, на грані мистецтва.

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

Основні ознаки зрілої організації:

• в компанії чітко визначені і документовані процедури управління вимогами, планування проектної діяльності, управління конфігурацією, створення і тестування програмних продуктів, відпрацьовані механізми управління проектами;

• ці процедури постійно уточнюються і удосконалюються;

• оцінки часу, складності та вартості робіт ґрунтуються на накопиченому досвіді, розроблених метриках і кількісних показниках, що робить їх досить точними;

• актуалізовані зовнішні і створені внутрішні стандарти на ключові процеси та процедури;

• існують обов'язкові для всіх правила оформлення методологічної і програмної документації та документації користувача;

• технології незначно змінюються від проекту до проекту, на основі стабільних і перевірених підходів та методик максимально використовуються напрацьовані в попередніх проектах організаційний і виробничий досвід, програмні модулі, бібліотеки програмних засобів;

• активно апробуються і впроваджуються нові технології, проводиться оцінка їх ефективності.

CMM визначає п'ять рівнів технологічної зрілості компанії (рис. 1), за якими замовники можуть оцінювати потенційних претендентів на підпис контракту, а розробники - удосконалювати процеси створення ПЗ.

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

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

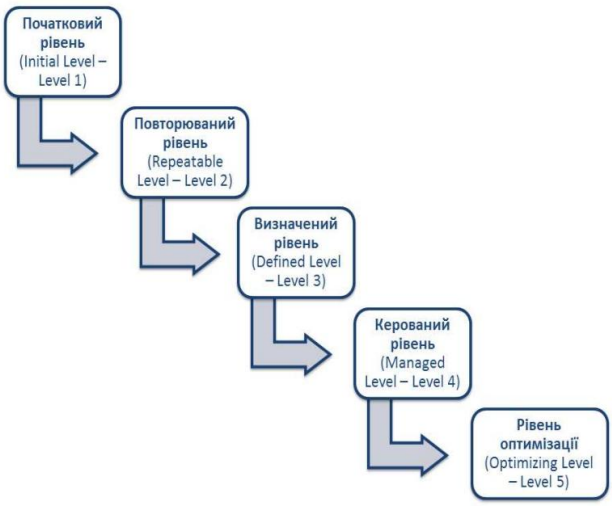


Рисунок 1 – Рівні зрілості компанії за стандартом CMM

Початковий рівень (Initial Level - Level 1)

На даному рівні компанія може отримати замовлення, розробити і передати замовнику програмний продукт. Стабільність розробок відсутня. Лише деякі процеси визначені, результат повністю залежить від зусиль окремих

співробітників. Успіх одного проекту не гарантує успішності наступного. До цієї категорії можна віднести будь-яку компанію, яка хоч якось виконує взяті на себе зобов'язання. Ключові області процесу цього рівня не зафіксовано.

Повторюваний рівень (Repeatable Level - Level 2)

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

У компанії розроблені деякі внутрішні стандарти і організовані спеціальні групи перевірки якості QA . Зміни версій кінцевого програмного продукту і створених проміжних програмних засобів відслідковуються в системі управління конфігурацією. Є необхідна дисципліна дотримання встановлених правил. Ефективні методики та процеси впроваджуються, що забезпечує можливість повторення успіху попередніх проектів в тій же прикладній галузі. Ключові галузі процесу розробки ПЗ цього рівня наведені на рисунку 2.

Визначний рівень (Defined Level - Level 3)

Рівень характеризується деталізованим методологічним підходом до управління (описані і закріплені в документованій політиці типові дії, що необхідні для багаторазового повторення: ролі і відповідальність учасників, стандартні процедури та операції, порядок дій, кількісні показники і метрики процесів, формати документів та ін.). Для створення і підтримки методологій в актуальному стані в організації підготовлена і постійно функціонує спеціальна група (рис. 2)

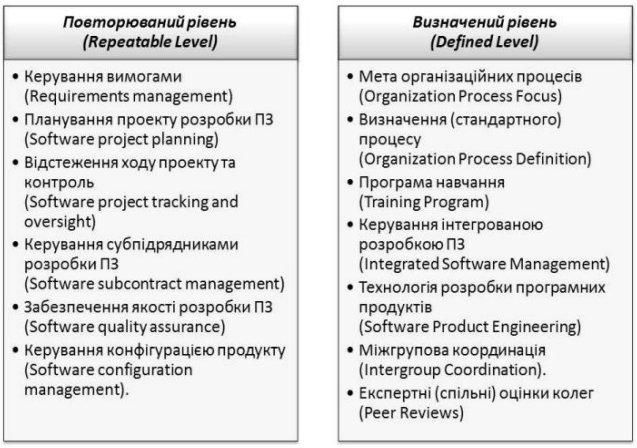


Рисунок 2 - Ключові області повторюваного і визначного рівнів зрілості компанії

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

стандартизовані і інтегровані в уніфіковану для всієї організації технологію створення ПЗ. Кожен проект використовує затверджену версію цієї технології, адаптовану до особливостей поточного проекту. Ключові галузі процесу розробки ПЗ цього рівня наведені на рисунку 2.

Керований рівень (Managed Level – Level 4)

Рівень, на якому розроблені і закріплені у відповідних нормативних документах кількісні показники якостей. Більш високий рівень управління проектами досягається за рахунок зменшення відхилень різних показників проекту від запланованих. При цьому тенденції зміни продуктивності процесу можна відокремити від випадкових варіацій на підставі статистичної обробки результатів вимірювань в процесах. Ключові області процесу розробки ПЗ цього рівня наведені на рисунку 3.

Рівень оптимізації (Optimizing Level – Level 5)

Для цього рівня заходи щодо вдосконалення розраховані не тільки на існуючі процеси, а й на впровадження, використання нових технологій і оцінку їх ефективності. Основним завданням всієї організації на цьому рівні є постійне вдосконалення існуючих процесів, в ідеалі спрямоване на запобігання відомої помилки або дефекту і попередження можливих. Застосовується механізм повторного використання компонентів від проекту до проекту (шаблони звітів, формати вимог, процедури і стандартні операції, бібліотеки модулів програмних засобів). Ключові галузі процесу розробки ПЗ цього рівня наведені на рисунку 3.

Рисунок 3 - – Ключові галузі керованого рівня зрілості і рівня оптимізації компанії

СММ визначає такий мінімальний набір вимог: реалізувати 18 ключових галузей процесу розробки ПЗ, що містять 52 мети, 28 зобов'язань компанії, 70 можливостей виконання (гарантій компанії) і 150 ключових практик.

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

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

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

Тема 9 Стандарт SPICE

(2 години)

Мета: Розглянути стандарт SPICE.

План:

1. Стандарт SPICE

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю

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

Стандарт SPICE успадкував багато рис більш ранніх стандартів, у тому числі і вже згадуваних ISO 9001 і CMM. Найбільше SPICE нагадує CMM. Точно так само, як і в CMM, основним завданням організації є постійне поліпшення

процесу розробки можливостей (в SPICE визначено 6 різних рівнів), але ці рівні застосовуються не тільки до організації в цілому, а й до окремо взятих процесів.

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

Безумовно, вдосконалення процесів життєвого циклу програмного забезпечення абсолютно необхідне. Проте слід мати на увазі, що побудова «більш зрілого» процесу розробки не обов'язково забезпечує створення більш якісного програмного забезпечення. Це хоча і пов'язані, але абсолютно різні процеси.

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

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

Блок змістових модулів №3 Технології розробки програмного забезпечення

Змістовий модуль 3.1 *Нотації розробки ER- діаграм*

Тема 10 Графічні нотації побудови ERD

(4 години)

Мета: Розглянути основні графічні нотації побудови діаграми «сутність-зв’язок»: Баркера, Чена та Мартіна.

План:

1. Підхід Мартіна
2. Сутність і зв'язки в нотації Мартина
3. Моделювання даних Case-метод Баркера
4. Сутність, відношення і зв'язки в нотації Чена

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питання для самоконтролю

1. Що таке діаграма «сутність-зв’язок»?
2. Як позначаються сутності та зв’язки в нотації Маптіна?
3. В чому відмінності в цих трьох нотаціях, а в чому вони схожі?

Діаграми "сутність - зв'язок"

Діаграми "сутність - зв'язок" (ERD) призначені для розробки моделей даних та забезпечують стандартний засіб визначення даних і відношень між ними. Фактично за допомогою ERD здійснюється деталізація сховищ даних проектованої системи, а також документуються сутності системи та засоби їхньої взаємодії, включаючи ідентифікацію об'єктів, важливих для предметної області (сутності), властивостей цих об'єктів (атрибутів) і їхніх відношень з іншими об'єктами (зв'язків).

Основні етапи підходу Мартина

IE-методологія Мартина є загальною стратегією розробки інформаційних систем, яка зосереджує увагу на стратегічному плануванні і бізнес процесах. В той же час вона є і інженерним підходом до розробки ПЗ, тобто забезпечує покрокову процедуру побудови інформаційної системи "згори-донизу" (дозволяючи при цьому працювати з неієрархічними структурами даних). Основні етапи підходу Мартина наведені на рисунку 1.

1. Етап стратегічного інформаційного планування починається з побудови стратегічного плану для бізнес системи, що включає мету і стратегії її досягнення. Далі будується модель предметної області, що відображає існуючу специфіку і визначає основні бізнес-поцеси та організаційну структуру бізнес-системи, а також визначає порядок розробки інформаційної системи. При моделюванні використовуються діаграми декомпозиції (ієрархічні деревоподібні структурні діаграми) і діаграми “сутність-зв'язок” для представлення основних бізнес-процесів і структур даних відповідно.

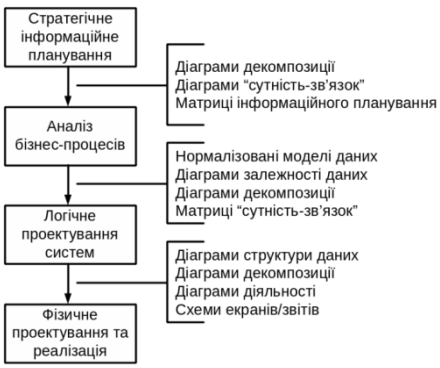


Рис. 1. Основні етапи методології Мартина.

