Конспект лекцій

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

Конспект лекцій

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 2022

Конспект лекцій 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

Мета розділу - дати вступ в предмет "основи програмної інженерії". Вивчивши цей розділ студенти повинні:

- мати поняття про те, що таке інженерія програмного забезпечення й чому вона важлива;
- знати відповіді па ключові питання, що ставляться до інженерії програмного забезпечення;
- розуміти етичні й професійні проблеми, що ставляться перед фахівцями із програмного забезпечення.

Змістовий модуль 1.1: Огляд інженерії програмного забезпечення

Тема 1.1.1 Вступ. Основні визначення і поняття інженерії програмного забезпечення

(2 години)

Мета: Пояснити студентам що таке інженерія програмного забезпечення. Ознайомити студентів з основними поняттями інженерії програмного забезпечення.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План:

1. Програмне забезпечення та програмні продукти.
2. Інженерія програмного забезпечення.
3. Автоматизована розробка програмного забезпечення (CASE)

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

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

Контрольні питання:

1. На які типи діляться програмні продукти?
2. Моделі технологічного процесу створення ПЗ?
3. Моделі розробки ПЗ?

Метою інженерії програмного забезпечення є ефективне створення програмних систем. Програмне забезпечення абстрактно й нематеріально. Воно не має фізичної природи, відкидає фізичні закони й не піддається обробці виробничими процесами. Такий спрощений погляд на ПЗ показує, що не існує фізичних обмежень на потенційні можливості програмних систем. З іншого боку, відсутність матеріального наповнення часом робить ПЗ надзвичайно складним і, отже, важким для розуміння "об'єктом".

Багато хто ототожнюють термін *програмне забезпечення* з комп'ютерними програмами. Це досить обмежене подання. Програмне забезпечення - це не тільки програми, але й вся супутня документація, а також конфігураційні дані, необхідні для коректної роботи програм. *Програмні системи* складаються із сукупності програм, файлів конфігурації, необхідних для установки цих програм, і документації, що описує структуру системи, а також містить інструкції для користувачів, що пояснюють роботу із системою, і часто адресу WEB-Вузла, де користувач може знайти саму останню інформацію про даний програмний продукт.

Фахівці із програмного забезпечення розробляють програмні продукти, тобто таке ПЗ, яких можна продати споживачеві. Програмні продукти діляться на два типи.

1. *Загальні програмні продукти.* Це автономні програмні системи, які створені компанією по виробництву ПЗ й продаються на відкритому ринку програмних продуктів будь-якому споживачеві, здатному їх купити. Іноді їх називають "коробковим ПЗ". Прикладами цього типу програмних продуктів можуть служити системи керування базами даних, текстові процесори, графічні пакети й засоби керування проектами.

2. *Програмні продукти, створені на замовлення.* Це програмні системи, які створюються за замовленням певного споживача. Таке ПЗ розробляється спеціально для даного споживача згідно з укладеним контрактом. Програмні продукти цього типу включають системи керування для електронних пристроїв, системи підтримки певних виробничих або бізнес-процесів, системи керування повітряним транспортом і т.п.

Важлива відмінність між цими типами програмних продуктів полягає в тім, що при створенні загальних програмних продуктів специфікація вимог на них розробляється компанією-виробником. Для замовлених програмних продуктів специфікація звичайно розробляється організацією, що купує даний продукт. *Специфікація необхідна розробникам ПЗ для створення будь-якого програмного продукту.*

Інженерія програмного забезпечення - це інженерна дисципліна, що охоплює всі аспекти створення ПЗ від початкової стадії розробки системних вимог через створення ПЗ до його використання.

Створення ПЗ - це сукупність процесів, що приводять до створення програмного продукту.

Абревіатура CASE позначає Computer-Aided Software Engineering - автоматизована розробка програмного забезпечення. Під цим розуміється широкий спектр програм, що застосовуються для підтримки й супроводу різних етапів створення ПЗ: аналізу системних вимог, моделювання системи, її налагодження й тестування й ін. Всі сучасні методи створення ПЗ використовують відповідні CASE-Засоби: редактори нотацій, застосовуваних для опису моделей, модулі аналізу, що перевіряють відповідність моделі правилам методу, і генератори звітів, що допомагають при створенні документації на розробляєме ПЗ. Крім того, CASE-Засоби можуть включати генератор коду, що автоматично генерує вихідний код програм на основі моделі системи, а також посібник користувача.

CASE-Засоби, призначені для аналізу специфікацій і проектування ПЗ, іноді називають CASE-Засобами верхнього рівня, оскільки вони застосовуються на початковій стадії розробки програмних систем. У той же час CASE-Засоби, націлені на підтримку розробки й тестування ПЗ, тобто відладчики, системи

аналізу програм, генератори тестів і редактори програм, часом називають CASE-Засобами нижнього рівня.

Основні поняття лекції:

Програмне забезпечення - це комп'ютерні програми й відповідна документація. Програмні продукти розробляються або по приватному замовленню, або для продажу на ринку програмних продуктів.

Інженерія програмного забезпечення - це інженерна дисципліна, що охоплює всі аспекти розробки програмного забезпечення.

Комп'ютерна наука - це теоретична дисципліна, що охоплює всі сторони обчислювальних систем, включаючи апаратні засоби й програмне забезпечення; інженерія програмного забезпечення - практична дисципліна створення й супроводу програмних систем.

Методи інженерії програмного забезпечення - це програмні системи, призначені для автоматизації процесу створення ПЗ.

CASE (Computer-Aided Software Engineering - автоматизоване проектування й створення ПЗ) - це програмні системи, призначені для автоматизації процесу створення ПЗ. CASE - засоби часто використовують, як основу методів інженерії програмного забезпечення.

Ознаки якісного ПЗ - програмні продукти повинні задовольняти вимогам функціональності й ефективності (з погляду користувача), а також бути надійними, зручними в експлуатації й мати можливості для модернізації.

Тема 1.1.2 Інформаційні системи

(2 години)

Мета: Ознайомити студентів з поняттям інформаційної системи. Розібрати що таке інформація і в яких фазах буває. Розглянути класифікацію систем.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Поняття інформаційної системи.
2. Поняття системи.
3. Властивості системи.
4. Класифікація інформаційних систем.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків заняття;

VI. Домашнє завдання:
   1. Ознайомитись з теоретичним матеріалом теми.
   2. Вивчити основні поняття лекції.
   3. Відповісти на контрольні питання.

Контрольні питання:

1. З яких компонентів складається система?
2. Які основні властивості має система?
3. Як можна класифікувати інформаційну систему?

Інформаційна система – це сукупність засобів і методів, що використовуються для обробки даних. Вона забезпечує збір, збереження, обробку,

пошук і видачу даних, необхідних в процесі ухвалення рішень. Вихідною продукцією ІС є інформація, на основі якої ухвалюються рішення.

Інформація – це сукупність знань про фактичні дані і залежності між ними.

Виділяють три *фази* існування інформації:

1. інформація, в асимільованій фазі – представлення повідомлень в свідомості людини, накладена на систему його понять і оцінок;

2. документована інформація – відомості, зафіксовані в знаковій формі на фізичних носіях;

3. інформація, що передається – відомості, що розглядаються у момент передачі інформації від джерела до приймача.

Поняття «*система*» охоплює комплекс взаємозв'язаних елементів, що діють як єдине ціле.

Система включає наступні *компоненти*:

1. структуру – це безліч елементів системи і взаємозв'язків між ними;

2. входи і виходи – матеріальні потоки або потоки повідомлень, що поступають в систему або виводяться нею;

3. закон поведінки системи – функція, що зв'язує зміни входу і виходу системи;

4. мета і обмеження.

Основним компонентом будь-який ІС є інформація, що зберігається в ній. Інформація в цьому фонді певним чином організована. Експлуатація ІС пов'язана з виконанням процедур пошуку, оновлення і обробки інформації. Початковими даними для ІС є питання, по яких організовується пошук і подальша обробка вибраних даних.

Властивості системи:

1. відносність – встановлює, що склад елементів, взаємозв'язків, входів, виходів, цілей і обмежень залежить від мети дослідження;

2. подільність – означає, що систему можна представити, як сукупність самостійних частин (підсистем), кожна з яких може розглядатися як система;

3. цілісність – указує на узгодженість цілей функціонування всієї системи з цілями її підсистем і елементів.

ІС можна класифікувати по цілому ряду ознак. У основу даних класифікацій покладені найбільш істотні ознаки, що визначають функціональні можливості і особливості побудови сучасних систем.

1. За типом даних, що зберігаються, ІС діляться на:

- фактографічні;
- документальні.

Фактографічні системи призначені для зберігання структурованих даних у вигляді чисел текстів. Над такими даними можна виконати різні операції.

У документальних системах інформація представлена у вигляді документа, що складається з найменувань, описів, рефератів і текстів. Пошук за неструктурованими даними здійснюється з використанням семантичних ознак. Відібрані документи надаються користувачеві, а обробка даних в таких системах практично не проводиться.

2. Грунтуючись на ступені автоматизації інформаційних процесів, ІС діляться на:

- ручні;
- автоматичні;
- автоматизовані.

Ручні ІС характеризуються відсутністю сучасних технічних засобів обробки інформації і виконанням всіх операцій людиною.

У автоматичних ІС всі операції по переробки інформації виконуються без участі людини.

Автоматизовані ІС припускають участь в процесі обробки інформації і людини, і технічних засобів, причому головна роль у виконанні рутинних операцій відводиться комп'ютеру. Саме цей клас систем відповідає сучасному представленню поняття ІС.

3. Залежно від характеру обробки даних ІС діляться на:

1. інформаційно-пошукові;
2. інформаційно-вирішальні.

Інформаційно-пошукові проводять введення, систематизацію, зберігання, видачу інформації по запиту користувача без складних перетворень даних.

Інформаційно-вирішальні здійснюють, крім того операції обробки інформації по певному алгоритму.

4. По характеру використання вихідної інформації такі системи прийнято ділити на:

1. системи керівники;
2. системи що радять.

Результуюча інформація керівників ІС трансформується в ухвалювані людиною рішення. Для цих систем характерні завдання розрахункового характеру і обробка великих об'ємів даних.

Що радять ІС виробляють інформацію, яка приймається людиною до уваги і враховується при формуванні управлінських рішень, а не ініціює конкретні дії. Ці системи імітують інтелектуальні процеси обробки знань, а не даних.

5. Залежно від сфери застосування розрізняють наступні класи ІС:

1. ІС організованого рівня призначені для автоматизації функцій управлінського персоналу, як промислових підприємств, так і не промислових об'єктів. Основними функціями подібних систем є оперативний контроль і регулювання, оперативний облік і аналіз, перспективне і оперативне планування, бухгалтерський облік, управління збутом, постачання і ін. економічні і організаційні завдання.

2. ІС управління технічними процесами (ТП) служить для автоматизації функцій виробничого персоналу по контролю і управлінню технічним процесом. У таких системах зазвичай передбачаються наявність розвинених засобів вимірювання параметрів технологічних процесів, процедур контролю допустимості значень, параметрів і регулювання технологічних процесів.

3. ІС автоматизованого проектування (САПР) призначені для автоматизації функцій інженерів-проектувальників, конструкторів, архітекторів, дизайнерів при створенні техніки. Основними функціями подібної системи є інженерні

розрахунки, створення графічної документації, створення проектної документації, моделювання об’єктів.

4. Інтегровані (корпоративні) ІС використовуються для автоматизації всіх функцій фірми, і охоплює весь цикл робіт від планування діяльності до збуту продукції. Вони включають ряд модулів (підсистем), що працюють в єдиному інформаційному просторі і виконують функції підтримки відповідних напрямів діяльності.

Основні поняття лекції:

1. *Інформаційна система* – це сукупність засобів і методів, що використовуються для обробки даних.
2. *Інформація* – це сукупність знань про фактичні дані і залежності між ними.
3. Поняття «*система*» охоплює комплекс взаємозв’язаних елементів, що діють як єдине ціле.

Тема 1.1.3 Життєвий цикл програмного забезпечення

(2 години)

Мета: Ознайомити студентів з поняттям життєвого циклу ПЗ, а також з поняттям структури ЖЦ.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Життєвий цикл ПЗ.
2. Структура життєвого циклу.
3. Процеси, що складають структуру ЖЦ.
4. Типи моделей технологічного процесу.

5. Моделі процесу розробки.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

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

Контрольні питання:

1. З чого складається структура ЖЦ?
2. В чому різниця між верифікацією, тестуванням та перевіркою?
3. Що в себе включає управлянні конфігукацією?

Методологія проектування ІС описує процес створення і супроводу систем у вигляді *життєвого циклу* (ЖЦ) ІС представляючи його як деяку послідовність стадій і виконуваних на них процесів. Для кожного етапу визначається склад і послідовність виконуваних робіт, ролі і відповідальність учасників і т.д. такий формальний опис ЖЦ ІС дозволяє спланувати і організувати процес колективної розробки і забезпечити управління цим процесом.

ЖЦ ІС можна представити як ряд подій, що відбуваються з системою в процесі її створення і використання.

Модель ЖЦ відображає різні стани системи, починаючи з моменту виникнення необхідності в даній ІС і закінчуючи моментом її повного виходу з вживання.

Модель ЖЦ - це структура, що містить процеси, дії і завдання, які здійснюються в ході розробки, функціонування і супроводу програмного продукту в продовж всього життя системи, від визначення вимог до завершення її використання.

Структура ЖЦ програмного забезпечення (ПЗ) за стандартом ISO/IEC 122071 базується на трьох групах процесів:

1. основні процеси ЖЦ ПЗ (придбання, постачання, розробка, експлуатація, супровід)

2. допоміжні процеси, що забезпечують виконання основних процесів (документування, управління конфігурацією, забезпечення якості, верифікація, атестація, оцінка, аудит, вирішення проблем)
3. організаційні процеси (управління проектами, створення інфраструктури проекту, визначення, оцінка і поліпшення самого ЖЦ, навчання)

Розробка включає всі роботи із створення ПЗ і його компонентів відповідно до заданих вимог, включаючи оформлення проектної і експлуатаційної документації, підготовку матеріалів, необхідних для перевірки працездатності і відповідної якості програмних продуктів, матеріалів необхідних для організації навчання персоналу і т.д. Розробка ПЗ включає, як правило, аналіз, проектування і реалізацію (програмування).

Експлуатація включає роботи по впровадженню компонентів ПЗ в експлуатацію, зокрема конфігурацію БД і робочих місць користувачів, забезпечення експлуатаційною документацією, проведення навчання персоналу і т. д., і безпосередньо експлуатацію, зокрема локалізацію проблем і усунення причин їх виникнення, моделювання ПЗ в рамках встановленого регламенту, підготовку пропозицій по вдосконаленню, розвитку і модернізації системи.

Управління проектом пов'язане з питаннями планування і організації робіт, створення колективів розробників і контролю за термінами і якістю виконання робіт.

Технологічне і організаційне забезпечення проекту включає вибір методів і інструментальних засобів для реалізації проекту, визначення методів опису проміжних станів розробки, розробку методів і засобів випробувань ПЗ, навчання персоналу і т.п. Забезпечення якості проекту пов'язане з проблемами верифікації, перевірки і тестування ПЗ. ВЕРИФІКАЦІЯ - це процес визначення того, чи відповідає поточний стан розробки, досягнутий на даному етапі, вимогам цього етапу. ПЕРЕВІРКА дозволяє оцінити відповідність параметрів розробки з початковими вимогами. Перевірка частково співпадає з ТЕСТУВАННЯМ, яке пов'язане з ідентифікацією відмінностей між результатами, що діють і очікуваними, і оцінкою відповідності характеристик початковим вимогам.

В процесі *реалізації* проекту важливе місце займають питання ідентифікації, опису і контролю конфігурації окремих компонентів і всієї системи в цілому.

Управління конфігурацією є одним з допоміжних процесів, що підтримують основні процеси ЖЦ ПЗ, перш за все процеси розробки і супроводу ПЗ. При створенні проектів складних ІС, що складаються з багатьох компонентів, кожен з яких може мати різновиди або версії, виникає проблема обліку їх зв'язків і функцій, створення уніфікованої структури і забезпечення розвитку всієї системи. Управління конфігурацією дозволяє організувати, систематично враховувати і контролювати внесення змін до ПЗ на всіх стадіях ЖЦ.

Кожен процес характеризується певними завданнями і методами їх рішення, початковими даними, отриманими на попередньому етапі, і результатами. Результатами аналізу, зокрема, є функціональні моделі, інформаційні моделі і відповідні ним діаграми. ЖЦ ПЗ носить ітераційний характер: результати чергового етапу часто викликають зміни в проектних рішеннях, вироблених на попередніх етапах.

Створення ПЗ - це сукупність процесів, що приводять до створення програмного продукту. Ці процеси ґрунтуються головним чином на технологіях інженерії програмного забезпечення. Існує чотири фундаментальних процеси, які властиві будь-якому проекту створення ПЗ.