2. На етапі аналізу основні бізнес-процеси, розроблені на етапі 1, використовуються для розбиття загальної задачі на часткові, при цьому основну увагу приділяється визначенню інформаційної діаграми "сутність-зв'язок", що трансформується в нормалізовану модель даних, а діаграми декомпозиції розподіляються за підзадачами. Для представлення процесів служать DFD, діаграми залежності даних (діалект DFD) і діаграми декомпозиції, а для співставлення даних і процесів, в яких ці дані використовуються, застосовуються матриці "сутність процесів".
3. На етапі логічного проектування IE стає аналогічною SE для розробки ПЗ. Базою для проектування є процеси, розроблені на етапі аналізу. Використовуючи методики функціональної декомпозиції "згори-донизу", проектуються специфікації обробки в процесах і їх логічні структури даних. При цьому використовується діаграми структур даних (діалект ERD), визначальні типи сутностей, їхні атрибути і зв'язки, діаграми декомпозиції і діаграми діяльності (у вигляді мініспецифікацій), що деталізують логіку процесів. Для погодження вимог користувача створюються прототипи інтерфейсів користувача з допомогою схем екранів/звітів.
4. На етапі фізичного проектування і реалізації реалізується перетворення логічної моделі ІС в фізичну та її реалізація.

Сутність і зв'язки в нотації Мартина

Сутність являє собою множину екземплярів реальних або абстрактних об'єктів (людей, подій, станів, ідей, предметів), що мають спільні атрибути або характеристики. Будь-який об'єкт системи може бути представлений лише однією сутністю, що повинна бути унікально ідентифікована. При цьому ім'я сутності повинно відображати тип або клас об'єкту, а не його конкретний екземпляр (наприклад, СПІВРОБІТНИК, а не конкретне прізвище або ім'я працівника).

Символи ERD, нотації Мартина, відповідні сутності наведені на рис. 2.

Рис. 2. Елементи діаграми в нотації Мартина.

Незалежна сутність являє незалежні дані, що завжди наявні в системі. При цьому зв'язки з іншими сутностями можуть як існувати, так і бути відсутні. В свою

чергу, залежна сутність являє дані, що залежить від інших сутностей. Тому вона повинна завжди мати зв'язки з іншими сутностями.

Список атрибутів приводиться всередині прямокутника, який позначає сутність. Ключові атрибути підкреслюються. Зв'язки позначаються лініями, що з'єднують сутності, вигляд лінії в місці з'єднання з сутністю позначає тип зв'язку. Тип зв'язку обирається з наступної множини:

{ "О або 1", "0 або більше", "1 до 1", "1 або більше", "багато до багатьох"}

Практика показала, що для більшості застосувань достатньо використати наступні типи зв'язків.

1. 1*1 (один-до-одного). Зв'язок даного типу використовується, як правило, на верхніх рівнях ієрархії моделі даних, а на нижніх рівнях зустрічається порівняно рідко.

2: 1*n (один-до-багатьох). Зв'язок даного типу використовується найбільш часто.

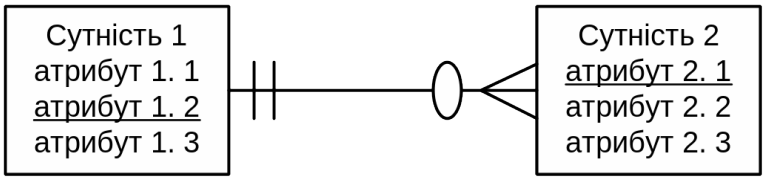
3. n*m (багато-до-багатьох). Зв'язок даного типу звичайно використовується на ранніх етапах проектування з метою прояснення ситуації. В подальшому кожен такий зв'язок повинен бути перетворений в комбінацію зв'язків типу 1 та 2 .

На рис.3 показані типи зв'язків та їх умовні позначення в нотації Мартина.

Позначення	Тип звязку
	Немає
	1, 1
	0, 1
	M, N
	0, N
	1, N

Рис. 3. Позначення зв’язків в нотації Мартина

Ім'я зв'язку вказується на ліній. Наприклад:



На рисунку 4 наведена діаграма "сутність-зв'язок", що демонструє зв'язки між об'єктами прикладу банкомат.

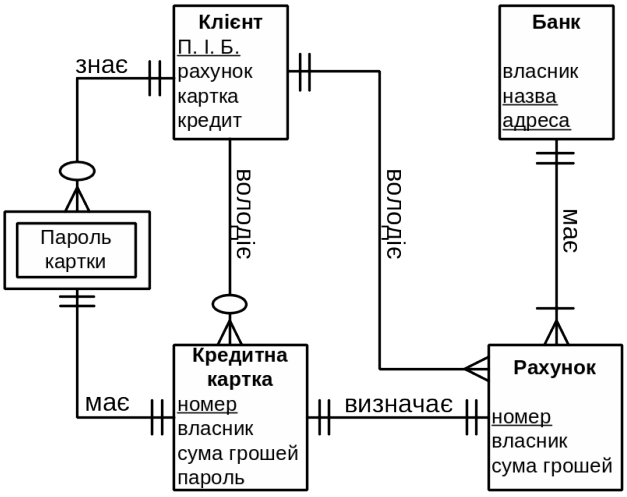


Рис. 4. Діаграма "сутність-зв'язок" для прикладу.

Згідно до цієї діаграми кожен БАНК містить такі атрибути: власник, назва, адреса і МАЄ один або більше РАХУНКІВ. Кожен клієнт має атрибути: П. І. Б, рахунок, картка, кредит і може ВОЛОДІТИ (водночас) однією або більше КРЕДИТНОЮ КАРТКОЮ і одним або більше БАНКІВСЬКИМ РАХУНКОМ, кожен з яких ВИЗНАЧАЄ рівно одну КРЕДИТНУ КАРТКУ (відзначимо, що у клієнта може і не бути ані рахунку, ані кредитної картки). В свою чергу РАХУНОК має наступні атрибути: власник, номер, сума грошей. КРЕДИТНА КАРТКА МАЄ лише один залежний від неї ПАРОЛЬ КАРТКИ, а кожен КЛІЄНТ ЗНАЄ (але може і забути) ПАРОЛЬ КАРТКИ, і містить такі атрибути: номер, власник, сума грошей, пароль.

Моделювання даних Case-метод Баркера

Мета моделювання даних полягає в забезпеченні розробника ІС концептуальною схемою бази даних у формі однієї моделі або декількох локальних моделей, які відносно легко можуть бути відображені в будь-яку систему баз даних.

Найбільш поширеним засобом моделювання даних є діаграми "сутність-зв’язок" (ERD) 3 їхньою допомогою визначаються важливі для предметної області об'єкти (сутності), їх властивості (атрибути) і відношення один з одним (зв'язки) ERD безпосередньо використовуються для проектування реляційних баз даних.

Нотація ERD була уперше введена П. Ченом (Chen) і отримала подальший розвиток у роботах Баркера [8]. Метод Баркера буде викладатися на прикладі моделювання діяльності компанії по торгівлі автомобілями. Нижче приведені витримки з інтерв'ю, проведеного з персоналом компанії.

Головний менеджер: один з основних обов'язків — утримання автомобільного майна. Він повинен знати, скільки заплачено за машини і які накладні витрати. Володіючи цією інформацією, він може встановити нижню ціну, за якою міг би продати даний примірник. Крім того, він відповідає за продавців і йому потрібно знати, хто що продає і скільки машин продав кожний із них.

Продавець: йому потрібно знати, яку ціну встановлювати і яка нижня ціна, за якою можна здійснити операцію Крім того, йому потрібна основна інформація про машини: рік випуску марка, модель і т.п.

Адміністратор: його задача зводиться до впорядкування контрактів, для чого потрібна інформація про покупця, автомашину і продавця, оскільки саме контракти приносять продавцям винагороди за продукцію.

Перший крок моделювання - витяг інформації з інтерв'ю і виділення сутностей.

Сутність (Entity) ─ реальний або уявний об'єкт, що має суттєве значення для аналізованої предметної області, інформація про який підлягає збереженню (рис. 5).

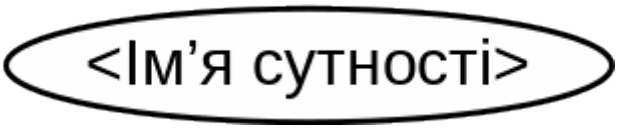


Рис. 1. Графічне зображення сутності.

Кожна сутність повинна мати унікальний ідентифікатор. Кожний екземпляр сутності повинний однозначно ідентифікуватися і відрізнятися від всіх інших екземплярів даного типу сутності. Кожна сутність повинна мати деякі властивості:

• кожна сутність повинна мати унікальне ім'я, і до того ж самого імені повинна завжди застосовуватися та сама інтерпретація Та сама інтерпретація не може застосовуватися до різноманітних імен, якщо тільки вони не є псевдонімами;

• сутність володіє одним або декількома атрибутами, котрі або належать сутності, або унаслідуються через зв'язок;

• сутність володіє одним або декількома атрибутами, які однозначно ідентифікують кожний екземпляр сутності;

• кожна сутність може мати будь-яку кількість зв'язків з іншими сутностями моделі,

Звертаючись до приведених вище витримкам з інтерв'ю, очевидно, що сутності, які можуть бути ідентифіковані з головним менеджером - це автомашини і продавці. Продавцю потрібні автомашини і пов'язані з їх продажем дані. Для адміністратора важливі покупці, автомашини, продавці і контракти. Виходячи з цього, виділяються 4 сутності (автомашина, продавець, покупець, контракт), що зображаються на діаграмі в такий спосіб (рис. 6)

Рис. 6. Наступним кроком е моделювання.

Зв'язок (Relationship) це проіменована асоціація між двома сутностями, значима для аналізованої предметної області. Зв'язок — це асоціація між сутностями, при якому, як правило, колений екземпляр однієї сутності, що називається батьківською сутністю, асоційований із довільною (у тому числі нульовою) кількістю екземплярів другої сутності, яка називається сутністю-

нащадком, а кожний екземпляр сутності-нащадка асоційований у точності з одним екземпляром сутності-батька. Таким чином, екземпляр сутності-нащадка може існувати тільки при існуванні сутності батька.

Зв'язку може даватися ім'я, що виражається граматичним оборотом дієслова і яке розміщується біля лінії зв'язку. Ім'я кожного зв'язку між двома даними сутностями повинно бути унікальним, але імена зв'язків у моделі не зобов'язані бути унікальними. Ім'я зв'язку завжди формується з погляду батька, так що речення може бути утворене з'єднанням імені сутності-батька, імені зв'язку, вираження ступеня й імені сутності-нащадка.

Наприклад, зв'язок продавця з контрактом може бути виражений в такий спосіб

• продавець може одержати винагороду за 1 або більше контрактів;

• контракт повинний бути ініційований рівно одним продавцем.

Ступінь зв'язку й обов'язковість і графічно зображають у такий спосіб (рис. 7).

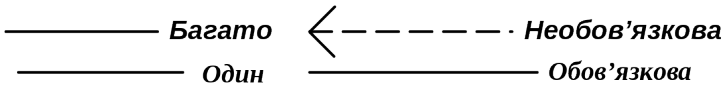


Рис. 7

Таким чином, 2 речення, що описують зв'язок продавця з контрактом, графічно будуть виражені в такий спосіб (рис. 8).

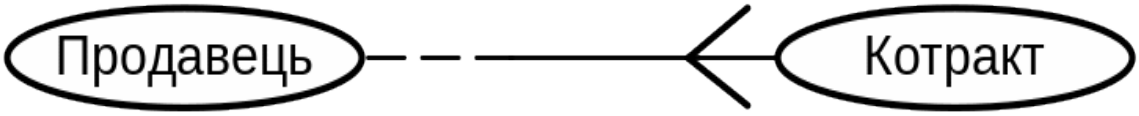


Рис. 8

Описавши також зв'язки інших сутностей, отримаємо наступну схему (рис. 9).

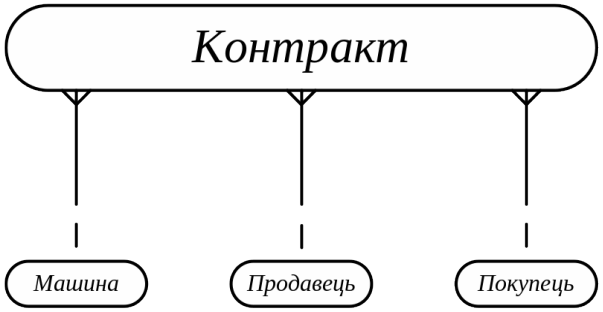


Рис. 9

Останнім кроком моделювання є ідентифікація атрибутів.

Атрибут - будь-яка характеристика сутності, значима для аналізованої предметної області і призначена для кваліфікації, ідентифікації, класифікації, кількісної характеристики або вираження стану сутності. Атрибут представляє тип характеристик або властивостей, асоційованих із множиною реальних або абстрактних об'єктів (людей, місць, подій, станів, ідей, пар предметів і т. д.). Екземпляр атрибута - це визначена характеристика окремого елемента множини. Екземпляр атрибута визначається типом характеристики і її значенням, що

називається значенням атрибута. У ER- моделі атрибути асоціюються з конкретними сутностями. Таким чином, екземпляр сутності повинен мати єдине визначене значення для асоційованого атрибута.

Атрибут може бути або обов'язковим, або необов'язковим (рис. 6). Обов'язковість означає, що атрибут не може приймати невизначених значень (null values). Атрибут може бути або описовим (тобто звичайним дескриптором сутності), або входити до складу унікального ідентифікатора (первинного ключа).

Унікальний ідентифікатор - це атрибут або сукупність атрибутів і/або зв'язків, призначений для унікальної ідентифікації кожного екземпляра даного тип) сутності. У випадку повної ідентифікації кожний екземпляр даного типу сутності цілком ідентифікується своїми власними ключовими атрибутами, у протилежному випадку в його ідентифікації беруть участь також атрибути іншої сутності-батька(рис. 10).

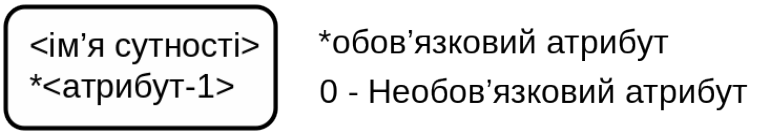


Рис. 10

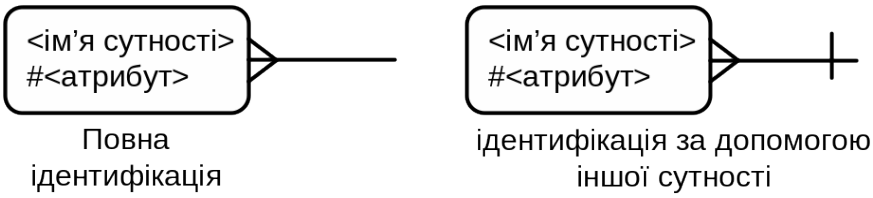


Рис. 11

Кожний атрибут ідентифікується унікальним ім'ям, яке виражається граматичним оборотом іменника, що описує характеристику, яка представляється атрибутом. Атрибути зображаються у вигляді списку імен всередині блока асоційованої сутності, причому кожний атрибут займає окремий рядок. Атрибути, що визначають первинний ключ, розміщаються нагорі списку і виділяються знаком "#".

Кожна сутність повинна володіти хоча б одним можливим ключем. Можливий ключ сутності - це один або декілька атрибутів, чиї значення однозначно визначають кожний екземпляр сутності. При існуванні декількох можливих ключів один із них позначається в якості первинного ключа, а інші - як альтернативні ключі.

З урахуванням наявної інформації доповнимо побудовану раніше діаграму (рис.8). Крім перерахованих основних конструкцій модель даних може містити ряд додаткових.

Підтипи і супертипи: одна сутність є узагальнюючим поняттям для групи подібних сутностей (рис. 13).

Зв'язки, що виключають один одного: кожний екземпляр сутності бере участь тільки в одному зв'язку з групи зв'язків, що взаємно виключають один одного (рис.14).

Кожній сутності присвоюється унікальне ім'я і номер, що розділяються косою рисою "/" і розміщуються над блоком.

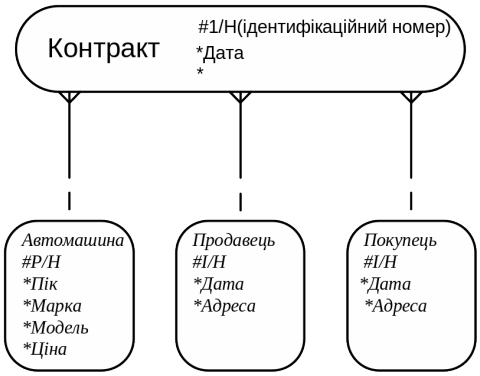


Рис. 12

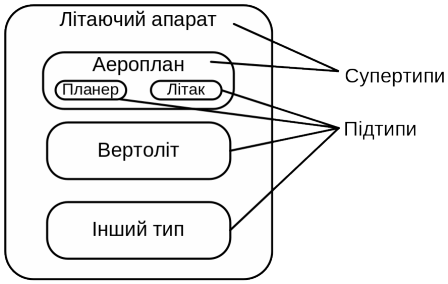


Рис. 13. Підтипи і супертипи.

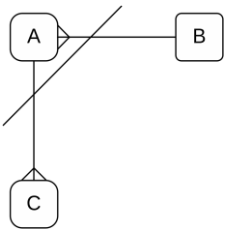


Рис. 14. Зв’язки, що виключаються.

Рекурсивний зв'язок: сутність може бути зв'язана сама із собою (рис. 15).

Непересувні зв'язки (non-transferrable): екземпляр сутності не може бути перенесений з одного екземпляра зв'язку в інший (рис. 16).

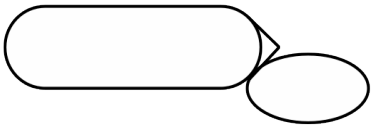


Рис. 15. Рекурсивний зв’язок.

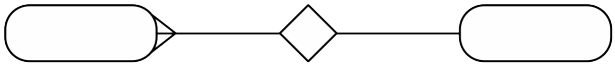


Рис. 16. Нерекурсивний зв’язок.

Сутність, відношення і зв'язки в нотації Чена

Сутність являє собою множину екземплярів реальних або абстрактних об'єктів (людей, подій, станів, ідей, предметів і т. ін.), що мають спільні атрибути або характеристики. Будь-який об'єкт системи може бути представлений лише однією сутністю, що повинна бути унікально ідентифікована. При цьому ім'я сутності повинно відображати тип або клас об'єкту, а не його конкретний екземпляр (наприклад, КНИГА, а не назва конкретної книги).

Відношення в самому загальному вигляді являє собою зв'язок між двома і більшою кількістю сутностей. Найменування відношення здійснюється за допомогою граматичного звороту дієслова (МАЄ, ВИЗНАЧАЄ, МОЖЕ ВОЛОДІТИ і т. ін.)

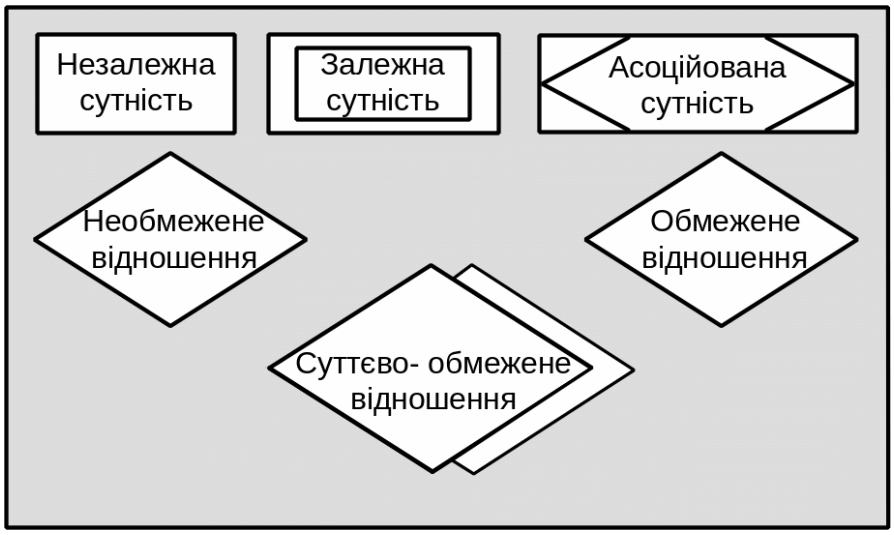


Рис.17. Позначення в ERD (нотація Чена).

Незалежна сутність являє незалежні дані, що завжди наявні в системі. При цьому відношення з іншими сутностями можуть як існувати, так і бути відсутні. В свою чергу, залежна сутність являє дані, що залежать від інших сутностей в системі. Тому вона повинна завжди мати відношення з іншими сутністю. Асоційована сутність представляє дані, що асоціюються з відношеннями між двома і більш сутностями.

Необмежене (обов'язкове) відношення являє собою безумовне відношення, тобто, відношення, що завжди існує до того часу, доки існують стосовні до справи сутності. Обмежене (необов'язкове) відношення являє собою умовне відношення між сутностями. Істотно-обмежене відношення використовується коли відповідні сутності взаємно незалежні в системі.