1. *Розробка специфікації вимог на програмне забезпечення.* Вимоги визначають функціональні характеристики системи й обов'язкові для виконання.
2. *Створення програмного забезпечення.* Розробка й створення ПЗ згідно специфікації на нього.
3. *Атестація програмного забезпечення.* Створене ПЗ повинне пройти атестацію для підтвердження відповідності вимогам замовника.
4. *Удосконалювання (модернізація) програмного забезпечення*. ПЗ повинне бути таким, щоб його можна було модернізувати відповідно до змінених вимог споживача.

При виконанні різноманітних програмних проектів ці процеси можуть бути організовані різними способами й описані на різних рівнях деталізації. Тривалість реалізації цих процесів також далеко не завжди однакова. І взагалі, різні організації, що займаються виробництвом ПЗ, найчастіше використовують різні процеси для створення програмних продуктів навіть одного типу. З іншого боку,

певні процеси більше підходять для створення програмних продуктів одного типу й менш - для іншого типу програмних додатків. Якщо використовувати невідповідний процес, це може привести до зниження якості й функціональності розроблювального програмного продукту.

Під *моделлю життєвого циклу* розуміється структура, що визначає послідовність виконання й взаємозв'язки процесів, дій і завдань, що виконуються протягом ЖЦ. Модель ЖЦ залежить від специфіки ІС і специфіки умов, у яких остання створюється й функціонує.

Моделі можуть відображати процеси, які є частиною технологічного процесу створення ПЗ, компоненти програмних продуктів і дії людей, що беруть участь у створенні ПЗ. Опишемо коротко типи моделей технологічного процесу створення програмного забезпечення.

1. *Модель послідовності робіт.* Показує послідовність етапів, що виконуються у процесі створення ПЗ, включаючи початок і завершення кожного етапу, а також залежність між виконанням етапів. Етапи в цій моделі відповідають певним роботам, що виконуються розробниками ПЗ.
2. *Модель потоків даних і процесів.* В ній процес створення ПЗ представляється у вигляді безлічі активностей (процесів), у ході реалізації яких виконуються перетворення певних даних. Наприклад, на вхід активності (процесу) створення специфікації ПЗ надходять певні дані, на виході цієї активності одержують дані, які надходять на вхід активності, що відповідає проектуванню ПЗ, і т.д. Активність у такій моделі часто є процесом більше низького порядку, чим етапи робіт у моделі попереднього типу. Перетворення даних при реалізації активностей можуть виконувати як розробники ПЗ, так і комп'ютери.
3. *Рольова модель.* Модель цього типу представляє ролі людей, включених у процес створення ПЗ, і дії, що виконуються ними в цих ролях.

Існує також велика кількість різноманітних моделей процесу розробки програмного забезпечення (тобто підходів до процесу розробки).

1. *Каскадний підхід.* Весь процес створення ПЗ розбивається на окремі етапи: формування вимог до ПЗ, проектування й розробка програмного продукту,

його тестування й т.д. Перехід до наступного етапу здійснюється тільки після того, як повністю завершуються роботи на попередньому.Кожний етап завершується випуском повного комплекту документації, достатньої для того, щоб розробка могла бути продовжена іншою командою розробників.

Позитивні сторони застосування каскадного підходу полягають у наступному:

- На кожному етапі формується закінчений набір проектної документації, що відповідає категоріям повноти й погодженості;
- Етапи робіт, що виконуються в логічній послідовності дозволяють планувати строки завершення всіх робіт і відповідні витрати.

Каскадна схема розробки ПЗ

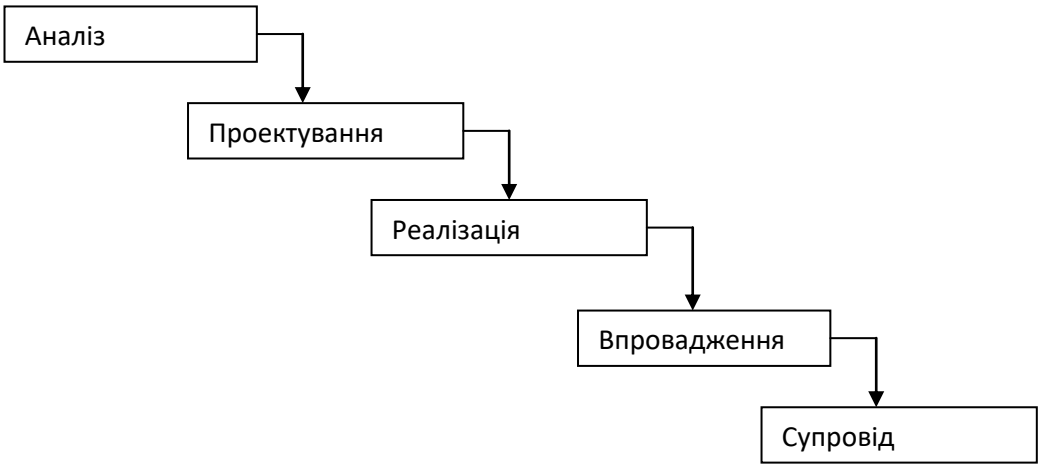


Рисунок 1 – Каскадна модель життєвого циклу ПЗ

Каскадний підхід добре зарекомендував себе при побудові ІС, для яких на самому початку розробки можна досить точно й повно сформулювати всі вимоги, для того, щоб надати розробникам волю реалізувати їх якнайкраще з технічної точки зору. У цю категорію попадають складні розрахункові системи, системи реального часу й інші подібні завдання. Однак у процесі використання цього підходу виявився ряд його недоліків, викликаних, насамперед тим, що реальний процес створення ПЗ ніколи повністю не вкладається в таку жорстку схему. У процесі створення ПЗ постійно виникає потреба в поверненні до попередніх етапів і уточненні або перегляді раніше ухвалених рішень. У результаті реальний процес створення ПЗ приймав наступний вид.

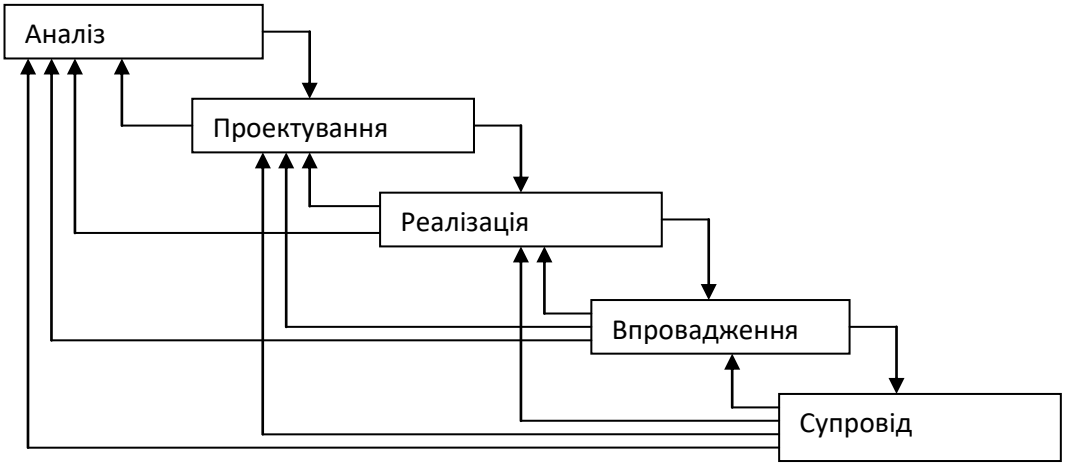


Рисунок 2 – Реальний процес розробки ПЗ за каскажною моделлю

Основним недоліком каскадного підходу є істотне запізнення з одержанням результатів. Узгодження результатів з користувачами відбувається тільки в точках, запланованих після завершення кожного етапу робіт, вимоги до ІС «заморожені» у вигляді технічного завдання на увесь час її створення. Таким чином, користувачі можуть внести свої зауваження тільки після того, як робота над системою буде повністю завершена. У випадку неточного викладу вимог або їхньої зміни протягом тривалого часу створення ПЗ, користувачі одержують систему не задовольняючу їхнім потребам. Моделі (як функціональні так і інформаційні) можуть застаріти одночасно з їхнім затвердженням.

2. *Еволюційний підхід (спіральна модель).* Тут послідовно перемежовуються етапи формування вимог, розробки ПЗ і його атестації. Первісна програмна система швидко розробляється на основі деяких абстрактних (загальних) вимог. Потім вони уточнюються й деталізуються відповідно до вимог замовника. Далі система допрацьовується й атестується відповідно до нових уточнених вимог. Така послідовність дій може повторитися кілька разів.Спіральна модель ЖЦ робить упор на початкові етапи ЖЦ: аналіз і проектування. На цих етапах реалізуємість технічних рішень перевіряється шляхом створення прототипів. Кожний виток спирали відповідає створенню фрагмента або версії ПЗ, на ньому уточнюються мети й характеристики проекту, визначається його якість і плануються роботи наступного витка спирали. Таким чином, заглиблюються й послідовно

конкретизуються деталі проекту й у результаті вибирається обґрунтований варіант, що доводить до реалізації.

Розробка ітераціями відображає об'єктивно існуючий спіральний цикл створення системи. Повне завершення робіт на кожному етапі дозволяє переходити на наступний етап не чекаючи повного завершення робіт на поточному. При ітеративному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне ж завдання якнайшвидше показати користувачам системи працездатний продукт, тим самим активізуючи процес уточнення й доповнення вимог.

Основна проблема спірального циклу - визначення моменту переходу на наступний етап. Для її рішення необхідно ввести тимчасові обмеження на кожний з етапів ЖЦ. Перехід здійснюється відповідно до плану, навіть якщо не вся запланована робота закінчена. План складається на основі статичних даних отриманих у попередніх проектах і особистому досвіді розроблювачів.

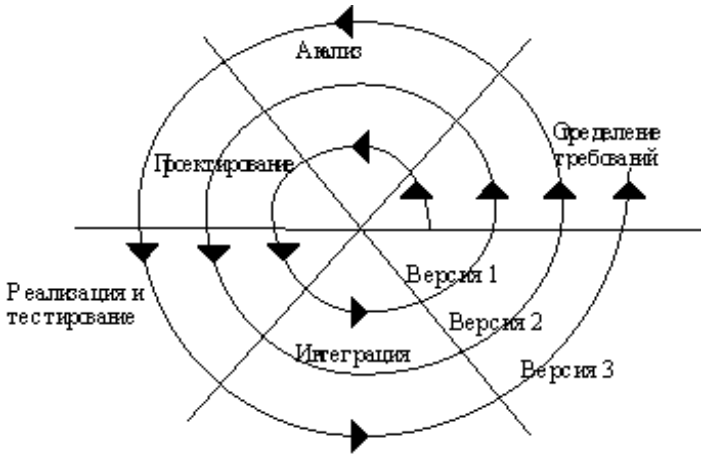


Рисунок 3 – Спіральна модель ЖЦ

3. *Формальні перетворення.* Підхід заснований на розробці формальної математичної специфікації програмної системи й перетворенні цієї специфікації за допомогою спеціальних математичних методів у програмі. Таке перетворення задовольняє умові "збереження коректності". Це означає, що отримана програма буде в точності відповідати розробленій специфікації.

4. *Складання програмного продукту з раніше створених компонентів.* Передбачається, що окремі складові частини програмної системи вже існують,

тобто створені раніше. У цьому випадку технологічний процес створення ПЗ основну увагу приділяє інтеграції окремих компонентів у загальне ціле, а не створенню цих компонентів.

Основні поняття лекції:

1. Процес створення і супроводу систем у вигляді *життєвого циклу* (ЖЦ) ІС - це деяка послідовність стадій і виконуваних на них процесів.
2. *Модель ЖЦ* - це структура, що містить процеси, дії і завдання, які здійснюються в ході розробки, функціонування і супроводу програмного продукту в перебігу всього життя системи, від визначення вимог до завершення її використання.
3. *Верифікація* - це процес визначення того, чи відповідає поточний стан розробки, досягнутий на даному етапі, вимогам цього етапу.
4. *Перевірка* дозволяє оцінити відповідність параметрів розробки з початковими вимогами.
5. *Тестування* пов'язане з ідентифікацією відмінностей між результатами, що діють і очікуваними, і оцінкою відповідності характеристик на початкові вимоги.

Блок змістових модулів №2 Основи моделювання програмного забезпечення

Мета розділу – дати основні поняття про CASE-засоби та структурний підхід до проектування. Дати основні поняття о вимогах, пропонованих до програмних систем, і показати різні способи представлений цих вимог

Вивчивши цей розділ студенти повинні:

- Мати основні поняття про CASE-засоби та CASE-технології
- Володіти основними поняттями структурного підходу до проектування
- Розрізняти ознаки якісного програмного забезпечення
- Вміти створювати ієрархію SADT-діаграм з використанням різних видів зв'язків
- мати поняття про концепції користувальницьких системних вимог і знати, чому для запису цих вимог використовуються різні способи;
- розуміти розходження між функціональними та нефункціональними вимогами;
- знати стандарти документування вимог до програмного забезпечення.

Змістовий модуль 2.1 Структурний підхід до проектування ПЗ

Тема 2.1.1 Суть структурного підходу до проектування

(2 години)

Мета: Ознайомити студентів із суттю структурного підходу до проектування, а також розглянути та навчитися відрізняти базові та основні принципи структурного підходу.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.
II. Повідомлення теми та мети заняття;
III. Виклад нового матеріалу:

План:

1. Поняття структурного підходу до проектування.
2. Суть структурного підходу
3. Принципи використання структурного підходу.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Навести приклади використання базових та основних принципів проектування.
3. Відповісти на контрольні питання.

Контрольні питання:

1. Які існують основні принципи інженерії програмного забезпечення?
2. Назвіть основні види діаграм в структурному аналізі.
3. Чому неможливо керуватися тільки базовими принципами?

Суть структурного підходу до розробки ІС полягає в її декомпозиції (розбитті) на функції, що автоматизуються: система розбивається на функціональні підсистеми, які у свою чергу діляться на підфункції, що підрозділяються на задачі і т. д. Процес деталізації продовжується аж до конкретних процедур. При цьому система, що автоматизується, представляє цілісне уявлення, в якому всі складові компоненти взіємозв'язані. При розробці системи «знизу-вгору» від окремих задач до всієї системи цілісність втрачається, виникають проблеми при інформаційній стиковці окремих компонентів.

Всі найпоширініші методології структурного підходу базуються на ряді загальних принципів.

До базових відносяться:

1. *Принцип «розділяй та володій»* - принцип рішення складних проблем шляхом розбиття їх на безліч менших незалежних задач, легких для розуміння і рішення.

2. *Принцип ієрархічного управління* – принцип заключається в організації складових частин проблеми в ієрархічні деревовидні структури з додаванням нових деталей на кожному рівні.

Виділення двох базових принципів інженерії ПЗ не означає, що решта принципів є другорядними, ігнорування буть якого з них може привести до непередбачуваних наслідків (в тому числі і до невдачі всього проекту).

Основні принципи:

1. *Принцип абстрагування* – полягає у виділені істотних, з деяких позицій, аспектів системи і відвернення від неістотних з метою представлення проблеми в простому загальному вигляді.
2. *Принцип формалізації* – полягає в необхідності строгого методичного підходу до рішення проблеми.
3. *Принцип хования* – полягає в необхідності ховати неістотну на конкретному етапі інформацію, кожна частина системи має тільки необхідну їй інформацію.
4. *Принцип концептуальної спільності* – полягає в проходжені єдиної філософії на всіх етапах ЖЦ (структурний аналіз – структурне проектування – структурне програмування – структурне тестування; об’єктно-орієнтований аналіз - об’єктно-орієнтоване проектування - об’єктно-орієнтоване програмування - об’єктно-орієнтоване тестування).
5. *Принцип повноти* – полягає в контролі на відсутність зайвих елементів.
6. *Принцип несуперечності* – полягає в обґрунтованості і узгодженості елементів.
7. *Принцип логічної незалежності* – полягає в концепції уваги на логічному проектуванні для забезпечення незалежності від фізичного проектування.
8. *Принцип незалежності даних* – полягає в тому, що моделі даних повинні бути проаналізовані і спроектовані незалежно від процесів їх логічної обробки, а також від їх логічної структури і розподілу.

9. *Принцип структуризації даних* – полягає в тому, що дані повинні бути структуровані і ієрархічно організовані.
10. *Принцип доступу кінцевого користувача* – полягає в тому, що користувач повинен мати засоби доступу до БД, які він може використовувати безпосередньо (без програмування).

Дотримання вказаних принципів необхідне при організації робіт на початкових етапах ЖЦ незалежно від типу розробляємого програмного забезпечення і методологій, що при цьому використовуються.Користуючись всіма принципами в комплексі, можна на більш раніх стадіях розробки зрозуміти, що представлятиме з себе майбутня система, знайти промахи і недоліки, що у свою чергу полегшить роботи на подальших етапах ЖЦ і знизить вартість розробки.