Для ідентифікації вимог, в відповідності до яких сутності утворюють відношення, використовуються зв'язки. Кожен зв'язок з'єднує сутність та відношення і може бути направленим тільки від відношення до сутності.

Значення зв'язку характеризує його тип та, як правило, наступної множини: {"0 або 1", "0 або більше", "1", "1 або більше", "p : q" (діапазон)}.

Пара значень зв'язків, що належать до одного й того ж відношення, визначає тип цього відношення. Практика показала, що для більшості застосувань достатньо використати наступні типи відношень:

1). 1*1 (один-до-одного). Відношення даного типу використовуються, як правило, на верхніх рівнях ієрархії моделі даних, а на нижніх рівнях зустрічаються порівняно рідко.

2). 1*п (один-до-багатьох). Відношення даного типу використовується найбільш часто.

3). n*m (багато-до-багатьох). Відношення даного типу звичайно використовуються на ранніх етапах проектування з метою прояснення ситуації. В подальшому кожне з таких відношень повинно бути перетворене в комбінацію відношень типів 1 та 2 (можливо, з доданням допоміжних асоціативних сутностей та введенням нових відношень).

На рис. 18 наведена діаграма "сутність-зв'язок", що демонструє відношення між об' єктами на прикладі банкомату.

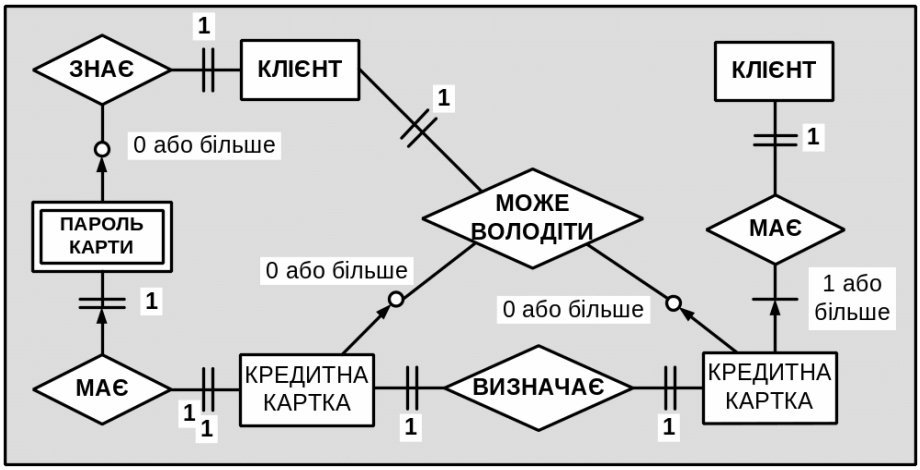


Рис.18. Діаграма “сутність-зв’язок”

Згідно до цієї діаграми кожен БАНК МАЄ один або більш РАХУНКІВ В БАНКУ. Крім того, кожен КЛІЄНТ МОЖЕ ВОЛОЛ1ТИ (водночас) однією або більше КРЕДИТНОЮ.

КАРТКОЮ КАРТКОЮ і одним або більш БАНКІВСЬКИМ РАХУНКОМ, кожен з яких ВИЗНАЧАЄ рівно одну КРЕДИТНУ КАРТКУ (відзначимо, що у клієнта може і не бути ані рахунку, ані кредитної карти). Кожна КРЕДИТНА КАРТА МАЄ лише один залежний від неї ПАРОЛЬ КАРТКИ, а кожен КЛІЄНТ ЗНАЄ (але може і забути) ПАРОЛЬ КАРТКИ.

Змістовий модуль 3.2 *Методології структурного підходу*

Тема 11 Методологія функціонального моделювання SADT

(4 години)

Мета: Розглянути детально методологію функціонального моделювання SADT.

План:

1. Методологія SADT
2. Контекстна діаграма
3. Приклад

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.
3. Дати відповіді на контрольні питання.

Питания для самоконтролю

1. Для чого використовується методологія SADT?
2. Перелічіть правила побудови моделі SADT.
3. Які види відносин тут використовуються?

Методологія SADT – це сукупність методів, правил і процедур, призначених для побудови моделей об'єкта предметної області. Функціональна модель SADT відображає функціональну структуру об'єкта, тобто вироблені ним дії й зв'язки між ними. Ця методологія є основою сімейства методологій моделювання IDEF. Для моделювання економічних об'єктів і процесів доцільно використовувати структурну методологію IDEF0, яка реалізує методику функціонального моделювання. Ця методика рекомендується для системного аналізу складних штучних систем управління, виробництва, бізнесу, що включають обладнання й спеціальне програмне забезпечення.

IDEF0 пов'язана з функціональними аспектами й відповідає на запитання "що робить система?". В результаті моделювання система постає перед аналітиками у вигляді набору взаємозалежних функцій. В основу IDEF0-методології закладена концепція:

1) блокове моделювання і його графічне представлення – графік блоків і дуг SADTдіаграми відображає функцію у вигляді блоку, а інтерфейси входу/виходу представляються дугами, що відповідно входять і виходять з нього;

2) лаконічність і точність – виконання правил SADT вимагає лаконічності й точності розроблюваної документації й іменування структурних елементів (блоків і стрілок), не накладаючи надмірних обмежень на дії аналітика;

3) передача інформації – модель повинна бути розроблена так, щоб надалі з нею могли працювати й розуміти, що в неї закладене;

4) строгість і формалізм – розробка моделей вимагає дотримання строгих формальних правил, що забезпечують переваги методології відносно однозначності й цілісності складних багаторівневих моделей;

5) ітеративне моделювання – розробка моделі являє собою покрокову, ітеративну процедуру;

6) відокремлення "організації" від "функцій" – виключення впливу організаційної структури на функціональну модель.

Основним структурним елементом IDEF0-методології є функція, яка визначає процеси, дії, операції. Ім'я функції задається дієсловом. Другий структурний елемент IDEF0-методології – це стрілки, які бувають різних видів:

1) вхідна стрілка – показує те, що необхідно для виконання функції (сировина, гроші);

2) вихідна стрілка – є результат виконання функції (прибуток, готова продукція);

3) стрілка-механізм – визначає виконувачів функції (співробітники, устаткування, технологія);

4) стрілка-керування – регламентує виконання функції (статут, ДСТУ).

Усі стрілки поділяють на два класи: внутрішні й граничні (рис. 1). У загальному вигляді IDEF0-модель являє собою набір погоджених діаграм, фрагмента тексту й глосарію (словника даних). Діаграма – частина моделі, що складається із взаємозалежних блоків.

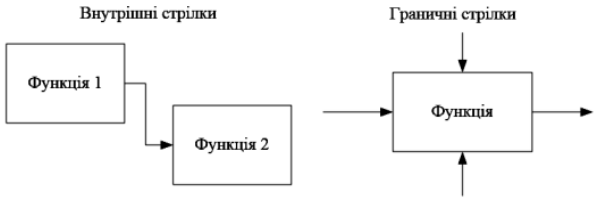


Рис. 1.

Контекстна діаграма – це діаграма самого верхнього рівня, що представляє систему загалом, у вигляді "чорного ящика", і зв'язує її із зовнішнім світом за допомогою інтерфейсних дуг. Контекстна діаграма складається з одного функціонального блоку, будь-якої кількості стрілок, мети моделювання й точки зору. Приклад контекстної діаграми забезпечення діяльності меблевого цеху з метою опису роботи цеху і точки зору експерта наведений на рис. 2. Мета моделювання вказує, для чого розробляється конкретна модель. Точка зору визначає посадову особу або підрозділ організації, з чийого погляду розробляється модель. Після розробки контекстної діаграми проводять процес декомпозиції. Декомпозиція – це розбивка функції на підфункції, тобто більш детальне її представлення.

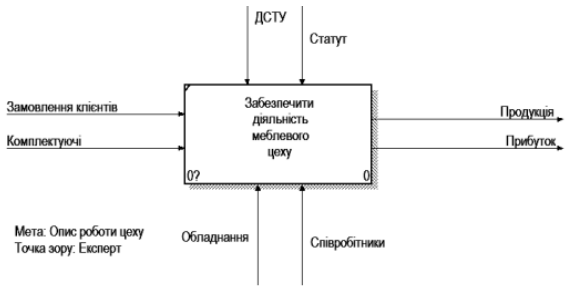


Рис. 2

Для збереження цілісності моделі на практиці використовується ICOM кодогенерація – процес, який автоматично переносить стрілки, приєднані до функціонального блоку, на діаграми декомпозиції (діаграми-нащадки). У такий спосіб підтримується зв'язок між діаграмами вищого рівня й діаграмами нижчого рівня ієрархії, зберігається цілісність моделі. Для тих самих цілей в IDEF0-моделі існує поняття тунелювання, або тунельної стрілки, яка на моделі відображається у вигляді круглих або квадратних дужок. Тунель у границі показує, що цієї стрілки немає на батьківській діаграмі, тобто на верхньому рівні декомпозиції вона неважлива. Між функціями в моделі існують такі типи відношень:

а) Домінування. Має два можливі значення: по-перше, блоки, розташовані вище, більш важливі й домінантні в рамках розглянутої предметної області, по-друге, блоки, розташовані вище, виконуються раніше за часом, наприклад, якщо розглянути початкову стадію роботи промислового підприємства, то першою

функцією буде проведення маркетингових досліджень, потім проектні роботи, після чого закупівля матеріалів, виробництво й продаж готової продукції.

б) Управління. Результат першої функції – це управляючий вплив для інших функцій. Приклад: перша функція – "розробити навчально-методичні вказівки", вихід функції – "учбово-методичні вказівки", тоді друга функція – "виконати лабораторну роботу" (рис. 3).

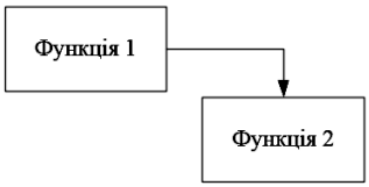


Рис. 3.

в) Вихід-вхід. Вихід однієї функції є входом для іншої (рис. 4).

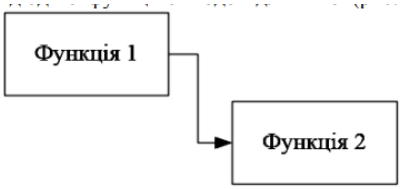


Рис. 4.

г) зворотний зв'язок (ЗЗ) управління. Вихід однієї функції є управлінням для іншої (рис. 5).