У структурному аналізі використовується в основному дві групи засобів, що ілюструють функції, які виконуються системою, а також ілюструють відносини між даними в системі. Кожній групі засобів відповідають певні види моделей (діаграм), найпоширеніші з них це:

1. SADT (Structured Analysis and Design Technique) – моделі і відповідні функціональні діаграми.
2. DFD (Data Fiow Diagram) – діаграми потоків даних
3. ERD (Entity-Relationship Diagram) – діаграми «сутність – зв'язок».

На стадії проектування ІС моделі розширяються, уточнюються і доповнюються діаграмами, що відображають структуру програмного забезпечення: архітектуру ПЗ, структурні схеми і діаграми екраних форм.

Перелічені моделі в сукупності дають повний опис ІС, не залежно від того, чи є вона існуючою обо тільки розробляється. Склад діаграм у кожному конкретному випадку залежить від необхідної повноти опису системи.

Тема 2.1.2 Вимоги до програмного забезпечення

(2 години)

Мета: Ознайомити студентів з поняттям вимоги до ПЗ. Розглянути різні способи виявлення вимог до програмного забезпечення. Розглянути способи тестування вимог.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Вимоги до ПЗ
2. Джерела та шляхи виявлення вимог
3. Аналіз вимог. Параметри тестування документації
4. Основні принципи тестування вимог
5. Властивості якісних вимог
6. Техніки тестування вимог

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

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

Контрольні питання:

1. Що таке вимоги до програмного забезпечення?
2. Назвіть основні способи виявлення вимог?
3. Як документуються вимоги до ПЗ?
4. Як тестуються вимоги?

Вимога — опис того, які функції та з дотриманням яких умов має виконувати програма в процесі вирішення корисного для користувача завдання.

Вимоги є відправною точкою для визначення того, що проектна команда буде проектувати, реалізовувати та тестувати. Елементарна логіка

каже нам, якщо у вимогах щось «не те», то й реалізовано буде «не те», тобто колосальна робота безлічі людей буде виконана даремно.

Описуючи важливість вимог, наголошується, що вони:

- **Дозволяють зрозуміти**, що та з дотриманням яких умов система повинна робити.
- Надають можливість **оцінити масштаб** змін та керувати змінами.
- Є основою формування **плану проекту** (зокрема плану тестування).
- Допомагають запобігати чи **вирішувати конфліктні ситуації**.
- **Спрощують** розміщення пріоритетів у наборі завдань.
- Дозволяють **об'єктивно оцінити** рівень прогресу у створенні проекту.

У загальному випадку **документацію** можна розділити на **дві великі групи** залежно від часу та місця її використання.

Продуктна документація (product documentation, development documentation) використовується проектною командою під час розробки та підтримки продукту. Вона включає:

- **План проекту** (project management) у тому числі **тестовий план** (test).
- **Вимоги** до програмного продукту (product requirements document) та **функціональні специфікації** (functional specifications document, FSD; software requirements specification, SRS).
- **Архітектуру та дизайн** (architecture and design).
- **Тест-кейси** та набори тест-кейсів (test cases, test suites).
- **Технічні специфікації** (technical specifications), такі як схеми бази даних, описи алгоритмів, інтерфейсів тощо.

Проєктна документація (project documentation) включає як продуктну документацію, та і деякі додаткові види документації і використовується як на стадії розробки, та й на більш ранніх і пізніх стадіях (наприклад, на стадії впровадження й експлуатації).

- **Користувальницьку та супровідну документацію** (user and accompanying documentation), таку як вбудована допомога, посібник зі встановлення та використання, ліцензійні угоди тощо.

- **Маркетингову документацію** (market requirements document, MRD), яку представники розробника чи замовника використовують як у початкових етапах (для уточнення суті та концепції проекту), і на фінальних етапах розвитку проекту (для просування продукту над ринком)

Джерела та шляхи виявлення вимог:

1. Інтерв'ю, опитування, анкетування
2. Аналіз нормативної документації
3. Мозговий штурм, семінар
4. Спостереження за виробничою діяльністю, «фотографування» робочого дня
5. Аналіз моделей діяльності
6. Аналіз конкурентних продуктів
7. Аналіз статистики використання попередніх версій системи
8. Нарада
9. Use case

Анкетування

Даний спосіб має на увазі під собою складання листа-опитувальника (анкети, брифа), який може містити відкриті (вимагають від опитуваного сформулювати його відповідь) та закриті (вимагають від опитуваного вибрати відповідь із запропонованих варіантів) питання.

Анкетування використовується для того, щоб підтвердити або деталізувати відомі раніше вимоги, вибрати параметри для рішень.

Найвідомішим прикладом анкетування може бути "**Бриф на розробку сайту**" - анкета, яка містить список основних вимог та інформацію про майбутній сайт.

Переваги:

- **Висока швидкість** одержання результатів.
- Порівняно **невеликі матеріальні витрати**.

Недоліки:

- Цей метод **не підходить для виявлення неявних вимог**.
- При складанні опитувальника **фізично неможливо врахувати всі необхідні питання**.

Аналіз нормативної документації

Ця методика може бути використана за **наявності в організації документації,** яка може допомогти у визначенні потреб замовника. Приклади документації включають: **регламенти, описи процесів, структура організації, специфікації продукту, різні процедури, стандарти та інструкції, шаблони документів і т. д**.

Виявлені вимоги є основою для подальшого аналізу та мають бути деталізовані. Ця методика застосовна, наприклад, при автоматизації усталених в організації регламентованих бізнес процесів.

Плюс:

- Швидке отримання інформації.

Мінус:

- Даний спосіб не застосовується за наявності в компанії тільки базових документів (або за їх повної відсутності) або якщо в компанії замовника не підтримується актуальність документації.

Мозковий штурм — найчастіше використовуваний метод отримання вимог, пов'язані з новими чи погано вивченими напрямами діяльності організації замовника чи функціями системи.

Він дозволяє **зібрати безліч ідей** від різних зацікавлених осіб **(стейкхолдерів)** у найкоротші терміни та практично безкоштовно.

Під час мозкового штурму учасники «накидають» **будь-які ідеї** щодо вирішення даної проблеми.

З допомогою цієї методики можна опрацювати кілька різних варіантів вирішення заданої проблеми, і навіть вирішити конфлікти вимог.

Плюси:

- Дозволяє **генерувати безліч (у тому числі і нестандартних) варіантів рішень,** а також дає можливість учасникам розвивати ідеї один одного.

Мінуси:

- Учасники мозкового штурму мають бути **мотивовані на** ідеї.
- Важко застосуємо у розподілених командах.

Нарада:

Нарада — зустріч, орієнтована на обговорення конкретних питань, визначених та озвучених учасникам заздалегідь. На такі зустрічі залучаються люди, які дотримуються різних точок зору щодо поточної проблеми та можуть допомогти описати вимоги, ґрунтуючись на поглядах із різних сторін. У процесі наради **уточнюється загальний перелік вимог, виявляються приховані вимоги та вирішуються конфлікти вимог.**

Наради є одним із ключових практик у Agile, т.к. в них беруть участь усі зацікавлені у розвитку проекту та вирішенні проблеми сторони.

Плюси:

- Дозволяє **розвинути та деталізувати вимоги, визначити пріоритети.**

Недоліки:

- Консенсусу необов'язково буде досягнуто.

Use case

Use cases або **варіанти використання** дозволяють зібрати та сформувати **функціональні вимоги від імені учасників.** Діаграми варіантів використання визначають **межі рішення та показують зв'язки із зовнішніми системами та учасниками.** Метод дозволяє деталізувати вимоги з погляду користувачів, а також допомагає уточнити та систематизувати функціонал, який потрібно реалізувати.

Плюси:

- Дозволяє опрацювати всі варіанти розвитку сценарію (основний та альтернативні сценарії).

Мінуси:

- Метод не застосовується для збору нефункціональних вимог.

Аналіз вимог. Параметри тестування документації

1. Чіткість та ясність

Розпочати тестування вимог можна з поверхневого огляду документації. Це складно назвати саме тестуванням, але нерідко вже на даному етапі виявляється чимало недоліків. Почнемо зі звичайного сценарію. Ви почали читати вимоги і вже з перших рядків у Вас виникає безліч запитань до автора (наприклад, «Який очікуваний результат після натискання на цю кнопку?» або «Що буде, якщо я скасую замовлення?»). Це погано. Після прочитання документації не повинно бути запитань. **Зовсім. Вимоги – це як зведення законів для товару, а закони не допускають двозначність, «води» та неточності.** Документація повинна давати гранично ясну інформацію про те, як повинен працювати кожен окремий модуль та весь продукт у цілому. На жаль, після прочитання більшості вимог залишається низка питань.

Приклад.

У вимогах було записано: «**У полі Ім'я користувача можуть бути введені літери та цифри**». Розробник почав з'ясовувати у аналітика, які саме літери (кирилиця, латиниця чи арабські) та які цифри (цілі, дробові, римські) маються на увазі. Після уточнення вимог розробник реалізував функціонал згідно з коментарями аналітика. Завдання перейшло у тестування. Тестувальник не розумів, за якими критеріями перевіряти це поле, і теж почав розпитувати аналітика.

Наслідки:

- Витрачений час кількох членів команди.
- Розбіжність підсумкового та спочатку планованого функціоналів.

Як тестувати:

Якщо у Вас після прочитання вимог залишилися питання – необхідне доопрацювання.

Якщо розробники часто уточнюють деталі в чатах, це поганий знак. Подальше («глибше») дослідження вимагає набагато більших часових витрат.

2. Актуальність

Необхідність підтримання актуальності вимог видається очевидною.

Проте, на деяких проектах вимоги не оновлюються місяцями, або роками.

Це може бути пов'язано з тим, що в штаті немає аналітика, а виконуючий його обов'язки співробітника просто не вистачає часу.Трапляється й інше: **вимоги оновлюють лише за наявності дійсно значущих змін, при цьому різні «дрібниці» у вигляді зміни кнопок або текстів ігноруються**.

Приклад.

Вирішили змінити положення кнопок на сторінці авторизації. Аналітик не став правити документацію, а написав розробнику особисте повідомлення з проханням поправити розташування кнопок. Розробник вніс виправлення та закрив завдання. Під час чергового регресійного тестування тестувальник вирішив, що це дефект і завів на нього баг. Інший розробник повернув кнопки на попередні позиції згідно з документацією.

Наслідки:

- Час кількох членів команди витрачено марно.
- Підсумкова позиція кнопок не відповідає очікуваному результату.

Як тестувати:

- За наявності повідомлень у командному чаті потрібно переконатися, що оновлені вимоги задокументовані.
- Необхідно **порівняти дати оновлення технічного завдання та пояснювальної записки з датою останнього оновлення вимог**.

3. Логіка

Як випливає з назви, робота системи має бути логічною. Користувач не може змінити налаштування свого профілю або написати листа до того,

як пройде авторизацію в системі. Звучить, знову ж таки, елементарно, але в проектах з безліччю клієнтів або зі складною логікою такі помилки часто припускаються.

Приклад.

У мобільному додатку виникла потреба реалізувати функціонал електронного підпису документа. Користувачеві пропонувалося ввести свої дані, після чого вони автоматично підставлялися в шаблон документа. Додаток відкривав документ і пропонував його підписати. Якщо користувач розумів, що в документі є помилки, то виправити їх вже не міг: у нього була можливість тільки підписати цей документ. Закриття програми або його переустановка не допомагали – при вході користувача до облікового запису відразу відображався той самий документ на підпис.

Наслідки:

- Користувач у сказі.
- Подальша робота з акаунтом без звернення до техпідтримки неможлива.

Як тестувати:

- Намалювати **зразкову блок-схему** роботи системи відповідно до вимог та переконатися, що **в ній немає логічних прогалин**.
- Переконатись, що у вимогах **описано необхідний основний функціонал**.
- Переконатись, що **взаємодія між модулями системи викладена коректно**.

4. можливі сценарії

У документації мають бути докладно описані як очевидні, і неочевидні варіанти використання системи. До очевидних (позитивних) варіантів, наприклад, можна віднести введення коректної пари логін/пароль. До неочевидних (негативних) – введення некоректної пари логін/пароль або відсутність цих даних зовсім.

Приклад. Часто з погляду упущені такі моменти, як тексти помилок, поведінка системи при втраті зв'язку, а також обробка помилок, пов'язаних із сторонніми сервісами (наприклад, з оплатою). Наслідки: При втраті зв'язку система поводиться некоректно (відсутність помилок, зависання). Повідомлення про помилки не є очевидними. У найгіршому разі можливі репутаційні чи фінансові втрати. Як тестувати: Намалювати блок-схему окремого модуля системи, у межах якої позначити всі можливі умови та дії користувача. Переконатись, що у вимогах є опис кожного можливого випадку.

5. інтеграція

Має сенс виділити інтеграцію зі сторонніми сервісами, тому що тут доводиться виходити за рамки перевірки документації. Перед початком розробки аналітики, як правило, вивчають роботу сторонньої системи, а потім описують схему взаємодії цієї системи з продуктом, що розробляється. В даному випадку, ймовірність помилки дуже велика, оскільки помилитися можуть як аналітики, і представники стороннього сервісу, які консультували чи писали документацію.

Приклад. На проекті потрібно було реалізувати можливість авторизації через сторонній сервіс. Аналітик помилково вивчив застарілу документацію стороннього сервісу та описав свідомо неробочу схему взаємодії. Розробники розпочали роботу, відповідно до готової схеми, але постійно отримували помилки. Вони «допитували» аналітика, а той поспіхом дзвонив на техпідтримку стороннього сервісу та з'ясовував причини помилок. Наслідки: Затримка розробки функціоналу тиждень. Як тестувати: Необхідно вручну перевірити, що сторонній сервіс обробляє всі необхідні запити відповідно до описаної схеми. Перевірити, чи вказав аналітик коректно та в повному обсязі всю необхідну для розробки інформацію.

Основні принципи тестування вимог

Тестування вимог краще проводити до початку розробки. Для цього потрібно розрахувати необхідний час на перевірку та заморозити

документацію, що тестується, до закінчення перевірки. Проводити тестування вимог можуть як аналітики, і тестувальники. Однак, для досягнення кращого результату опис та перевірку вимог слід доручати різним людям. Виявлення дефектів документації нічим не відрізняється від виявлення дефектів по продукту: баги слід заносити в систему баг-трекінгу як завжди. У тому випадку, коли перевірка вимог проводиться паралельно з розробкою, вкрай бажано попередити команду розробки про знайдені дефекти (щоб вони могли вчасно виправити помилку). Рівень деталізації вимог (як і глибина тестування) залежить від рівня проекту. Немає сенсу перевіряти час реакцію кнопку в проекті, який тільки запустився (якщо це, звичайно, не відноситься до ключового функціоналу).

Властивості якісних вимог

1. Атомарність, поодинокість (atomicity). Вимога є атомарною, якщо її не можна розбити на окремі вимоги без втрати завершеності і вона визначає одну і одну ситуацію.

2. Несуперечність, послідовність (consistency). Вимога не повинна містити внутрішніх протиріч та протиріч іншим вимогам та документам.

3. Недвозначність (unambiguousness, clearness). Вимога має бути описана без використання жаргону, неочевидних абревіатур та розпливчастих формулювань, а також повинна допускати лише однозначне об'єктивне розуміння та бути атомарною у плані неможливості різного трактування поєднання окремих фраз.

4. Обов'язковість, потреба (obligatoriness) актуальність (up-to-date). Якщо вимога не є обов'язковою до реалізації, то вона має бути просто виключена з набору вимог. Якщо вимога потрібна, але не дуже важлива, для вказівки цього факту використовується вказівка пріоритету (див. проранжованість по ...). Також мають бути виключені (або перероблені) вимоги, що втратили актуальність.

5. Простежуваність (traceability). Простежуваність буває вертикальною (vertical traceability) та горизонтальною (horizontal traceability). Вертикальна

простежуваність дозволяє співвідносити між собою вимоги на різних рівнях вимог, горизонтальна – співвідносити вимоги з тест-планом, тест-кейсами, архітектурними рішеннями тощо. Для забезпечення простежуваності часто використовуються спеціальні інструменти управління вимогами (requirements management tool) та/або матриці простежуваності (traceability matrix).

6. Модифікованість (modifiability) - ця властивість характеризує внесення змін в окремі вимоги та набір вимог. Можна говорити про наявність модифікованості в тому випадку, якщо, при доопрацюванні вимог, шукану інформацію легко знайти, а її зміна не призводить до порушення інших описаних у цьому переліку властивостей.

7. Проранжованість за важливістю, стабільністю, терміновістю (ranked for importance, stability, priority). Важливість характеризує залежність успіху проекту від успіху реалізації вимоги. Стабільність характеризує ймовірність того, що в найближчому майбутньому до вимоги не буде внесено жодних змін. Терміновість визначає розподіл за часом зусиль проектної команди щодо реалізації тієї чи іншої вимоги.

8. Коректність (correctness) та перевіряємість (verifiability). Фактично, ці властивості випливають із дотримання всіх перелічених вище (або, можна сказати, вони не виконуються, якщо порушено хоча б одне з вищеперелічених). На додачу, можна зазначити, що проверяемость передбачає можливість створення об'єктивного тест-кейса (тест-кейсів), що однозначно показує, що вимога реалізована вірно і поведінка програми точно відповідає вимогам.

Техніки тестування вимог

Взаємний перегляд (peer review) або рецензування

є однією з найбільш активно використовуваних технік тестування вимог і може бути представлений в одній із трьох наступних форм (у міру наростання його складності та ціни):

Швидкий (беглий) перегляд (walkthrough)

може виражатися як у показі автором своєї роботи колегам з метою створення загального розуміння та отримання зворотного зв'язку, так і в простому обміні результатами роботи між двома та більше авторами для того, щоб колега висловив свої питання та зауваження. Це найшвидший, дешевий вид перегляду, що часто використовується.

Технічний перегляд (technical review)

виконується групою спеціалістів. В ідеальній ситуації – кожен фахівець має представляти свою галузь знань. Тестований продукт не може вважатися досить якісним, поки що хоча б у одного переглядача залишаються зауваження.

Формальна інспекція (inspection)

являє собою структурований, систематизований і документований підхід до аналізу документації. Для його виконання залучається велика кількість спеціалістів. Саме виконання займає досить багато часу, тому цей варіант перегляду використовується досить рідко (як правило, при отриманні на супровід та доопрацювання проекту, створенням якого раніше займалася інша компанія)

Питання

Наступною очевидною технікою тестування та підвищення якості вимог є використання (повторне) техніки виявлення вимог, а також (як окремий вид діяльності) — постановка питань. Якщо хоч щось у вимогах викликає у вас нерозуміння чи підозру, то ставте запитання. Можна запитати представників замовника або звернутися до довідкової інформації. З багатьох питань можна звернутися до досвідченіших колег за умови, що вони мають відповідну інформацію, раніше отриману від замовника. Головне, щоб Ваше питання було сформульоване таким чином, щоб отримана відповідь дозволила покращити вимоги.

Блок змістових модулів №3 Технології розробки програмного забезпечення

Мета розділу - дати основні поняття методологій проектування та нотацій побудови ER-діаграм. Вивчивши цей розділ студенти повинні:

- мати поняття про те, методологія швидкої розробки додатків RAD та методологія SSADM;
- Розрізняти графічні нотації П. Чена та Баркера, а також вміти будувати модель середовища;
- Розрізняти життєві системи за різними показниками;
- Будувати моделі методології IDEF3.

Змістовий модуль 3.1 *Нотації розробки ER- діаграм*

Тема 3.1.1 Нотація Баркера

(2 години)

Мета: Ознайомитись з основними поняттями CASE-методу Баркера атрибут, зв'язок, сутність, а також розглянути додаткові конструкції методу.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.
II. Повідомлення теми та мети заняття;
III. Виклад нового матеріалу:

План:

1. Мета моделювання даних;
2. Етапи моделювання даних;
3. Додаткові конструкції методу.

IV. Узагальнення та систематизація знань;
V. Підведення підсумків занять;
VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Вивчити основні поняття лекції.

3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Які основні кроки моделювання при використання методології Баркера?
2. Що таке сутність?
3. Властивості сутності.
4. Що таке зв'язок?
5. Що таке атрибут?
6. Що таке можливий ключ сутності?
7. Які додаткові конструкції існують в даному методі?

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

Найпоширенішим засобом моделювання даних є діаграми «сутність-зв’язок» (ERD). З їх допомогою визначаються важливі для предметної області об’єкти (сутності), їх властивості (атрибути) і відносини між ними (зв’язки). ERD безпосередньо використовуються для проектування реляційних баз даних.

Нотація ERD буда вперше введена П. Ченом і одержала подальший розвиток в работах Баркера.

Перший крок моделювання – виділення сутностей

Сутність (Entity) – реальний або уявний об’єкт, що має істотне значення для даної предметної області, інформація про який підлягає зберіганню.

<ім’я сутності>

Рисунок 1 – Зовнішня сутність

Кожна сутність повинна володіти унікальним ідентифікатором. Кожний екземпляр сутності повинен односзначно ідентифікуватися і відрізнятися від всіх інших екземплярів даного типу сутності. Кожна сутність повинна володіти деякими властивостями:

1. Кожна сутність повинна мати унікальне ім’я, і до одного і тогож імені повинна завжди застосуватися одна і таж інтерпритація. Одна і таж

інтерпритація не може застосовуватись до різних імен, якщо тільки вони не є псевдонімами.

2. Сутність володіє одним або декількома атрибутами, які або належать сутності, або успадковуються через зв’язок.
3. Сутність володіє одним або декількома атрибутами, які однозначно ідентифікують кожний екземпляр сутності.
4. Кожна сутність може володіти будь-якою кількістю зв’язків з іншими сутностями моделі.

Наприклад, для автосалону сутності, які можуть бути ідентифіковані з головним менеджером - це автомашини й продавці. Продавцеві важливі автомашини й пов'язані з їхнім продажем дані. Для адміністратора важливі покупці, автомашини, продавці й контракти. Виходячи із цього, виділяються 4 сутності (автомашина, продавець, покупець, контракт), які зображуються на діаграмі в такий спосіб

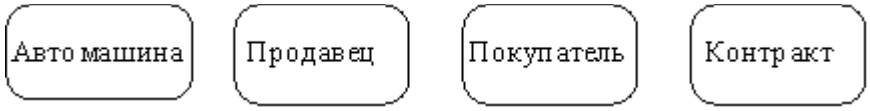


Рисунок 2 – Приклад сутностей

Другий крок моделювання – ідентифікація зв’язків

Зв’язок (Relationship) – пойменована асоціація між двома сутностями, значуща для даної предметної області. Зв’язок – це асоціація між сутностями, при якій, як правило, кожний екземпляр однієї сутності, що зветься батьківською, асоціюється з довільною кількістю (у тому числі нульовою) екземплярів іншої сутності, що зветься сутністю - нащадком. А кожний екземпляр сутності-нащадка асоційований в точності з одним екземпляром сутності-батька. Таким чином екземпляр сутності-нащадка може існувати тільки при наявності батьківської сутності.

Зв’язку може даватися ім’я. Ім’я кожного зв’язку між двома даними сутностями повинне бути унікальним, але імена зв’язків в моделі не зобов’язані бути унікальними.

Ступінь зв’язку і обов’язковість графічно зображуються таким чином:

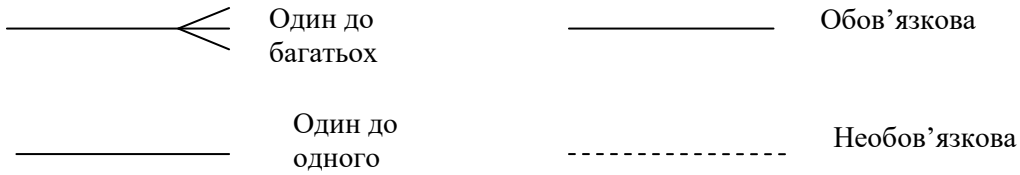


Рисунок 3 – Ступінь та обов’язковість звязків

Таким чином, 2 пропозиції, що описують зв'язок продавця з контрактом, графічно будуть виражені в такий спосіб (малюнок 4).

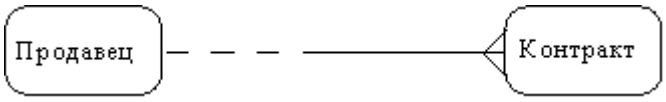


Рисунок 4 – Зв'язок продавця з контрактом

Описавши також зв'язку інших сутностей, одержимо наступну схему (малюнок 5).

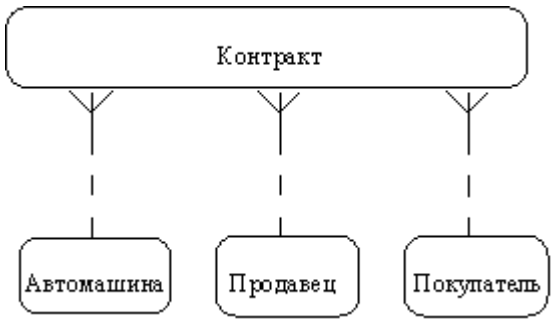


Рисунок 5 – Зв’язки між сутностями

Третій крок моделювання – ідентифікація атрибутів

Атрибут – будь-яка характеристика сутності, значуща для даної предметної області і призначена для кваліфікації, ідентифікації, класифікації, кількісної характеристики сутності. Атрибут представляє тип характеристик або властивостей, асоційованих з безліччю реальних або абстрактних об’єктів (людей, місць, подій, ідей тощо). Екземпляр атрибуту – це певна характеристика окремого елементу множини. Екземпляр атрибуту визначається типом характеристики і її значенням, званим значенням атрибуту. В ER-моделі атрибути асоціюються з конкретними сутностями. Таким чином, екземпляр сутності повинен володіти єдиним певним значенням для асоційованого атрибуту.

Атрибут може бути або обов’язковим, або необов’язковим. Обов’язковість означає, що атрибут не може приймати не визначених значень (nill values). Атрибут

може бути або описовим (тобто звичним дескриптором сутності), або входити до складу унікального ідентифікатора (первинного ключа).

Унікальний ідентифікатор – це атрибут або сукупність атрибутів і/чи зв'язків, призначена для унікальної ідентифікації кожного екземпляра даного типу сутності. У разі повної ідентифікації кожний екземпляр даного типу сутності повністю ідентифікується своїми власними ключовими атрибутами, інакше і його ідентифікації беруть участь атрибути іншої батьківської сутності.

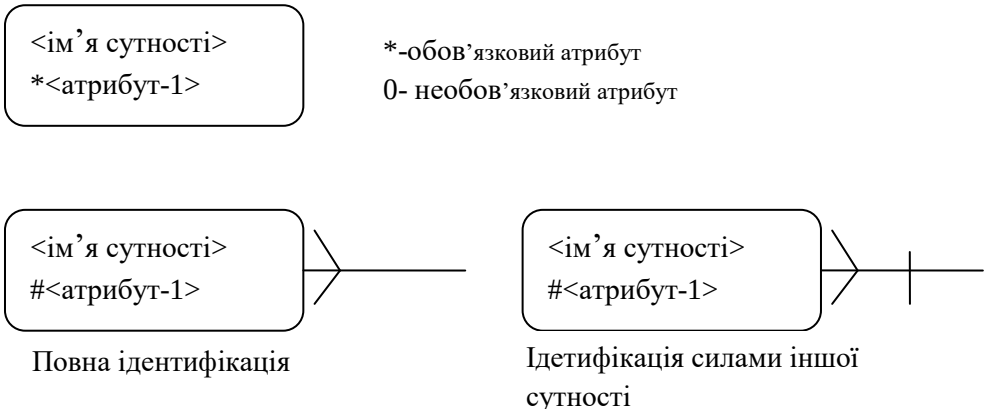


Рисунок 6 - Атрибути

Кожний атрибут ідентифікується унікальним ім'ям. Атрибути зображуються у вигляді списку імен усередині блоку асоційованої сутності, причому кожний атрибут займає окремий рядок. Атрибути, що визначають первинний ключ, розміщуються 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 з них позначається як первинний ключ, а інші як альтернативні ключі.

З урахуванням наявної інформації доповнимо побудовану раніше діаграму (малюнок 7).

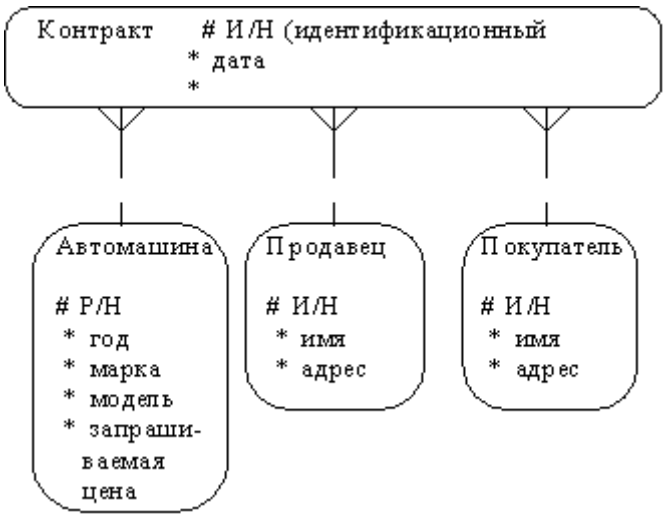


Рисунок 7 – Атрибути стностей

Крім перелічених основних конструкцій модель даних може містити рід додаткових.

Підтипи і супертипи: одна сутність є узагальнюючим поняттям для групи подібних сутностей.

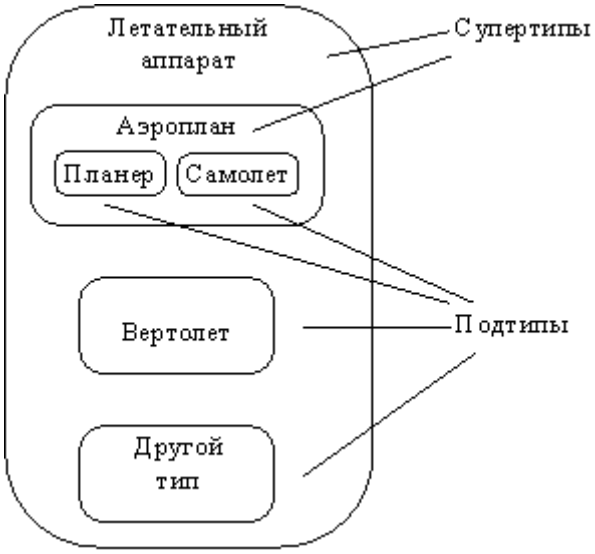


Рисунок 8 – Підтипи та супертипи

Зв’язки, що взаємно виключаються: кожний екземпляр сутності бере участь тільки в одному зв’язку з групи зв'язків, що взаємно виключають.

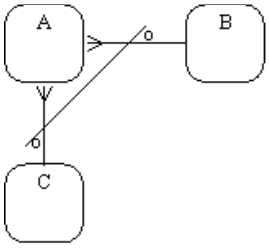


Рисунок 9 – зв'язки, що взаємно виключають

Рекурсивний зв'язок: сутність може бути зв'язана сама із собою.

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

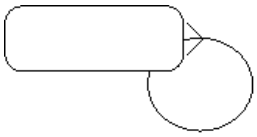


Рисунок 10 – Рекурсивний зв'язок

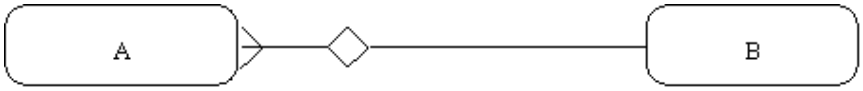


Рисунок 11 – Непереміщуємий зв'язок

Основні поняття лекції:

Сутність (Entity) – реальний або уявний об’єкт, що має істотне значення для даної предметної області, інформація про який підлягає зберіганню.

Зв’язок (Relationship) – пойменована асоціація між двома сутностями, значуща для даної предметної області. Зв’язок – це асоціація між сутностями, при якій, як правило, кожний екземпляр однієї сутності, що зветься батьківською, асоціюється з довільною кількістю (у тому числі нульовою) екземплярів іншої сутності, що зветься сутністю - нащадком.

Атрибут – будь-яка характеристика сутності, значуща для даної предметної області і призначена для кваліфікації, ідентифікації, класифікації, кількісної характеристики сутності. Атрибут представляє тип характеристик або властивостей, асоційованих з безліччю реальних або абстрактних об’єктів (людей, місць, подій, ідей тощо).

Екземпляр атрибуту – це певна характеристика окремого елементу множини.

Унікальний ідентифікатор – це атрибут або сукупність атрибутів і/чи зв’язків, призначена для унікальної ідентифікації кожного екземпляра даного типу сутності.

Можливий ключ сутності – це один або декілька атрибутів, чиї значення однозначно визначають кожний екземпляр сутності.

Тема 3.1.2 Нотація П. Чена

(2 години)

Мета: Ознайомитись з поняттями CASE-методу Баркера.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План:

1. Необов’язковий зв'язок;
2. Повний зв'язок:
   a. Обов’язків зв'язок;
   b. Слабкий зв'язок;
   c. Зв'язок «супертип - підтип»;
   d. Асоціативний зв'язок.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

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

Контрольні питання:

1. В чому різниця нотації Чена від нотації Баркера?
2. Які види повного зв’язку бувають?
3. Що таке асоціативний зв'язок та асоціативний об’єкт?
4. Що відбувається при слабкому зв’язку з ключем сильної сутності?
5. Що таке зв'язок «супертип - підтип»?