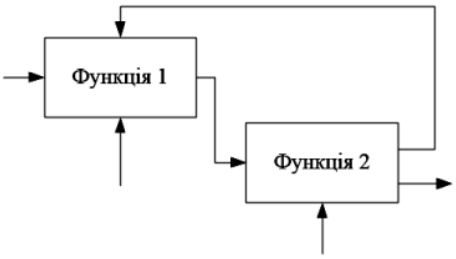


Рис. 5.

д) ЗЗ по входу. Є аналогом відношення типу "вихід-вхід" (рис. 6).

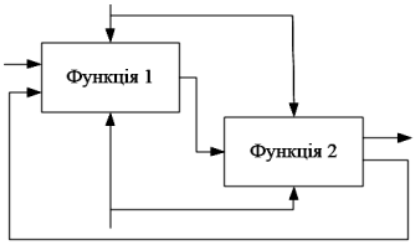


Рис. 6.

е) Вихід-механізм. Це досить непоширений тип відношень. Приклад: підприємство займається випуском продукції, а потім у своїй подальшій діяльності використовує це встаткування на інших етапах (рис. 7).

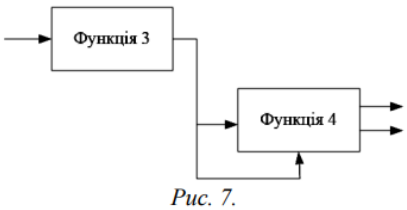


Рис. 7.

Правила побудови діаграм наступні:

1. До складу моделі обов'язково повинна входити контекстна діаграма.

2. Блоки на діаграмі повинні розташовуватися (переважно) по діагоналі (відношення домінування).

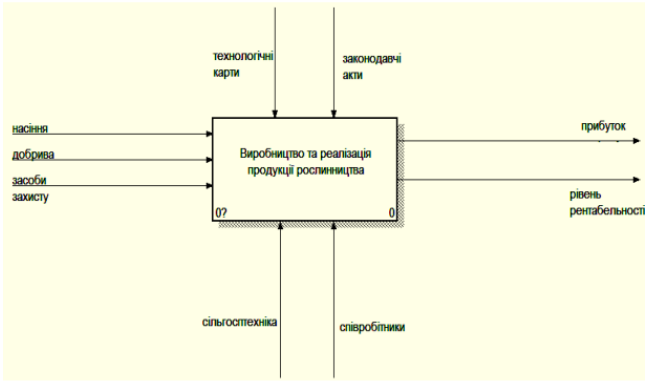
3. Неконтекстні діаграми повинні містити кількість функціональних блоків від 3 до 6.

4. Імена функцій і стрілок повинні бути унікальними. Імена функцій повинні бути задані дієсловом, імена стрілок – іменем-іменником.

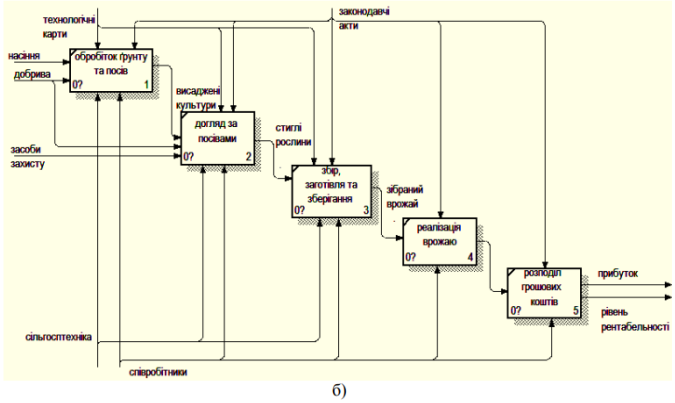
5. У будь-якого функціонального блоку обов'язково повинна бути хоча б одна стрілка-управління й одна стрілка-вихід. Стрілки-входу може не бути, але в цьому випадку стрілка-управління буде одночасно представляти управляючу й вихідну інформації.

6. При розробці моделі необхідно прагнути до зменшення кількості необов'язкових перетинань стрілок, мінімізувати число петель і поворотів кожної стрілки.

7. Стрілки повинні поєднуватися, якщо мають загальне джерело.



а)



б)

Рис. 8

На рис. 8 наведений приклад IDEF0-моделі діяльності підприємства вирощування й реалізації продукції (а – контекстна діаграма; б – діаграма декомпозиції).

Тема 12 Діаграми потоків даних

(4 години)

Мета: Розглянути побудову діаграми потоків даних використовуючи сервіс MindOnMind.

План:

1. Побудова ДПД

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Зробити опорний конспект лекції.

Давайте створимо ДПД, використовуючи інструмент для створення діаграм і розумових карт MindOnMind, інструмент номер один у мережі на сьогоднішній день.

Це також онлайн-конструктор діаграм потоку даних, який подобається всім. Загалом, він має всі параметри, попередні налаштування та гарні атрибути, які пропонує користувачам для створення як карт, так і діаграм.

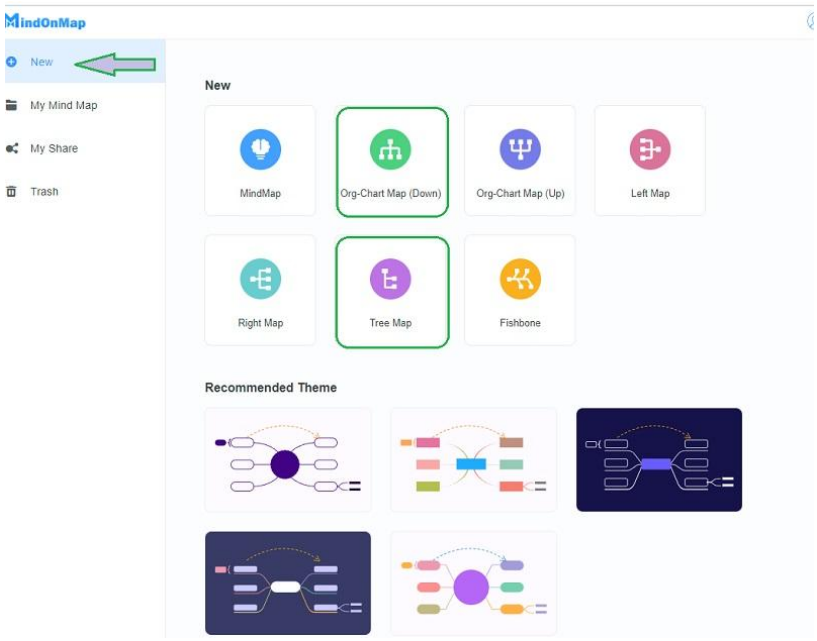
Крім того, MindOnMind може зберігати ваш шедевр у найбезпечнішій галереї, створеній з вашого облікового запису електронної пошти. Тим не менш, це дозволить вам експортувати всі ваші проекти, де ви зможете друкувати в будь-який час. Крім того, вам не потрібно турбуватися про його ціну, оскільки ви можете використовувати його необмежено, не витрачаючи ні копійки. І тому давайте створимо діаграму потоку даних онлайн, дотримуючись наведених нижче повних і вичерпних інструкцій.

1 Відвідайте веб-сайт

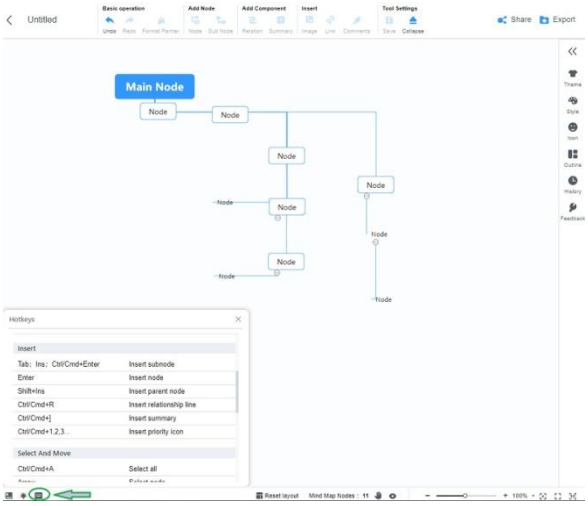
Відкрийте веб-переглядач на комп’ютері чи мобільному пристрої та відвідайте www.mindonmap.com. На початковій сторінці натисніть Створіть свою ментальну карту і розпочніть вхід за допомогою свого облікового запису Gmail.

2. Почніть з Нового

На наступній сторінці натисніть новий вкладку, а потім виберіть тему або діаграму з параметрів. Оскільки ми будемо робити ДПД, пропонуємо вам вибрати TreeMap або Карта організаційної діаграми (вниз).

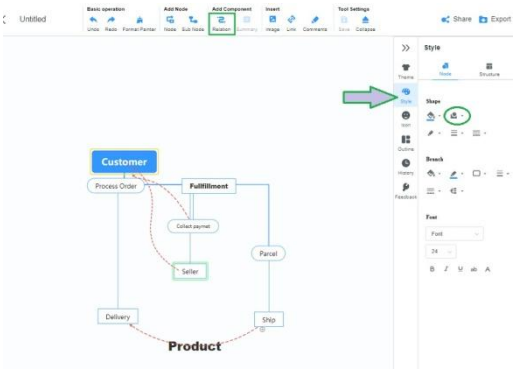


Ви можете почати навігацію цим інструментом діаграми потоку даних, коли досягнете основного полотна. Оскільки ви просто дивитеся на сингл Головний вузол, перейдіть до Гарячі клавіші опція, щоб побачити кнопки, які допоможуть вам розгорнути діаграму. Крім того, якщо вам потрібно виділити вузол, просто перетягніть його в потрібне місце.



Крок 3. Позначте позначення

Застосуйте позначення ДПД до ваших вузлів, змінюючи їх форми. Для цього перейдіть до Меню і клацніть Стиль. Потім перейдіть до Стиль фігури вибрати для вашої організації.

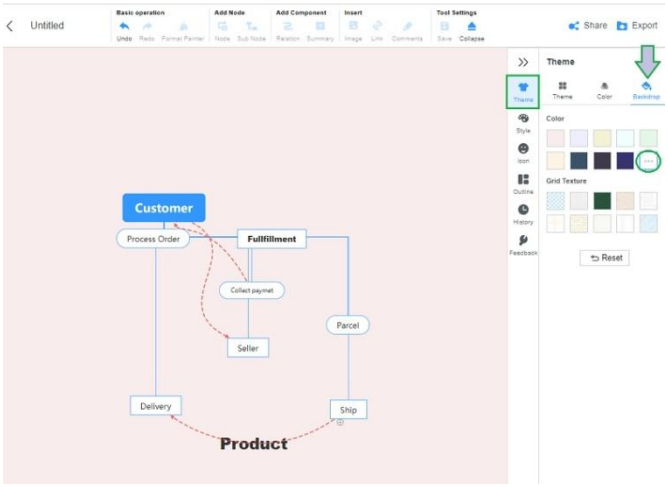


4 Додати сяйво

Перевагою цього інструменту є те, що ви можете створити сяючу діаграму, додавши до неї кольори.