В Case-засобі Vantage Team Builder використовується один з варіантів нотації П. Чена. На Er-Діаграмах сутність позначається прямокутником, що містять

ім'я сутності, а зв'язок - ромбом, зв'язаним лінією з кожної із взаємодіючих сутностей. Числа над лініями означають ступінь зв'язку (рисунок 1).

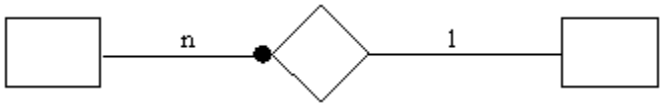


Рисунок 1 – Позначення сутностей та зв’язків

Зв'язки є багато направленими й можуть мати атрибути (за винятком ключових). Виділяють два види зв'язків:

- необов'язковий зв'язок (optional);
- повний зв'язок (total)

У **необов'язковому зв'язку** можуть брати участь не всі екземпляри сутності.

Рисунок 2 – Необов’язковий зв’язок

На відміну від необов'язкового зв'язку в **повному (total) зв'язку** беруть участь усі екземпляри хоча б однієї із сутностей. Це означає, що екземпляри такого зв'язку існують тільки за умови існування екземплярів іншої сутності. Повний зв'язок може мати один з 4- х видів: обов'язковий зв'язок, слабкий зв'язок, зв'язок "супертип-підтип" і асоціативний зв'язок.

Обов'язковий (mandatory) зв'язок описує зв'язок між "незалежної" і "залежної" сутностями. Усі екземпляри залежної ("обов'язкової") сутності можуть існувати тільки при наявності екземплярів незалежної ("необов'язкової") сутності, тобто екземпляр "обов'язкової" сутності може існувати тільки за умови існування певного екземпляра "необов'язкової" сутності.

У прикладі мається на увазі, що кожний автомобіль має принаймні одного водія, але не кожний службовець управляє машиною.

Рисунок 3 – Обов’язковий зв’язок

У **слабкому (Weak) зв'язку** існування однієї із сутностей, що належить деякій безлічі ("слабкої") залежить від існування певної сутності, що належить іншій безлічі ("сильної"), тобто екземпляр "слабкої" сутності може бути

ідентифікований тільки за допомогою екземпляра "сильної" сутності. Ключ "сильної" сутності є частиною складеного ключа "слабкої" сутності.

Слабкий зв'язок завжди є бінарним і має на увазі обов'язковий зв'язок для "слабкої" сутності. Сутність може бути "слабкою" в одному зв'язку й "сильною" в інший, але не може бути "слабкою" більш, ніж в одному зв'язку. Слабкий зв'язок може не мати атрибутів.

Приклад на малюнку: ключ (номер) рядка в документі може не бути унікальним і повинен бути доповнений ключем документа.

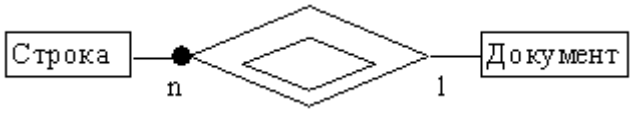


Рисунок 4 – Слабкий зв'язок

Зв'язок "супертип-підтип" - загальні характеристики (атрибути) типу визначаються в сутності-супертипі, сутність-підтип успадковує всі характеристики супертипу. Екземпляр підтипу існує тільки за умови існування певного екземпляра супертипу. Підтип не може мати ключа (він імпортує ключ із супертипу). Сутність, що є супертипом в одному зв'язку, може бути підтипом в іншому зв'язку. Зв'язок супертипу не може мати атрибутів.

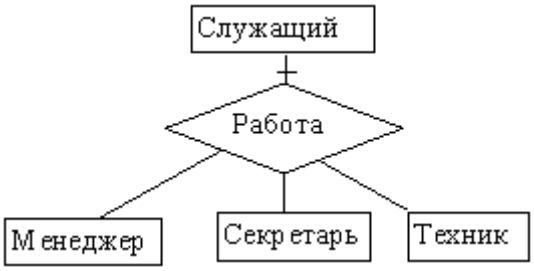


Рисунок 5 – Зв'язок супертип підтип

В **асоціативному зв'язку** кожний екземпляр зв'язки (асоціативний об'єкт) може існувати тільки за умови існування певних екземплярів кожної із взаємозалежних сутностей. **Асоціативний об'єкт** - об'єкт, що є одночасно сутністю й зв'язком. **Асоціативний зв'язок** - це зв'язок між декількома "незалежними" сутностями й однієї "залежною" сутністю. Зв'язок між незалежними сутностями має атрибути, які визначаються в залежній сутності. Таким чином, залежна сутність визначається в термінах атрибутів зв'язки між іншими сутностями.

У прикладі літак виконує посадку на злітну смугу в заданий час при певній швидкості й напрямку вітру. Оскільки ці характеристики застосовні тільки до

конкретної посадки, вони є атрибутами посадки, а не літака або злітної смуги. Пілот, що виконує посадку, зв'язаний набагато сильніше з конкретною посадкою, чому з літаком або злітною смугою.

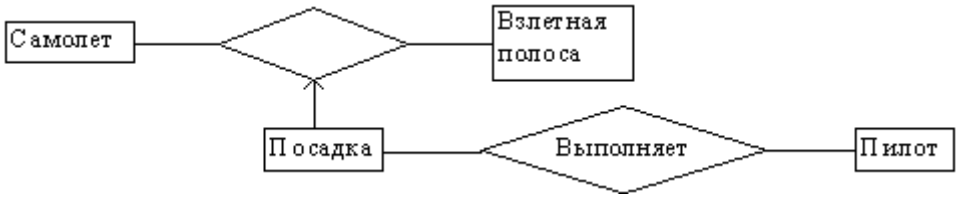


Рисунок 6 – Асоціативний зв'язок

Первинний ключ кожного типу сутності позначається зірочкою (*).

Er-Діаграма повинна підкорятися наступним правилам:

- кожна сутність, кожний атрибут і кожний зв'язок повинні мати ім'я (зв'язок супертипу або асоціативний зв'язок може не мати імені);
- ім'я сутності повинне бути унікально в рамках моделі даних;
- ім'я атрибута повинне бути унікально в рамках сутності;
- ім'я зв'язку повинне бути унікально, якщо для неї генерується таблиця БД;
- кожний атрибут повинен мати визначення типу даних;
- сутність у необов'язковому зв'язку повинна мати ключовий атрибут. Те ж саме ставиться до сильної сутності в слабкому зв'язку, супертипу у зв'язку " супертип-підтип" і необов'язкової сутності в обов'язковій (повної) зв'язки;
- підтип у зв'язку " супертип-підтип" не може мати ключовий атрибут;
- в асоціативній або слабкому зв'язку може бути тільки одна асоціативна (слабка) сутність;
- зв'язок не може бути одночасно обов'язкової, " супертип-підтип" або асоціативної.

Основні поняття лекції:

У *необов'язковому зв'язку* можуть брати участь не всі екземпляри сутності.

Обов'язковий (mandatory) зв'язок описує зв'язок між "незалежної" і "залежної" сутностями. Усі екземпляри залежної ("обов'язкової") сутності можуть існувати тільки при наявності екземплярів незалежної ("необов'язкової") сутності, тобто екземпляр "обов'язкової" сутності може існувати тільки за умови існування

певного екземпляра "необов'язкової" сутності.

У *слабкому (Weak) зв'язку* існування однієї із сутностей, що належить деякій безлічі ("слабкої") залежить від існування певної сутності, що належить іншій безлічі ("сильної"), тобто екземпляр "слабкої" сутності може бути ідентифікований тільки за допомогою екземпляра "сильної" сутності.

Асоціативний об'єкт - об'єкт, що є одночасно сутністю й зв'язком.

Асоціативний зв'язок - це зв'язок між декількома "незалежними" сутностями й однієї "залежною" сутністю.

Змістовий модуль 3.2 *Методології структурного підходу*

Тема 3.2.1 Системний аналіз

(2 години)

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

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План:

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

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. 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.

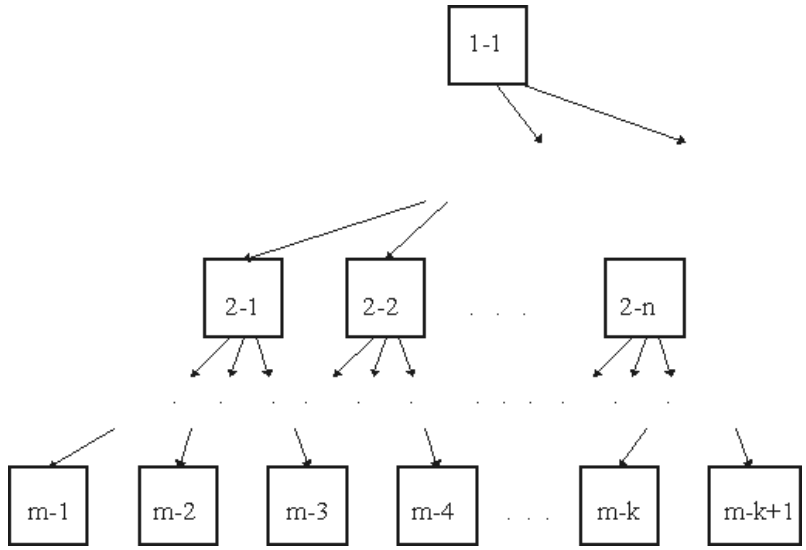
Приклад. Прикладами структур можуть бути структура звивин мозку, структура студентів на курсі, структура державного устрою, структура кристалічних ґрат речовини, структура мікросхеми й ін. Кристалічні ґрати

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.

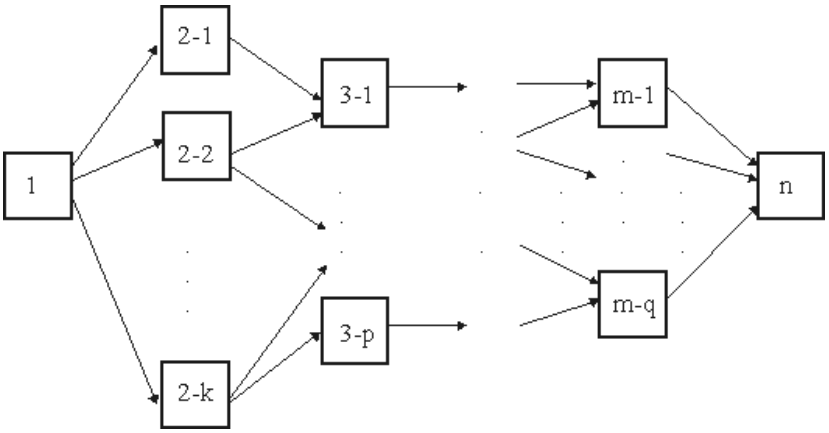
Структури систем бувають різного типу, різної топології (або ж просторової структури). Розглянемо основні топології структур (систем).

Лінійні структури:

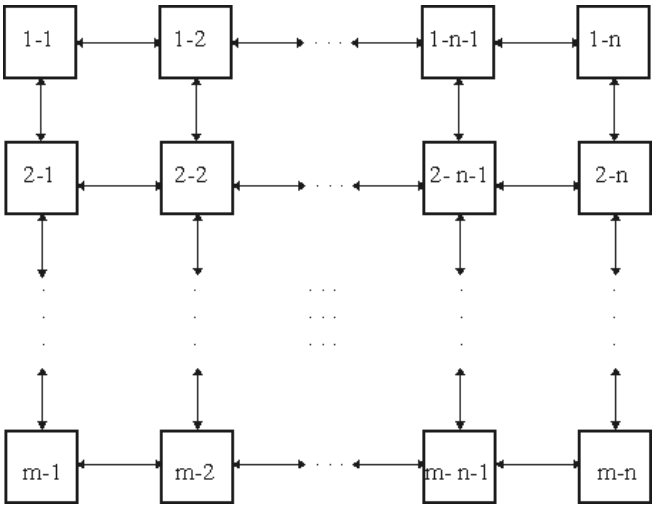
Ієрархічні древовидні структури:



Мережева структура:



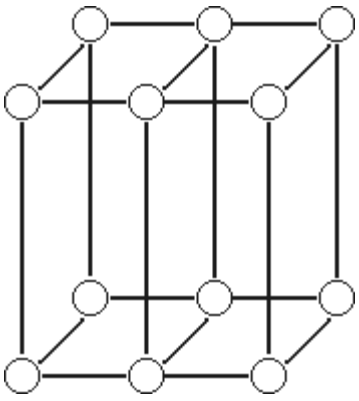
Матрична структура:



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).

Приклад. 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.

Основні поняття лекції:

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.

Мета - образ неіснуючого, але бажаного - з погляду завдання або розглянутої проблеми - стану середовища, тобто такого стану, який дозволяє вирішувати проблему при даних ресурсах.

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.

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

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

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).

Тема 2.2.2 Методологія функціонального моделювання SADT

(4 години)

Мета: Ознайоми студентів з основними поняттями SADT-моделі та порядком побудови даної моделі. Також розглянути концепції на яких базується SADT-модель.

Література:

1. І. Соммервіль «Инженерия программного обеспечения 6-е издание», М.: Вильялис, 2002, рус.;
2. За ред. Пономаренко В. С. «Проектування інформаційних систем» К. : Академія, 2002, укр.;
3. Бородакий Ю. В., Лободинский Ю. Г. «Информационные технологии методы, процессы системы», М.: Радио и связь, 2002, рус..
4. Маклаков С. В. «Моделирование бизнес-процессов с BPwin 4.0», М.: ДИАЛОГГМИФИ, 2002, рус.

Структура заняття:

VII. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

VIII. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

IX. Виклад нового матеріалу:

План:

1. Основні поняття SADT
2. SADT-моделі, точка зору проектування
3. Концепції методології SADT
4. Правила побудови моделі SADT

X. Узагальнення та систематизація знань;

XI. Підведення підсумків занять;

XII. Домашнє завдання:

4. Ознайомитись з теоретичними відомостями теми 2.2.2.

5. Вивчити основні поняття лекції.
6. Дати відповіді на контрольні питання.

Контрольні питання:

5. Що таке SADT-модель?
6. На яких концепціях базується SADT-модель?
7. Як виявляється місце розташування діаграми в ієрархії?
8. Як відбувається декомпозиція SADT-моделі?

Використання експертних систем, мов четвертого покоління і систем автоматизованого виробництва постійно розширюється. Успіх цих систем безпосередньо залежить від нашої здатності передувати їх розробці і впровадження опису всього комплексу проблем, які необхідно вирішити, вказівкою того, які функції системи повинні бути автоматизовані, визначенням точок інтерфейсу людини – машина і того, як взаємодіє система зі своїм оточенням. Іншими словами етап проектування є критичним для створення високоякісних систем.

Системне проектування – це дисципліна, що визначає підсистеми, компоненти і способи їх з'єднання, а також задає обмеження, при яких система повинна функціонувати, та вибирає найбільш ефективне поєднання людей, машин і програмного забезпечення для реалізації системи.

Під словом *«система»* ми розуміємо сукупність взаємодіючих компонент і взаємозв'язків між ними. Світ, в якому ми живемо, можна розглянути як складну взаємозв'язану сукупність природніх і штучних систем. Це можуть бути достатньо *складні* системи (наприклад, планети у складі Сонячної системи), системи *середньої складності* (космічний корабель) або *надскладні* системи (системи молекулярних взаємодій в живих організмах).

Під терміном *«моделювання»* ми розуміємо процес створення точного опису системи. Особливо важким виявляється опис систем середньої складності, таких, як система комутації в телефонних мережах, управління аероповітряними перевезеннями, збірка автівок тощо. З погляду людини, ці системи описати достатньо важко, тому що вони настільки великі, що практично неможливо перелічити всі їх компоненти із своїми взаємозв'язками, і в той же час недостатньо великі для вживання загальних спрощуючих припущень.

SADT (Технологія структурного аналізу і проектування) – це графічні позначення для опису систем.

Деякі області ефективного застосування SADT:

- Програмне забезпечення телефонних мереж
- Системна підтримка і діагностика
- Довгострокове і стратегічне планування
- Автоматизоване виробництво і проектування
- Конфігурація комп'ютерних систем
- Навчання персоналу
- Вбудоване програмне забезпечення оборонних систем
- Управління фінансами і матеріально-технічним постачанням

Широкий спектр областей указує на універсальність і потужність методології SADT. У програмі інтегрованої комп'ютеризації виробництва (ICAM) Міністерства оборони США була визначена корисність SADT, що привело до стандартизації і публікації її частини, званою IDEF-0. Під назвою IDEF-0 SADT використовувалась тисячами фахівців у військових і промислових організаціях.

Після виникнення SADT розробники усвідомили необхідність більшої впорядкованості. Таким чином, розробники почали формалізувати процес створення системи, розбиваючи його на наступні фази:

- Аналіз – визначення того, що система робитиме
- Проектування – визначення підсистем і їх взаємодія
- Реалізація – розробка підсистем окремо
- Об'єднання – з'єднання підсистем в єдине ціле
- Тестування – перевірка роботи системи
- Установка – введення системи в дію
- Функціонування – використання системи.

SADT – це методологія, розроблена спеціально для того, щоб полегшити опис і розуміння штучних систем, які потрапляють в розряд середньої складності.

SADT-модель – це опис системи. В SADT – моделях використовуються як природня так і графічна мови. Для передачі інформації про конкретну систему джерелом природньої мови служить людина, що описує систему, а джерелом графічної мови – сама методологія SADT.

З погляду SADT модель може бути зусереджена або на функціях системи, або на її об’єктах. SADT – моделі, орієнтовані на функції, прийнято називати функціональними моделями, а орієнтовані на об’єкти системи – моделями даних. Функціональна модель представляє з наобхідним ступенем деталізації систему функцій, які у свою чергу відображають свої взаємозв’язки через об’єкти системи.

При визначені моделі необхідно визначити позицію, з якої спостирігатеметься система, і створюватиметься її модель. Якість опису системи різко знижується, якщо вона не сфокусована ні на чому, SADT вимагає, щоб модель розглядалась весь час з однієї і тієїж позиції. Ця позиція називається *точкою зору* даної моделі.

Точку зору краще всього уявляти собі як місце (позиую) людини або об’єкту, в яке потрібно стати, щоб побачити систему в дії. З цієї фіксованої точки зору можна створити злагоджений опис системи так, щоб модель не дрейфувала навкруги, і в ній не змішувалися б незв’язані описи.

SADT – модель об’єднує і організовує діаграми в ієрархічні структури, в яких діаграми верхніх рівнів менш деталізовані ніж діаграми нижніх рвнів. Іншими словами, SADT – модель можна представити у вигляді деревовидної структури діаграм, де верхня діаграма є самою загальною, а самі нижні найбільш деталізовані.

Діаграма є основним робочим елементом при створені моделі. Розробник діаграм і моделей звичайно називається аналітиком, або, в термінології SADT автором. Діаграми мають власні синтаксичні правила відмінні від синтаксичних правил моделей.

Кожна SADT – діаграма містить блоки і дуги. *Блоки* зображують функції моделюємої системи. *Дуги* зв’язують блоки разом і відображають взаємодії і взаємозв’язки між ними.

Функціональні блоки на діаграмах зображуються прямокутниками. Блок представляє функцію або активну частину системи, наприклад, визначити ступінь виконання завдання, вибрати інструменти, підготовити робоче місце.

На відміну від інших графічних методів структурного аналізу в SADT кожна сторона блоку має особливе, цілком визначене призначення. Ліва сторона блоку призначена для входів, верхня – для управління, права – для виходів, нижня – для механізмів. Таке позначення відображає певні системні принципи: входи

перетворюються у виходи, управління обмежує або указуєумови виконання перетворень, механізми показують хто, що і як виконує функцію.

Дуги на SADT – діаграмі зображуються суцільними смугами із стрілками на кінцях. Для функціональних SADT – діаграм дуга представляє безліч об'єктів. Тут використовується загальне поняття об'єкт, оскільки дуги в SADT можуть представляти, наприклад, плани, даны в комп'ютерах, машини, інформацію тощо.

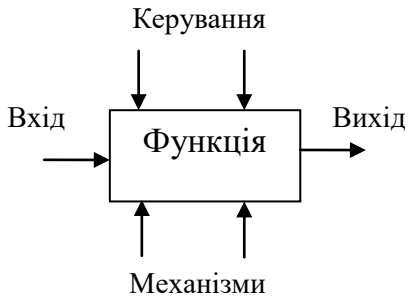
Методологія SADT є сукупністю методів, правил і процедур, призначених для побудови функціональної моделі об'єкту будь-якої предметної області. Функціональна модель SADT відображає функціональну структуру об'єкту, тобто виділені нею дії і зв'язки між цими діями. Основні елементи цієї методології грунтуються на наступних концепціях:

1. Графічне представлення блокового моделювання. Графіка блоків і дуг SADT – діаграми відображає функцію у вигляді блоку, а інтерфейси входу/виходу представляються дугами, які відповідно входять в блок і виходять з нього. Взаємодія блоків один з одним описується за допомогою інтерфейсних дуг виражаючих «обмеження», які у свою чергу визначають, коли і ким функції виконуються і управляються.
2. Концепція строгості і точності. Виконання правил SADT вимагає достатньої строгості і точності, не накладаючи при тому великих обмежень на дії аналітика. Правила SADT включають:
    a. Обмеження кількості блоків на кожному рівні (3-6 блоків)
    b. Зв'язаність діаграм (номери блоків)
    c. Унікальність міток і найменувань (відсутність імен, що повторюються)
    d. Синтаксичні правила для графіки блоків і дуг
    e. Розділ входів і управління (правило визначення ролі даних)

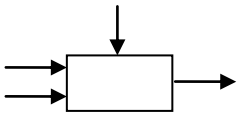
Методологія SADT може використовуватися для моделювання ширового круги систем і визначення вимог і функцій, а потім для розробки системи, яка задовльнятиме цим вимогам і реалізовуватиме ці функції. Вже існуючі системи SADT можуть використовуватися для аналізу функцій, що виконуються системою. А також для вказівки механізмів, за допомогою яких вони здійснюються.

Результатом використання методології SADT є модель, яка складається з діаграм та фрагментів текстів, що мають посилання один на одного.

Однією з найважливіших особливостей методології є поступове введення все більшої кількостірівнів деталізації в ході створення діаграм, які відображають модель.

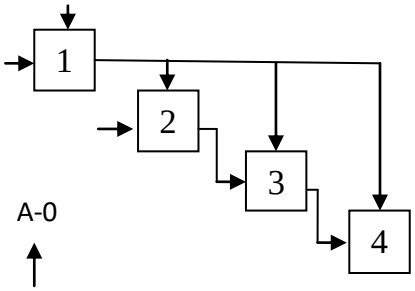


Функціональний блок і інтерфейсні дуги



Найбільш загальне представлення

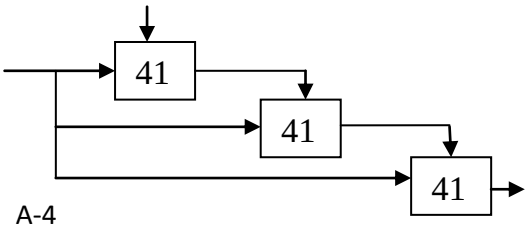
A-0



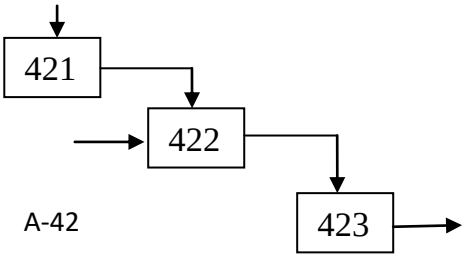
Більш детальне представлення

A-0

Ця діаграма є «батьківською» для цієї діаграми



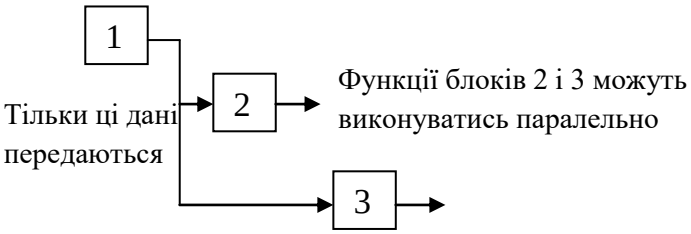
A-4



A-42

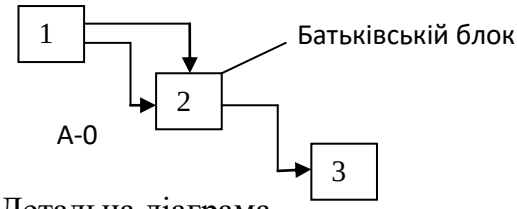
Структура SADT – моделі дікомпозиція діаграм

На наступних малюнках представлені різні варіанти використання функцій і з’єднання дуг з блоками.



Одночасне виконання

Батьківська діаграма

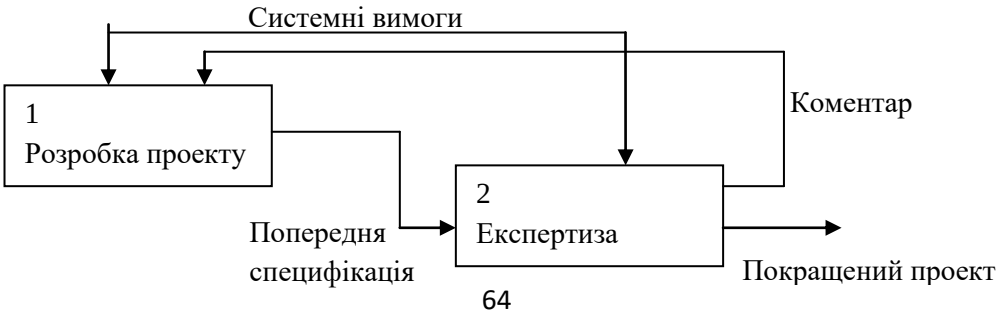


Детальна діаграма

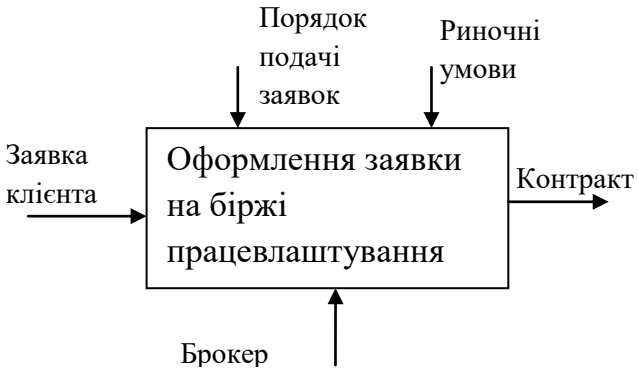
Деякі дуги приєднані до блоків обома кінцями, у інших жу один кінець залишається не приєднаним. Не приєднані дуги відповідають входам, управлінням і виходам батьківського блоку. Джерело або одержувача цих *прикордонних* дуг може бути знайдено тільки на батьківській діаграмі. Не приєднані кінці повинні відповідати дугам на початковій діаграмі. Всі граничні дуги повинні продовжуватися на батьківській діаграмі, щоб вона була повною і несуперечною.

На SADT-діаграмах не вказані явно ні послідовність, ні час. Зворотні зв’язки, ітеріції, процеси і функції, що перекриваються за часом можуть бути зображені за допомогою дуг. Зворотні зв’язки можуть виступати у вигляді коментарів, зауважень, виправлень тощо.

Приклад зворотнього зв’язку



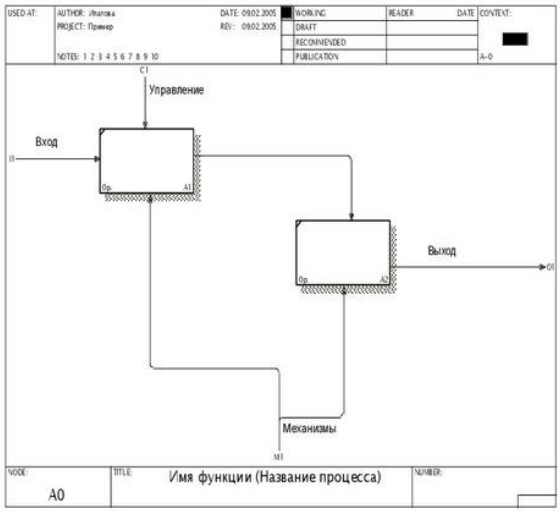
Як було відзначено механізми (дуги з нижньої сторони) показують засоби, за допомогою яких здійснюється виконання функцій. Механізм може бути людиною, комп’ютером або будь-яким іншим пристроєм, який допомагає виконувати дану функцію.



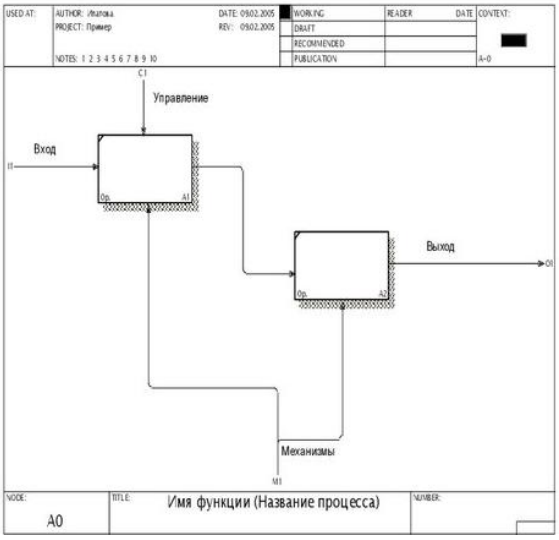
Приклад механізму

Взаємовплив блоків може виражатися або в пересиланні Виходу до інший функції для подальшого перетворення, або в виробленні керуючою інформації, розпорядчої, що саме повинна робити інша функція. Існують п'ять типів взаємозв'язків між блоками для опису їх відносин:

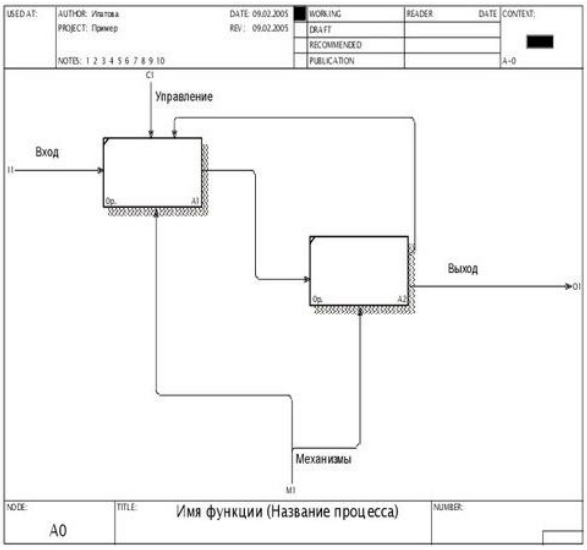
- управління,
- Вхід,
- Зворотній зв`язок по Управлінню,
- Зворотній зв`язок по Входу,
- Вихід-Механізм.



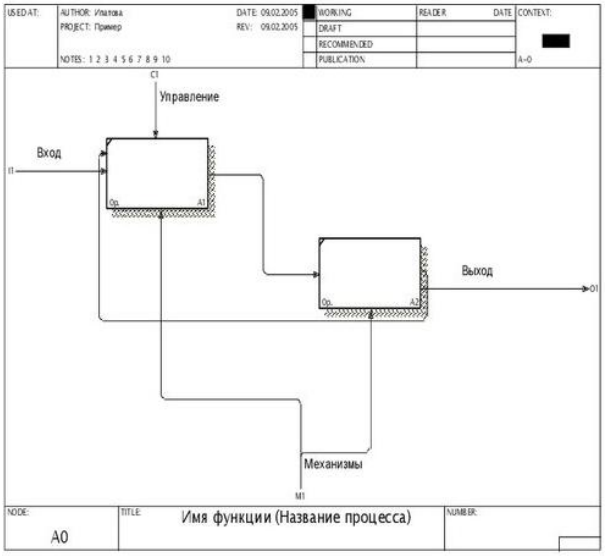
Відношення управління



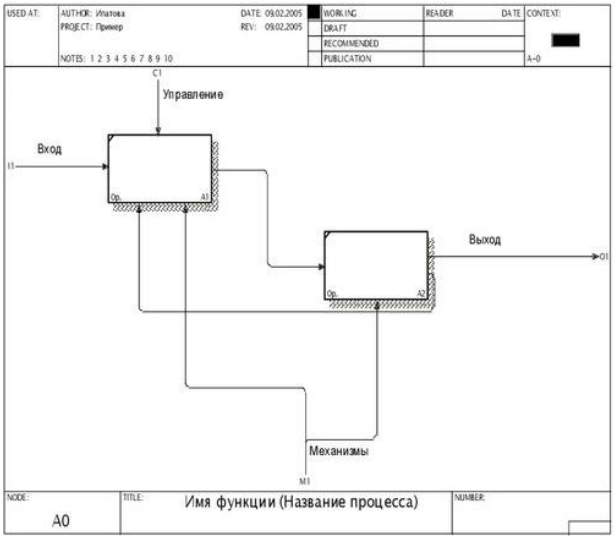
Відношення Вхід



Зворотній зв`язок по Управлінню



Зворотній зв`язок по Входу



Вихід-Механізм

Основні поняття лекції:

Системне проектцвання – це дисципліна, що визнацає підсистеми, компоненти і способи іх з'єднання, а також задає обмеження, при яких система повинна функціонувати, та вибирає найбільш ефективне поєднання людей, машин і програмного забезпечення для реалізації системи.