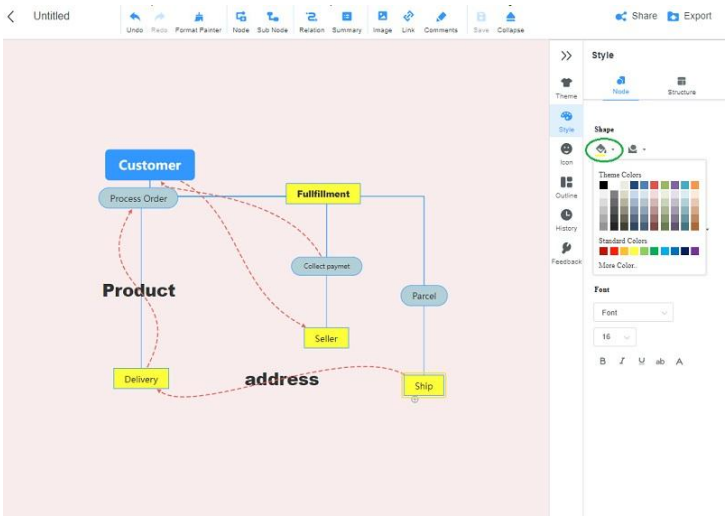
4.1. Змініть колір фону

Перейдіть до меню Бар, потім з Теми натисніть Задник. Виберіть їх серед різноманітних доступних тем. Крім того, натиснувши значок Сліпсис ви зможете персоналізувати тему відповідно до своїх уподобань.



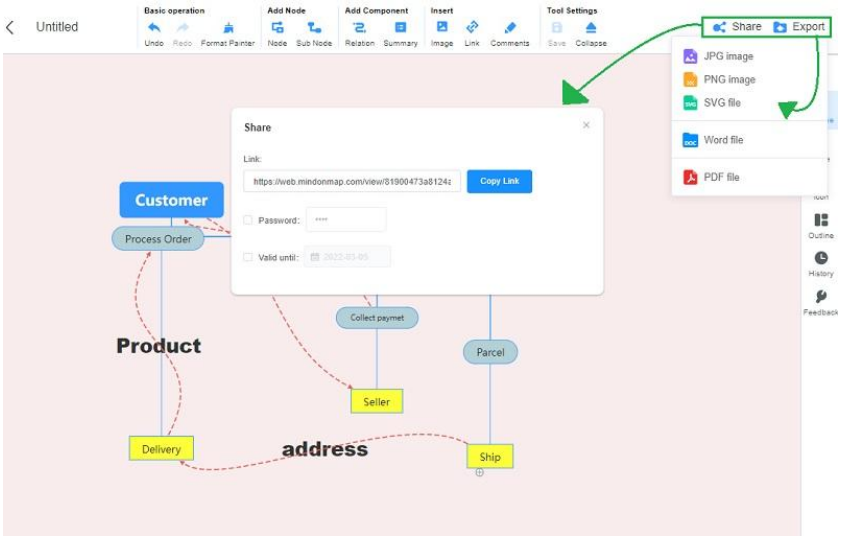
4.2. Додайте колір до понять

Щоб додати кольори до нотацій діаграми потоку даних, перейдіть до Стиль. Клацніть кожен вузол і натисніть Фарба значок біля стилю фігури. Звідти виберіть потрібний колір для заповнення позначення.



5 Поділитися або експортувати

Нарешті ви можете поділитися або експортувати свою діаграму, просто перейдіть до частини вгорі панелі меню. Якщо ви хочете надіслати діаграму своєму товаришу по команді для співпраці, натисніть Поділіться вкладка. З іншого боку, вдарте по Експорт кнопку, якщо ви прагнете отримати друковану копію зі свого пристрою.



Тема 13 Системний аналіз

(2 години)

Мета: Розглянути основні поняття системного аналізу: мета, задача, структура, система, системність. Розглянути класифікацію систем.

План:

1. Система та підсистема;
2. Мета та задача;
3. Структура систем;
4. Класифікація систем.

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що таке система та підсистема?
2. Що таке мета системи?
3. Дати визначення поняттю задача та розв’язок зачі.

4. В чому заключається складність дослідження погано формалізуємих систем?
5. Що таке структура системи?
6. Перелічіть та охарактеризуйте різні типи структур.
7. За якими критеріями відбувається класифікація системи?
8. В чому різниця між великими та складними ситемами?

Система - об'єкт, процес у якому елементи, що беруть участь, зв'язані деякими зв'язками й відносинами.

Підсистема - частина системи з деякими зв'язками й відносинами.

Будь-яка система складається з підсистем, кожна підсистеми будь-якої системи може бути розглянута сама як система.

Приклад. Наука - система, когнітивна система, що забезпечує одержання, перевірку, фіксацію (зберігання), актуалізацію знань суспільства. Наука має підсистеми: математика, інформатика, фізика, філологія й ін. Будь-яке знання існує лише у формі систем (систематизоване знання), а теорія - найбільш розвинена система їх організації в систему, що дозволяє не тільки описувати, але й пояснювати, прогнозувати події, процеси.

Мета - образ неіснуючого, але бажаного - з погляду завдання або розглянутої проблеми - стану середовища, тобто такого стану, який дозволяє вирішувати проблему при даних ресурсах. Це - опис, представлення якогось найбільш кращого стану системи.

Приклад. Основні соціально-економічні цілі суспільства:

- *економічний ріст;*
- *повна зайнятість населення;*
- *економічна ефективність виробництва;*
- *стабільний рівень цін;*
- *економічна свобода виробників і споживачів;*
- *слушний розподіл ресурсів і благ;*
- *соціально-економічна забезпеченість і захищеність;*
- *торговельний баланс на ринку;*
- *слушна податкова політика.*

Поняття мети конкретизується різними об'єктами й процесами.

Приклад. Ціль - функція (знайти значення функції). Ціль - вираження (знайти аргументи, що перетворюють вираження в тотожність). Ціль - теорема (сформулювати й/або довести теорему - тобто знайти умови перетворюючі сформульоване пропозицію в дійсне висловлення). Ціль - алгоритм (знайти, побудувати послідовність дій, продукций досягнення, що забезпечують, необхідного стану об'єкта або процесу перекладу його з вихідного стану у фінальне).

Цілеспрямована поведінка системи - поведінка системи (тобто послідовність прийнятих нею станів), що веде до мети системи.

Задача - якась безліч вихідних посилок (вхідних даних до завдання), опис мети, певної над безліччю цих даних і, може бути, опис можливих стратегій досягнення цієї мети або можливих проміжних станів досліджуваного об'єкта.

Приклад. Глобальне економічне завдання, з яким зустрічається будь-яке суспільство - коректне розв'язання конфлікту між фактично необмеженим людським споживанням товарів і послуг і обмеженими ресурсами (матеріальними, енергетичними, інформаційними, людськими), які можуть бути актуалізовані для задоволення цих потреб. При цьому розглядають наступні основні економічні завдання суспільства:

Що робити (які товари й послуги)?

Як робити (яким образом і де)?

Для кого робити (для якого покупця, ринку)?

Розв'язати задачі - означає визначити чітко ресурси й шляхи досягнення зазначеної мети при вихідних посилках.

Розв'язок задачі - опис або представлення того стану завдання, при якому досягається зазначена мета; розв'язком завдання називають і сам процес знаходження, опису цього стану.

Приклад. Розглянемо наступну "задачу": розв'язати квадратне рівняння (або скласти алгоритм його розв'язку). Така постановка проблеми неправильна, тому що не поставлена мета, завдання, не зазначене, як розв'язати завдання й що розуміти в якості розв'язку завдання. Наприклад, не зазначені загальний вид рівняння - наведене або ж не наведене рівняння (а алгоритми їх розв'язки - різні!).

Завдання також поставлене не повністю - не зазначений тип вхідних даних: речовинні або комплексні коефіцієнти рівняння, не визначені поняття розв'язку, вимоги до розв'язку, наприклад, точність кореня (якщо корінь вийде ірраціональним, а потрібно було визначити його з деякою точністю, то завдання обчислення наближеного значення кореня - автономне, не дуже просте завдання). Крім того, можна було б указати можливі стратегії розв'язку - класичне (через дискримінант), по теоремі Виета, оптимальним співвідношенням операндов і операції.

Опис (специфікація) системи - це опис усіх її елементів (підсистем), їхніх взаємозв'язків, мети, функції при деяких ресурсах тобто всіх припустимих станів.

Якщо вхідні посилки, мета, задача, розв'язок або, можливо, навіть саме поняття розв'язку погано описувані, то ці завдання називаються погано формалізуємими. Тому при розв'язку таких завдань доводиться розглядати цілий комплекс формалізованих завдань, за допомогою яких можна досліджувати цю погано формалізовану задачу. Складність дослідження таких завдань - у необхідності обліку різних, а часто й суперечливих критеріїв визначення, оцінки розв'язку задачі.

Приклад. Погано формалізуемими будуть, наприклад, завдання відновлення "розмитих" текстів, зображень, складання навчального розкладу в будь-якому великому вузі, складання "формули інтелекту", опису функціонування мозку, соціуму, перекладу текстів з однієї мови на іншій за допомогою ЕОМ і ін.

Структура - це все те, що вносить порядок у безліч об'єктів, тобто сукупність зв'язків і відносин між частинами цілого, необхідні для досягнення мети.

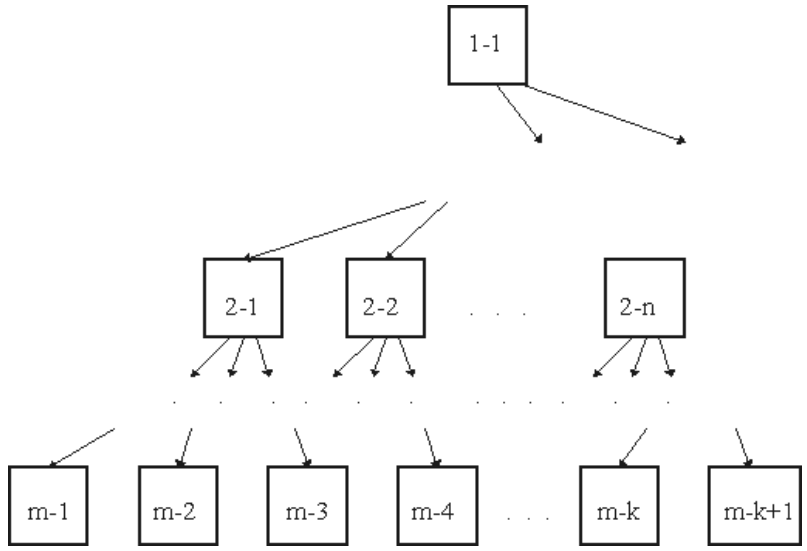
Приклад. Прикладами структур можуть бути структура звивин мозку, структура студентів на курсі, структура державного устрою, структура кристалічних ґрат речовини, структура мікросхеми й ін. Кристалічні ґрати алмаза - структура неживої природи; бджолині соти, смуги зебри - структури живої природи; озеро - структура екологічної природи; партія (суспільна, політична) - структура соціальної природи; Всесвіт - структура як живої й неживої природи.

Структури систем бувають різного типу, різної топології (або ж просторової

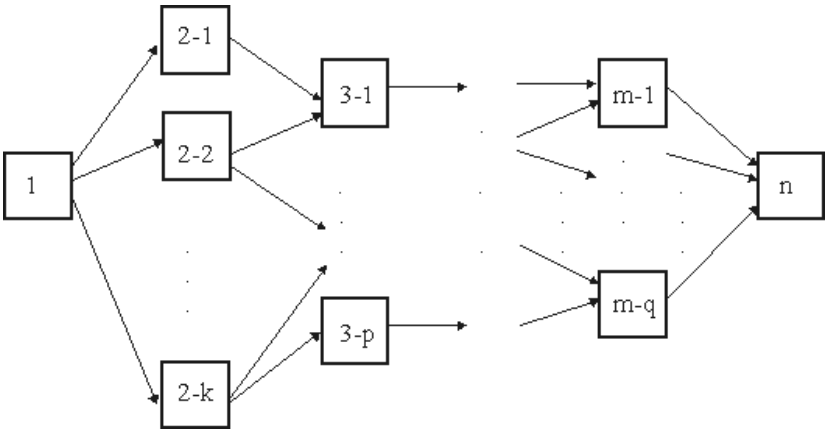
структури). Розглянемо основні топології структур (систем).

Лінійні структури:

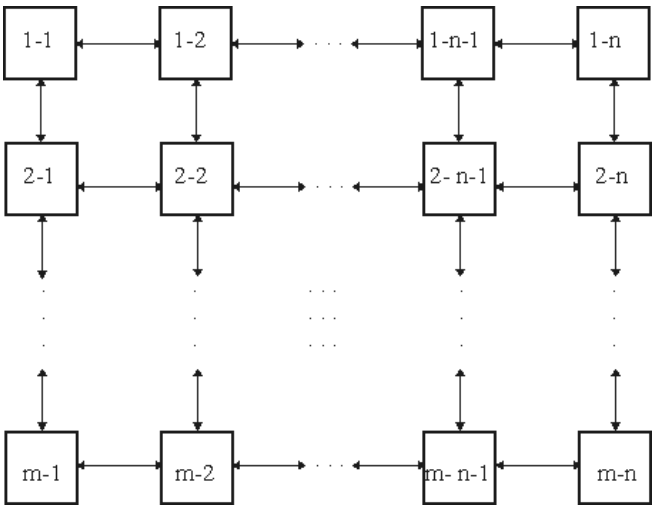
Ієрархічні древовидні структури:



Мережева структура:



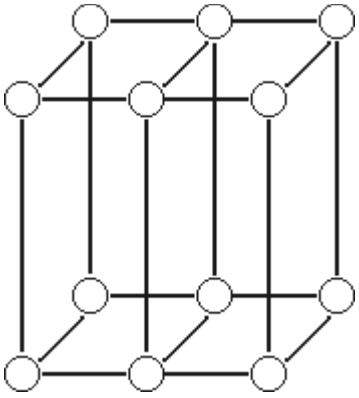
Матрична структура:



Приклад. *Прикладом лінійної структури є структура станцій метро на одній (не кільцевій) лінії. Прикладом ієрархічної структури є структура керування вузом: "Ректор - Проректора - Декани - Завідувачі кафедр і підрозділами - Викладачі кафедр і співробітники інших підрозділів". Приклад мережної структури - структура організації строительно - монтажних робіт при будівництві будинку: деякі роботи, наприклад, монтаж стін, благоустрій території й ін. можна виконувати паралельно. Приклад матричної структури - структура працівників відділу НДІ (Научно-дослідницький інститут) виконуючих роботи з одній і тій же темі.*

Крім зазначених основних типів структур використовуються й інш, що утворюються за допомогою їх коректних комбінацій - з'єднань і вкладень.

Приклад. *"Вкладення друг у друга" площинних матричних структур може привести до більш складної структури - структурі просторовій матричній.*



Класифікація систем.

Класифікацію систем можна здійснити за різними критеріями. Її часто жорстко неможливо проводити й вона залежить від мети й ресурсів. Приведемо основні способи класифікації (можливі й інші критерії класифікації систем).

1. По відношенню системи до навколишнього середовища:
   - відкриті (є обмін з навколишнім середовищем ресурсами);
   - закриті (немає обміну ресурсами з навколишнім середовищем).
2. По походженню системи (елементів, зв'язків, підсистем):
   - штучні (знаряддя, механізми, машини, автомати, роботи і т.д.);
   - природні (живі, неживі, екологічні, соціальні і т.д.);
   - віртуальні (уявлювані й, хоча вони в дійсності реально не існуючі, але функціонуючі так само, як і у випадку, якби вони реально існували);

- змішані (економічні, биотехнические, організаційні і т.д.).

3. По опису змінних системи:
   - с якісними змінними (що мають тільки лише змістовний опис);
   - с кількісними змінними (що мають дискретно або безупинно описувані кількісним образом змінні);
   - змішаного (кількісно - якісний опис).
4. По типу опису закону (законів) функціонування системи:
   - типу "Чорний ящик" (невідомий повністю закон функціонування системи; відомі тільки вхідні й вихідні повідомлення системи);
   - не параметризованні (закон не описаний, описуємо за допомогою хоча б невідомих параметрів, відомі лише деякі апріорні властивості закону);
   - параметризованні (закон відомий з точністю до параметрів і його можливо від ADE нести до деякого класу залежностей);
   - типу "Білий (прозорий) ящик" (повністю відомий закон).
5. По способу керування системою (у системі):
   - керовані ззовні системи (без зворотного зв'язку, регульовані, керовані структурно, информационно або функціонально);
   - керовані зсередини (самокеровані або саморегулювальні - програмно керовані, регульовані автоматично, адаптируємі - що пристосовуваються за допомогою керованих змін станів і системи, що самоорганізуємі – змінюючі в часі та просторі свою структурі найбільш оптимально
   - с комбінованим керуванням (автоматичні, напівавтоматичні, автоматизовані, організаційні).

<u>Приклад.</u> Розглянемо екологічну систему "Озеро". Це відкрита, природнього походження система, змінні якої можна описувати змішаним образом (кількісно і якісно, зокрема , температура водойми - кількісно описувана характеристика), структуру мешканців озера можна описати і якісно, і кількісно, а красу озера можна описати якісно. По типу опису закону функціонування системи, цю систему можна віднести до не параметризованным у цілому, хоча можливе виділення підсистем різного типу, зокрема , різного опису підсистеми

"Водорості", "Риби", " струмок, Що Впадає, ", " струмок, Що Випливає, ", "Дно", "Беріг" і ін. Система "Комп'ютер" - відкрита, штучного походження, змішаного опису, параметризованная, керована ззовні (програмно). Система "Логічний диск" - відкрита, віртуальна, кількісного опису, типу "Білий ящик" (при цьому вміст диска ми в цю систему не включаємо!), змішаного керування. Систем "Фірма" - відкрита, змішаного походження (організаційна) і опису, керована зсередини (адаптируемая, зокрема , система).

Система називається **великою**, якщо її дослідження або моделювання утруднене через велику розмірність, тобто безліч станів системи S має більшу розмірність.

Система називається **складною**, якщо в ній не вистачає ресурсів (головним чином, - інформаційних) для ефективного опису (станів, законів функціонування) і керування системою - визначення, опису керуючих параметрів або для прийняття розв'язків у таких системах (у таких системах завжди винна бути підсистема ухвалення рішення).

Приклад. Складними системами є, наприклад, хімічні реакції, якщо їх розглядати на молекулярному рівні; клітка біологічного утвору, розглянута на метаболическом рівні; мозок людини, якщо його розглядати з погляду виконуваних людиною інтелектуальних дій; економіка, розглянута на макрорівні (т.е макроекономіка); людське суспільство - на політико-релігійно- культурному рівні; ЕОМ (особливо, - п'ятого покоління), якщо її розглядати як засіб одержання знань; мова, - у багатьох аспектах.

Тема 14 Забезпечення якості ПЗ

(4 години)

Мета: Ознайомитись з поняттям якості програмного продукту, розглянути характеристики якості ПП, розглянути поняття тестування ПЗ.

План:

1. Якість ПП;
2. Тестування ПЗ;
3. Види тестування ПЗ.

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.

2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що таке якісний програмний продукт?
2. Назвіть характеристики якості.
3. Що таке тестування ПЗ?
4. Назвіть види тестування ПЗ.

Забезпечення якості ПЗ

Якість програмного забезпечення (Software quality) асоціація IEEE визначає як ступінь відповідності програмного забезпечення встановленій комбінації властивостей. У Міжнародному стандарті якості ISO 9000:2007 це поняття визначається як сукупність характеристик ПЗ, які забезпечують встановлені та очікувані вимоги . До характеристик якості ПЗ відносять (рис. 1):

• функціональність – виконання заявлених функцій, відповідність стандартам, функціональна сумісність, безпека, точність;

• надійність – стійкість до відмов, можливість відновлення, завершеність;

• ефективність – економія часу, ефективність використання;

• зручність у використанні – ергономічність, інтуїтивна зрозумілість, повна документація;

• зручність для супроводу – стабільність, придатність для контролю та внесення змін;

• портативність – зручність установки, можливість заміни, сумісність.