Моделювання – процес створення точного опису системи.

SADT–модель – це організація діаграм в ієрархічні структури, в яких діаграми верхніх рівнів менш деталізовані ніж діаграми нижніх рвнів.

Діаграма є основним робочим елементом при створені моделі.

Блоки зображують функції моделюємої системи. *Дуги* зв'язують блоки разом і відображають взаємодії і взаємозв'язки між ними.

Тема 3.2.3 Діаграма потоків даних. Нотація Йордана-Де Марко

(4 години)

Мета: Розглянути етапи виконання проєкту, контекстні діаграми нотацією Йордана-Де Марко.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План:

1. Фаза аналізу;
2. Аналіз поведінки системи:
   a. Початкова контекстна діаграма;
   b. Матриця списку подій;
   c. Контекстна діаграма;
3. Аналіз даних:
   a. Діаграма структур даних;
   b. ER-діаграма.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми.
2. Дати відповіді на контрольні питання.

Контрольні питання:

1. На які фази розбивається проект?
2. Що входить до фази аналізу?
3. З чого починається аналіз поведінки системи?
4. Як позначається процес на контекстній діаграмі?
5. Що входить в матрицю списка подій?
6. Події яких типів бувають?

7. Як відбувається побудова контекстної діаграми?
8. Як виділити конкретні потоки повідомлень з абстрактних грунтуючись на матриці списку подій?

У якості предметної області використовується опис роботи відеопрокату, яка одержує запити на фільми від клієнтів і стрічки, що вертаються клієнтами. Запити розглядаються адміністрацією відеопрокату з використанням інформації про клієнтів, фільми й стрічках. При цьому перевіряється й обновляється список орендованих стрічок, а також перевіряються записи про членство в бібліотеці. Адміністрація контролює також повернення стрічок, використовуючи інформацію про фільми, стрічки й список орендованих стрічок, який обновляється. Обробка запитів на фільми й повернень стрічок включає наступні дії: якщо клієнт не є членом бібліотеки, він не має права на оренду. Якщо необхідний фільм є в наявності, адміністрація інформує клієнта про орендну плату. Однак, якщо клієнт прострочив строк повернення наявних у нього стрічок, йому не дозволяється брати нові фільми. Коли стрічка вертається, адміністрація розраховує орендну плату плюс пені за несвоєчасне повернення.

Відеопрокат одержує нові стрічки від своїх постачальників. Коли нові стрічки надходять у бібліотеку, необхідна інформація про них фіксується. Інформація про членство в бібліотеці втримується окремо від записів про оренду стрічок.

Адміністрація бібліотеки регулярно готовить звіти за певний період часу про члени прокату, постачальниках стрічок, видачі певних стрічок і стрічках, придбаних бібліотекою.

Увесь проект розділяється на 4 фази: аналіз, глобальне проектування (проектування архітектури системи), детальне проектування й реалізація (програмування).

На фазі аналізу будується **модель середовища** (Environmental Model). Побудова моделі середовища включає:

- **аналіз поведінки системи** (визначення призначення ИС, побудова початкової контекстної діаграми потоків даних (DFD) і формування матриці списку подій (ELM), побудова контекстних діаграм);

- **аналіз даних** (визначення складу потоків даних і побудова діаграм структур даних (DSD), конструювання глобальної моделі даних у вигляді Er-Діаграми).

Призначення ИС визначає угода між проектувальниками й замовниками щодо призначення майбутньої ИС, загальний опис ИС для самих проектувальників і границі ИС. Призначення фіксується як текстовий коментар в "нульовому" процесі контекстної діаграми.

Наприклад, у цьому випадку призначення ИС формулюється в такий спосіб: ведення бази даних про члени бібліотеки, фільмах, оренді й постачальниках. При цьому керівництво бібліотеки повинне мати можливість одержувати різні види звітів для виконання своїх завдань.

Перед побудовою контекстної DFD необхідно проаналізувати зовнішні події (зовнішні об'єкти), що виявляють вплив на функціонування бібліотеки. Ці об'єкти взаємодіють із ИС шляхом інформаційного обміну з нею.

З опису предметної області випливає, що в процесі роботи бібліотеки беруть участь наступні групи людей: клієнти, постачальники й керівництво. Ці групи є зовнішніми об'єктами. Вони не тільки взаємодіють із системою, але також визначають її границі й зображуються на початковій контекстної DFD як термінатори (зовнішні сутності).

Початкова контекстна діаграма зображено на малюнку 1. На відміну від нотації Gane/Sarson зовнішні сутності позначаються звичайними прямокутниками, а процеси - окружностями.

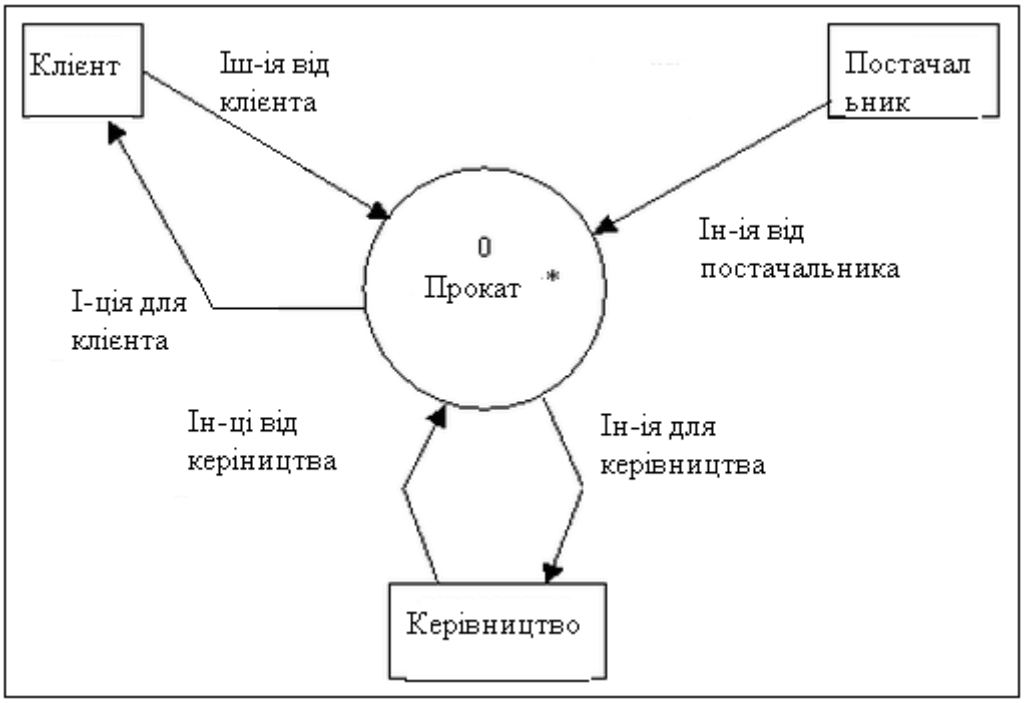


Рисунок 1 – Початкова контекстна діаграма

Список подій будується у вигляді матриці (ELM) і описує різні дії зовнішніх сутностей і реакцію ИС на них. Ці дії являють собою зовнішні події, що впливають на бібліотеку. Розрізняють наступні типи подій:

Абревіатура	Тип
NC	Нормальне керування
ND	Нормальні дані
NCD	Нормальне керування/дані
TC	Тимчасове керування
TD	Тимчасові дані
TCD	Тимчасове керування/дані

Усі дії позначаються як нормальні дані. Ці дані є подіями, які ИС сприймає безпосередньо, наприклад, зміна адреси клієнта, яке повинне бути відразу зареєстроване. Вони з'являються в DFD у якості вмісту потоків даних.

Матриця списку подій має такий вигляд:

№	Опис	**Тип**	Реакція

1	Клієнт бажає стати членом прокату	ND	Реєстрація клієнта як члена прокату
2	Клієнт повідомляє про зміну адреси	ND	Реєстрація зміненого адреси клієнта
3	Клієнт запитує оренду фільму	ND	Розгляд запиту
4	Клієнт повертає фільм	ND	Реєстрація повернення
5	Керівництво надає нові повноваження постачальникові	ND	Реєстрація постачальника
6	Постачальник повідомляє про зміну адреси	ND	Реєстрація зміненого адреси постачальника
7	Постачальник направляє фільм у бібліотеку	ND	Одержання нового фільму
8	Керівництво запитує новий звіт	ND	Формування необхідного звіту для керівництва

Для завершення аналізу функціонального аспекту поведінки системи будується повна контекстна діаграма, що включає діаграму нульового рівня. При цьому процес "бібліотека" декомпозируется на 4 процесу основні види, що відбивають, адміністративної діяльності бібліотеки. Існуючі "абстрактні" потоки даних між термінаторами й процесами трансформуються в потоки, що представляють обмін даними на більш конкретному рівні. Список подій показує, які потоки існують на цьому рівні: кожна подія зі списку повинне формувати деякий потік (подія формує вхідний потік, реакція - вихідний потік). Один "абстрактний" потік може бути розділено на більш ніж один "конкретний" потік.

Потоки на діаграмі верхнього рівня	**Потоки на діаграмі нульового рівня**

Інформація від клієнта	Дані про клієнта, Запит про оренду
Інформація для клієнта	Членська картка, Відповідь на запит про оренду
Інформація від керівництва	Запит звіту про нові члени, Новий постачальник, Запит звіту про постачальників, Запит звіту про оренду, Запит звіту про фільми
Інформація для керівництва	Звіт про нові члени, Звіт про постачальників, Звіт про оренду, Звіт про фільми
Інформація від постачальника	Дані про постачальника, Нові фільми

На наведеної DFD (рисунок 1) накопичувач даних "прокат" є глобальним або абстрактною виставою сховища даних.

Аналіз функціонального аспекту поведінки системи дає вистава про обмін і перетворенні даних у системі. Взаємозв'язок між "абстрактними" потоками даних і "конкретними" потоками даних на діаграмі нульового рівня виражається в діаграмах структур даних (рисунок 2).

На фазі аналізу будується глобальна модель даних, що представляється у вигляді діаграми " сутність-зв'язок" (рисунок 3).

Між різними типами діаграм існують наступні взаємозв'язки:

- ELM-DFD: події - вхідні потоки, реакції - вихідні потоки
- DFD-DSD: потоки даних - структури даних верхнього рівня
- DFD-ERD: накопичувачі даних - Er- Діаграми
- DSD-ERD: структури даних нижнього рівня - атрибути сутностей

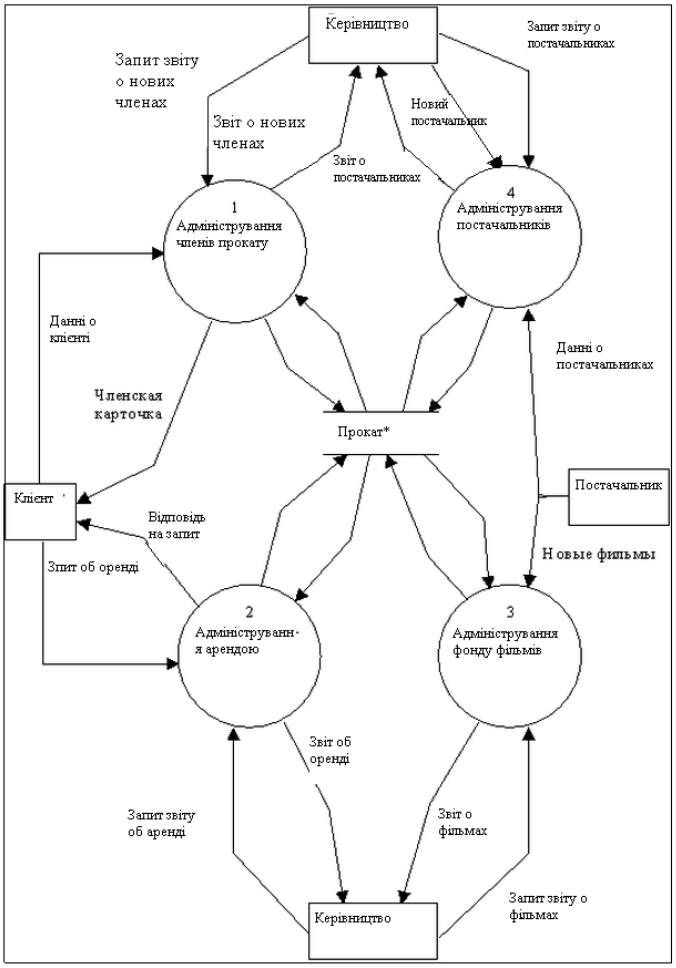


Рисунок 2 –контекстна діаграма

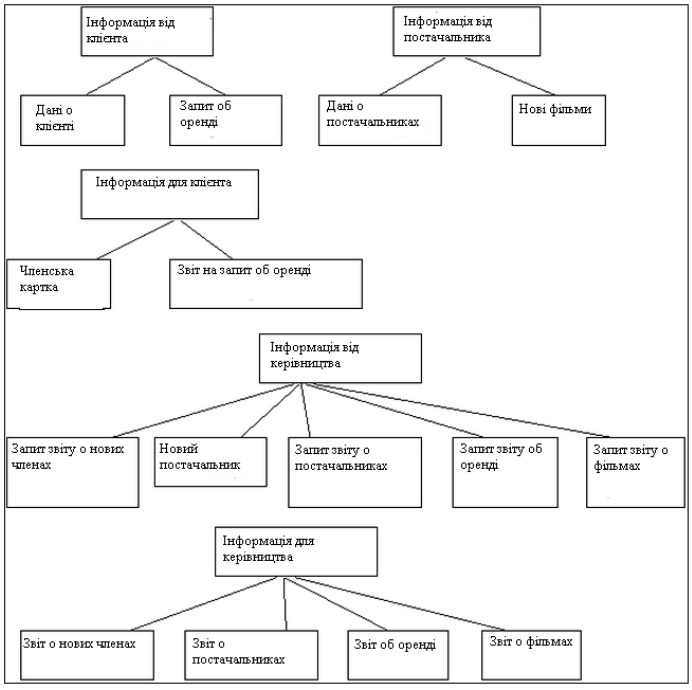


Рисунок 3 – діаграма структур даних

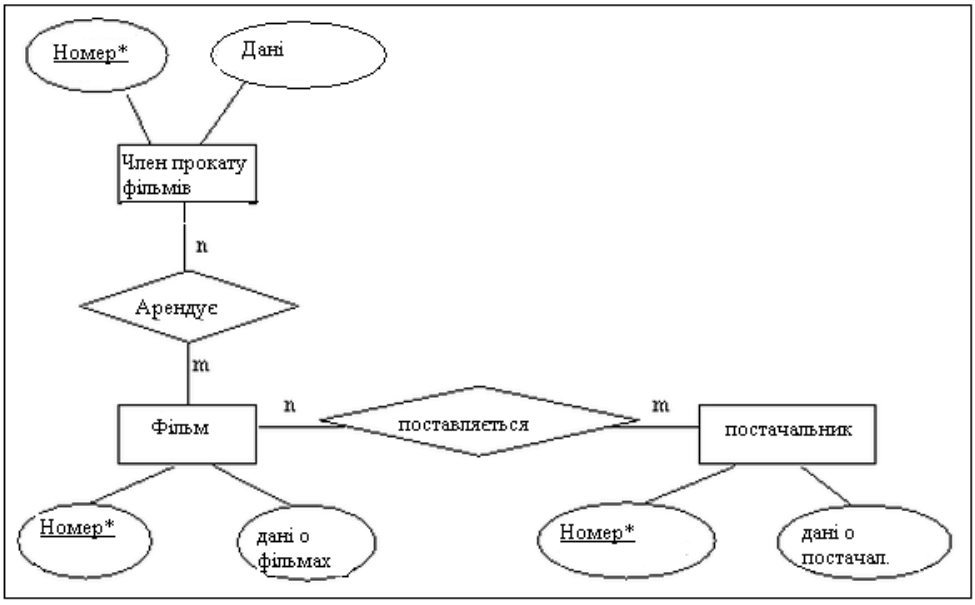


Рисунок 4 – ER-діаграма

На фазі **проектування архітектури** будується предметна модель. Процес побудови предметної моделі містить у собі:

- детальний опис функціонування системи;
- подальший аналіз використовуваних даних і побудова логічної моделі даних для наступного проектування бази даних;
- визначення структури користувацького інтерфейсу, специфікації форм і порядку їх появи;
- уточнення діаграм потоків даних і списку подій, виділення серед процесів нижнього рівня інтерактивних і неінтерактивних, визначення для них миниспецификаций.