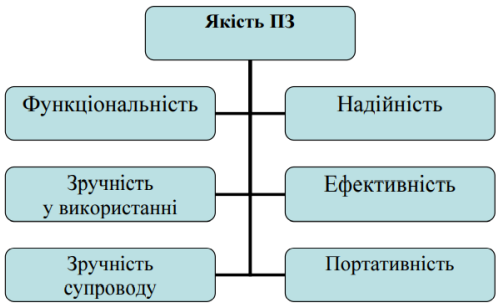


Рисунок 1 – Характеристики якості ПЗ

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

• повнота та коректність документації;

• коректність процедур встановлення та запуску;

• ергономічність використання;

• повнота тестування.

У 80-х рр. XX ст. розмір проектів із створення програмних систем зріс настільки, що вручну стало неможливо тестувати ПЗ, тому активно стали

розроблятися засоби автоматизації процесу тестування. Через дестиліття поняття якості ПЗ розширилося настільки, що було виділено окремий вид діяльності при створенні ПЗ – забезпечення якості (Quality assurance, QA). На цей час автоматизоване тестування значно поширене, а засоби автоматизованого тестування часто вбудовані у середовище програмування.

Для виконання заходів забезпечення якості ПЗ розробники часто використовують спеціальні групи контролю якості, які мають назву QA. Група QA всередині компанії фактично виконує роль вимогливого користувача. У деяких компаніях заборонено неформальне спілкування між групою розробників та групою тестувальників. Від відповідальності групи Якість ПЗ Функціональність Надійність Портативність Зручність у використанні Ефективність Зручність супроводу QA залежить успіх продукту. Якщо ПЗ не сподобалося, з будь-яких причин користувачам продукт назавжди втрачає репутацію і споживача.

Незнайдені на стадії розроблення помилки коштують дорого. Традиційна стратегія розроблення ПЗ підпорядковується фундаментальному правилу, що визначає, що впродовж роботи над проектом вартість внесення змін у створюване ПЗ збільшується за експонентою (рис. 2).

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

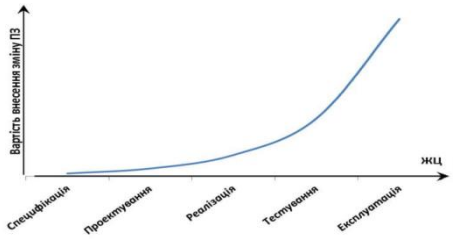


Рисунок 2 – Залежність вартості змін проектів від часу

Тестування ПЗ

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

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

Тестування потребує від виконавця в пешу чергу уважності та педантичності. Тестувальник повинен шукати недоліки будь-де в системі. Якщо початкове тестування для налагодження коду виконує сам програміст, то наступні етапи перевірки повинні готувати і виконувати інші особи, щоб перевірка була повноцінною, а не тільки для очікуваних проблемних місць.

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

• на 1000 рядків коду програміст у середньому робить 100 помилок, 70% з яких усуваються на стадії налагодження коду;

• при системному тестуванні у відділі контролю якості на ліквідацію однієї помилки потрібно від чотирьох до шістнадцяти годин;

• для усунення помилки, що була виявлена у ході експлуатації, потрібно від 33 до 88 годин.

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

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

Види тестування можна класифікувати за різними показниками:

1. **За рівнем знання системи**:

• чорний ящик (Black-box testing) – без перегляду програмного коду;

• білий ящик (White-box testing) – тестування із вивченням коду програми;

• сірий ящик (Gray-box testing) – тестування із частковим вивченням коду програми.

2. **За об’єктом тестування**:

• функціональне (Functional testing) – перевірка виконання заданих функцій;

• тестування продуктивності (Performance testing) – перевірка стабільності роботи програми або її частини при заданому навантаженні (Load testing), при перевантаженні (Stress testing) та тривалому середньому навантаженні (Stability testing);

• юзабіліті-тестування (Usability testing) – перевірка ергономічності ПЗ;

• перевірка інтерфейсу користувача (UI testing);

• перевірка безпеки (Security testing) – тестування роботи системи з точки зору безпеки інформації;

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

3. **За часом виконання тестування:**

• альфа-тестування (Alpha testing) – перевірка роботи ПЗ перед передачею в експлуатацію силами компанії-розробника;

• півгодинне тестування (Smoke testing) – коротка перевірка функцій системи, як правило, один тест для кожної функції;

• тестування нової функціональності (New feature testing);

• регресійне тестуванн (Regression testing) – перевірка роботи вже протестованих модулів;

• тестування під час передачі ПЗ (Acceptance testing);

• бета-тестування (Beta testing) – тестування ПЗ перед випуском сторонніми силами.

4. **За ступенем ізольованості компонентів**:

• компонентне тестування (Component / Unit testing) – перевірка коректності окремого модуля або компонента системи;

• інтеграційне тестування (Integration testing) – перевірка роботи модулів, об'єднаних у групу;

• системне тестування (System / end-to-end testing) – перевірка роботи зібраного ПЗ з метою встановлення відповідності очікуваним функціям.

5. **За ступенем автоматизації**:

• ручне тестування (Manual testing);

• автоматизоване тестуванняя (Automated testing);

• напівавтоаматизоване тестування (Semiautomated testing).

6. **За ступенем підготовленості до тестування**:

• формальне тестування (Formal testing) – тестування відповідно до плану, встановлених процедур;

• інтуїтивне тестування (Ad hoc testing) – тестування без попереднього плану дозволяє на раніх стадіях виявляти помилки.

7. **За ознакою позитивності сценаріїв**:

• перевірка позитивних сценаріїв (Positive testing);

• перевірка негативних сценаріїв (Negative testing).

Тема 15 Методи оцінки якості тестування. Види тестів

(4 години)

Мета: Ознайомитись з методами оцінки якості тестування та розглянути види тестів.

План:

1. Методи оцінки якості тестування;
2. Види тестів;

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що таке база помилок?

2. Що записується в базу помилок?
3. Що таке тестування?
4. Які елементи включає етап тестування?
5. На яких рівнях ведеться тестування?

Методи оцінки якості тестування

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

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

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

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

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

− low: косметичні поліпшення – помилка залишиється до наступної версії.

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

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

Види тестів

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

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

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

• середній час вирішення завдання;

• максимальний обсяг необхідний оперативної пам'яті;

• показники завантаження зовнішніх пристроїв необхідних для оцінки вартості рішення задачі.

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

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

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

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

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

Етап тестування включає три основних елементи:

1. генерація тестових прикладів, контроль процесу тестування;

2. аналіз вихідних і проміжних результатів програми;

3. врахування внеску кожного тестового прикладу в процес тестування.

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

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

Тестування ведеться на чотирьох рівнях:

1. рівень окремого програмного компонента (модуля);

2. рівень сполучень, на якому шукаються помилки між-компонентного інтерфейсу;

3. рівень зовнішніх функцій, на якому шукаються розбіжності програмних функцій і зовнішніх специфікацій програм;

4. комплексне тестування, випробування програмного комплексу на відповідність вихідним цілям.

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

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

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

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

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

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

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

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

• Метод дозволяє поєднати тестування модуля, тестування сполучень і тестування вхідних функцій.

• Рівномірний розподіл роботи з тестування протягом усього періоду створення комплексу - це дозволяє виявити помилки в головному модулі на ранній стадії розробки.

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

Методи тестування компонентів

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

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

Для оцінки повноти тестування використовують три критерії:

1. тестування вважається закінченим, якщо кожен оператор був виконаний хоча б раз;

2. тестування вважається закінченим, якщо в процесі вирішення тестових прикладів по кожній дузі блок-схеми програми був здійснений хоча б один перехід;

3. тестування закінчується, якщо в процесі вирішення тестового прикладу кожен шлях від входу до виходу пройдений хоча б раз.

Налагодження - процес пошуку та усунення помилок у програмі, що спирається на результати самої програми, повідомлення транслятора, і оперативної системи. Важливою особливість процесу налагодження є можливість робити експерименти на ЕОМ з метою виявлення помилок. Але експеримент повинен вестися в суворій відповідності з планом.

Процес налагодження полягає в багаторазовому повторенні трьох етапів:

1. виявлення помилки;

2. локалізація помилки;

3. виправлення помилки.

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

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

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

Тема 16 Верифікація програм

(2 години)

Мета: Ознайомитись з поняттям верифікації програм. Розглянути етапи верифікації

План:

1. Верифікація програм;

Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1) Які типи вимог до ПЗ можна виділити? Як виконується опис цих вимог?

2) Як пов’язані між собою строки проектних робіт, їх обсяг та вартість? Яким чином, керуючи цими показниками, можна забезпечити якість проекту?

3) Які елементи повинен містити проектний план?

4) Які форми планів проектних робіт вам відомі? У чому принципова різниця між ними?

5) Охарактеризуйте принципи керування проектом.

6) Які складові методології розроблення потрібно враховувати при її виборі?

7) З якими ризиками стикаються розробники ПЗ?

8) Перелічіть та коротко опишіть характеристики якості ПЗ.

9) Що таке Quality assurance, чим воно відрізняється від тестування?

10) Опишіть види тестування.

11) Яку інформацію доцільно зберігати в базі даних помилок?

Верифікація програм

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

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

Верифікація – ідея математичного доказу коректності програм.

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

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

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

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

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

• написати програму;

• визначити вхідні і вихідні умови для цієї програми;

• розрізати цикли і забезпечити кожну точку розрізу індуктивним затвердженням;

• отримати верифікаційні умови;

• довести істинність верифікаційних умов; якщо це не вдається, то не виключено, що якесь індуктивне твердження записано невірно або програма містить помилку.

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

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

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

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

Список використаних джерел інформації

1) І. Бородкіна, Г. Бородкін «Інженерія програмного забезпечення. Навчальний посібник», Центр учбової літератури, 2021 – 204 с.;

2) Р. Мартін «Чиста архітектура», Фабула, 2019 – 416 с.;

3) Ю. Рамський «Проектування й опрацювання баз даних: Посібник для вчителів», Навчальна книга Богдан, 416 с;

4) Ерік Еванс «Предметно-орієнтоване проектування (DDD): структуризація складних програмних систем», Діалектика, 2016 – 448 с..

5) Ю. Грицюк «Аналіз вимог до програмного забезпечення», Львівська Політехніка, 2018 – 456 с.

6) О. Перевозчикова «Інформаційні системи і структури даних», Києво-Могилянська академія, 2007 – 288с.

7) ГНАТОВСЬКА Г.А. Конспект лекцій з дисципліни «Технологія створення програмних продуктів», 2015 – 98 с