Результатами проектування архітектури є:

- модель процесів (діаграми архітектури системи (SAD) і миниспецификации структурованою мовою);
- модель даних (ERD і підсхеми ERD);
- модель користувацького інтерфейсу (класифікація процесів на інтерактивні й неінтерактивні функції, діаграма послідовності форм (FSD - Form Sequence Diagram), що показує, які форми з'являються в додатку й у якому порядку. На FSD фіксується набір і структура викликів екранних форм. Діаграми FSD утворюють ієрархію, на вершині якої перебуває головна форма додатка, що реалізує підсистему. На другому рівні перебувають форми, що реалізують процеси нижнього рівня функціональної структури, зафіксованої на діаграмах SAD.

На **фазі детального проектування** будується модульна модель. Під модульною моделлю розуміється реальна модель проектованої прикладної системи. Процес її побудови містить у собі:

- уточнення моделі бази даних для наступної генерації Sql- Пропозицій;
- уточнення структури користувацького інтерфейсу;
- побудова структурних схем, що відбивають логікові роботи користувацького інтерфейсу й модель бізнес- логіки (Structure Charts Diagram - SCD) і прив'язка їх до форм.

Результатами детального проектування є:

- модель процесів (структурні схеми інтерактивних і неінтерактивних функцій);
- модель даних (визначення в ERD усіх необхідних параметрів для додатків);
- модель користувацького інтерфейсу (діаграма послідовності форм (FSD), що показує, які форми з'являються в додатку й у якому порядку, взаємозв'язок між кожною формою й певною структурною схемою, взаємозв'язок між кожною формою й однієї або більш сутностями в ERD).

На **фазі реалізації** будується реалізаційна модель. Процес її побудови містить у собі:

- генерацію Sql- Пропозицій, що визначають структуру цільовий БД (таблиці, індекси, обмеження цілісності);
- уточнення структурних схем (SCD) і діаграм послідовності форм (FSD) з наступною генерацією коду додатків.

На основі аналізу потоків даних і взаємодії процесів зі сховищами даних здійснюється остаточне виділення підсистем (попереднє повинне було бути зроблене й зафіксоване на етапі формулювання вимог у технічному завданні). При виділенні підсистем необхідно керуватися принципом функціональної зв'язаності й принципом мінімізації інформаційної залежності. Необхідно враховувати, що на підставі таких елементів підсистеми як процеси й дані на етапі розробки повинне бути створений додаток, здатне функціонувати самостійно. З іншої сторони при угрупованні процесів і даних у підсистеми необхідно враховувати вимоги до конфігурування продукту, якщо вони були сформульовані на етапі аналізу.

Тема 3.2.4 Діаграма потоків даних. Нотація Гейна Сарсона

(2 години)

Мета: Ознайомитись з теорією діаграм потоків даних та їх нотацією Йордана-Де Марко.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План:

1. Зовнішні сутності;
2. Системи та підсистеми;
3. Процеси;
4. Накопичувачі даних;
5. Побудова ієрархій діаграм потоків даних;

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

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

Контрольні питання:

1. Що таке ДПД?
2. Перелічіть основні компоненти ДПД?
3. Що є зовнішньою сутністю?
4. Як графічно позначається підсистема?
5. Що таке копичувач даних?
6. Як відбувається побудова ієрархії діаграм потоків даних?
7. Які правила повинні виконуватися при деталізації ДПД?

В основі даної методології (методології Gane/Sarson) лежить побудова

моделі аналізованої ІС - проектованої або реально існуючої. Відповідно до методології модель системи визначається як ієрархія **діаграм потоків даних** (ДПД або DFD), що описують асинхронний процес перетворення інформації від її введення в систему до видачі користувачеві. Діаграми верхніх рівнів ієрархії (контекстні діаграми) визначають основні процеси або підсистеми ІС із зовнішніми входами й виходами. Вони деталізуються за допомогою діаграм нижнього рівня. Така декомпозиція триває, створюючи багаторівневу ієрархію діаграм, доти, поки не буде досягнутий такий рівень декомпозиції, на якому процес стають елементарними й деталізувати їх далі неможливо.

Джерела інформації (зовнішні сутності) породжують інформаційні потоки (потоки даних), що переносять інформацію до підсистем або процесів. Ті у свою чергу перетворюють інформацію й породжують нові потоки, які переносять інформацію до інших процесів або підсистем, накопичувачів даних або зовнішнім сутностям - споживачам інформації. Таким чином, основними компонентами діаграм потоків даних є:

- зовнішні сутності;
- системи/підсистеми;
- процеси;
- накопичувачі даних;
- потоки даних.

І. Зовнішня сутність

Зовнішня сутність являє собою матеріальний предмет або фізичну особу, що представляє собою джерело або приймач інформації, наприклад, замовники, персонал, постачальники, клієнти, склад. Визначення деякого об'єкта або системи як зовнішньої сутності вказує на те, що вона перебуває за межами границь аналізованої ІС. У процесі аналізу деякі зовнішні сутності можуть бути перенесені усередину діаграми аналізованої ІС, якщо це необхідно, або, навпаки, частина процесів ІС може бути винесена за межі діаграми й представлена як зовнішня сутність.

Зовнішня сутність позначається квадратом (Рисунок 4.26), розташованим як би "над" діаграмою, що й кидають на неї тінь, для того, щоб можна було виділити цей символ серед інших позначень:

Рисунок 1 – Зовнішня сутність

II. Системи й підсистеми

При побудові моделі складної ІС вона може бути представлена в самому загальному виді на так званій контекстній діаграмі у вигляді однієї системи як єдиного цілого, або може бути декомпозирована на ряд підсистем.

Підсистема (або система) на контекстній діаграмі зображується в такий спосіб (рисунок 2).

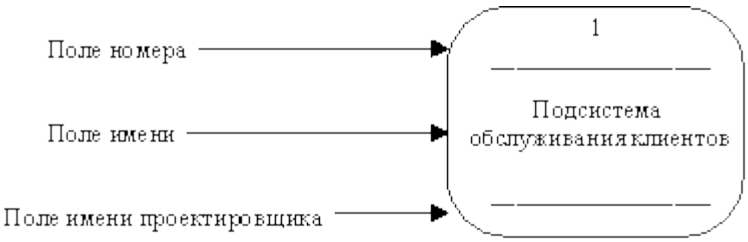


Рисунок 2 – Підсистема

Номер підсистеми служить для її ідентифікації. У поле імені вводиться найменування підсистеми у вигляді пропозиції з підлягаючим і відповідними визначеннями й доповненнями.

III. Процеси

Процес являє собою перетворення вхідних потоків даних у вихідні відповідно до певного алгоритму. Фізично процес може бути реалізований різними способами: це може бути підрозділ організації (відділ), що виконує обробку вхідних документів і випуск звітів, програма, апаратно реалізоване логічне обладнання і т.д.

Процес на діаграмі потоків даних зображується, як показано на рисунку 3.

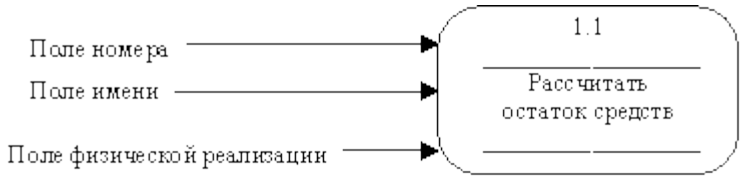


Рисунок 3 – процес

Номер процесу служить для його ідентифікації. У поле імені вводиться найменування процесу у вигляді пропозиції з активним недвозначним дієсловом у

невизначеній формі (обчислити, розрахувати, перевірити, визначити, створити, одержати), за яким ідуть іменники в знахідному відмінку, наприклад:

"Увести відомості про клієнтів";

"Уидати інформацію про поточні витрати";

"Перевірити кредитоспроможність клієнта".

Використання таких дієслів, як "обробити", "модернізувати" або "відредагувати" означає, як правило, недостатньо глибоке розуміння даного процесу й вимагає подальшого аналізу.

Інформація в поле фізичної реалізації показує, який підрозділ організації, програма або апаратне обладнання виконує даний процес.

IV. Накопичувачі даних

Накопичувач даних являє собою абстрактне обладнання для зберігання інформації, яку можна в будь-який момент помістити в накопичувач і через якийсь час витягти, причому способи приміщення й витяги можуть бути будь-якими.

Накопичувач даних може бути реалізований фізично у вигляді микрофиши, ящика в картотеці, таблиці в оперативній пам'яті, файлу на магнітному носії і т.д. Накопичувач даних на діаграмі потоків даних зображується, як показано на рисунку 4.

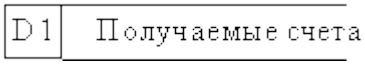


Рисунок 4 – накопичувач даних

Накопичувач даних ідентифікується буквою "D" і довільним числом. Ім'я накопичувача вибирається з міркування найбільшої інформативності для проектувальника.

Накопичувач даних у загальному випадку є прообразом майбутньої бази даних і опис, що зберігаються в ньому даних повинне бути вв'язане з інформаційною моделлю.

V. Потоки даних

Потік даних визначає інформацію, передану через деяке з'єднання від джерела до приймача. Реальний потік даних може бути інформацією, переданої по кабелю між двома обладнаннями, що пересилаються поштою листами, магнітними стрічками або дискетами, стерпними з одного комп'ютера на інший і т.д.

Потік даних на діаграмі зображується лінією, що кінчається стрілкою, яка

показує напрямок потоку (рисунок 5). Кожний потік даних має ім'я, що відбиває його зміст.

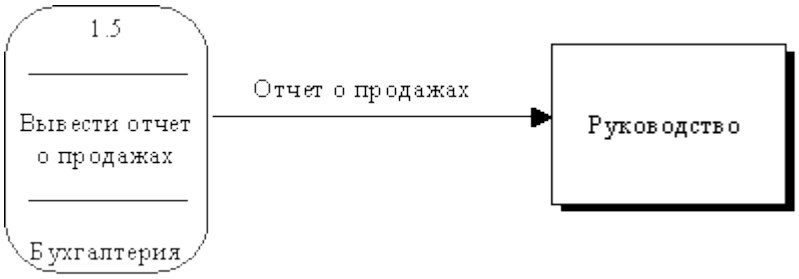


Рисунок 5 – Потік даних

VI.Побудова ієрархії діаграм потоків даних

Першим кроком при побудові ієрархії ДПД є побудова контекстних діаграм. Звичайно при проектуванні щодо простих ІС будується єдина контекстна діаграма із зіркоподібною топологією, у центрі якої перебуває так званий головний процес, з'єднаний із приймачами й джерелами інформації, за допомогою яких із системою взаємодіють користувачі й інші зовнішні системи.

Якщо ж для складної системи обмежитися єдиною контекстною діаграмою, то вона буде містити занадто велика кількість джерел і приймачів інформації, які важко розташувати на аркуші паперу нормального формату, і крім того, єдиний головний процес не розкриває структури розподіленої системи. Ознаками складності (у змісті контексту) можуть бути:

- наявність великої кількості зовнішніх сутностей (десять і більш);
- розподілена природа системи;
- багатофункціональність системи із уже сложившейся або виявленим угрупованням функцій в окремі підсистеми.

Для складних ІС будується ієрархія контекстних діаграм. При цьому контекстна діаграма верхнього рівня містить не єдиний головний процес, а набір підсистем, з'єднаних потоками даних. Контекстні діаграми наступного рівня деталізують контекст і структуру підсистем.

Ієрархія контекстних діаграм визначає взаємодія основних функціональних підсистем проектованої ИС як між собою, так і із зовнішніми вхідними й вихідними потоками даних і зовнішніми об'єктами (джерелами й приймачами інформації), з якими взаємодіє ИС.

Розробка контекстних діаграм вирішує проблему строгого визначення

функціональної структури ИС на самій ранній стадії її проектування, що особливо важливо для складних багатофункціональних систем, у розробці яких беруть участь різні організації й колективи розроблювачів.

Після побудови контекстних діаграм отриману модель слід перевірити на повноту вихідних даних про об'єкти системи й ізольованість об'єктів (відсутність інформаційних зв'язків з іншими об'єктами).

Для кожної підсистеми, що присутствующей на контекстних діаграмах, виконується її деталізація за допомогою ДПД. Кожний процес на ДПД, у свою чергу, може бути деталізований за допомогою ДПД або миниспецификации. При деталізації повинні виконуватися наступні правила:

1. правило балансування - означає, що при деталізації підсистеми або процесу діаграма, що деталізує, у якості зовнішніх джерел/приймачів даних може мати тільки ті компоненти (підсистеми, процеси, зовнішні сутності, накопичувачі даних), з якими має інформаційний зв'язок детализируемая підсистема або процес на батьківській діаграмі;
2. правило нумерації - означає, що при деталізації процесів повинна підтримуватися їхня ієрархічна нумерація. Наприклад, процеси, що деталізують процес із номером 12, одержують номери 12.1, 12.2, 12.3 і т.д.

Мініспецифікація (опис логіки процесу) повинна формулювати його основні функції таким чином, щоб надалі фахівець, що виконує реалізацію проекту, смог виконати їх або розробити відповідну програму.

Мініспецифікація є кінцевою вершиною ієрархії ДПД. Розв'язок про завершення деталізації процесу й використання миниспецификации ухвалюється аналітиком виходячи з наступних критеріїв:

- наявності в процесу щодо невеликої кількості вхідних і вихідних потоків даних (2-3 потоку);
- можливості опису перетворення даних процесом у вигляді послідовного алгоритму;
- виконання процесом єдиної логічної функції перетворення вхідної інформації у вихідну;

- можливості опису логіки процесу за допомогою миниспецификации невеликого обсягу (не більш 20-30 рядків).

При побудові ієрархії ДПД переходити до деталізації процесів випливає тільки після визначення змісту всіх потоків і накопичувачів даних, яке описується за допомогою структур даних. Структури даних конструюються з елементів даних і можуть містити альтернативи, умовні входження й ітерації. Умовне входження означає, що даний компонент може отсутствовать у структурі. Альтернатива означає, що в структуру може входити один з перерахованих елементів. Ітерація означає входження будь-якого числа елементів у зазначеному діапазоні. Для кожного елемента даних може вказуватися його тип (безперервні або дискретні дані). Для безперервних даних може вказуватися одиниця виміру (кг, див і т.п.), діапазон значень, точність вистави й форма фізичного кодування. Для дискретних даних може вказуватися таблиця припустимих значень.

Після побудови закінченої моделі системи її необхідно верифицировать (перевірити на повноту й погодженість). У повній моделі всі її об'єкти (підсистеми, процеси, потоки даних) повинні бути докладно описані й деталізовані. Виявлені недеталізовані об'єкти слід деталізувати, повернувшись на попередні кроки розробки. У погодженій моделі для всіх потоків даних і накопичувачів даних повинне виконуватися правило збереження інформації: усі вступники куда-либо дані повинні бути лічені, а всі зчитувані дані повинні бути записані.

Основні поняття лекції:

Зовнішня сутність являє собою матеріальний предмет або фізичну особу, що представляє собою джерело або приймач інформації, наприклад, замовники, персонал, постачальники, клієнти, склад.

Процес являє собою перетворення вхідних потоків даних у вихідні відповідно до певного алгоритму.

Накопичувач даних являє собою абстрактне обладнання для зберігання інформації, яку можна в будь-який момент помістити в накопичувач і через якийсь час витягти, причому способи приміщення й витяги можуть бути будь-якими.

Потік даних визначає інформацію, передану через деяке з'єднання від джерела до приймача.

Правило балансування - означає, що при деталізації підсистеми або процесу діаграма, що деталізує, у якості зовнішніх джерел/приймачів даних може мати тільки ті компоненти (підсистеми, процеси, зовнішні сутності, накопичувачі даних), з якими має інформаційний зв'язок детализируемая підсистема або процес на батьківській діаграмі.

Правило нумерації - означає, що при деталізації процесів повинна підтримуватися їхня ієрархічна нумерація.

Міністецифікація (опис логіки процесу) повинна формулювати його основні функції таким чином, щоб надалі фахівець, що виконує реалізацію проекту, смог виконати їх або розробити відповідну програму.

Тема 3.2.5 Сімейство методологій IDEF, принцип побудови моделей IDEF-3

(2 години)

Мета: Розглянути сімейство методологій IDEF. Отримати представлення про побудову моделей методології IDEF-3.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.
II. Повідомлення теми та мети заняття;
III. Виклад нового матеріалу:

План:

1. Сімейство методологій IDEF;
2. Методологія IDEF3;
   a. Роботи;
   b. Зв’язки;
   c. Перехрестя;

IV. Узагальнення та систематизація знань;
V. Підведення підсумків занять;
VI. Домашнє завдання:

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

Контрольні питання:

1. Які методології входять до сімейства IDEF?
2. Що дозволяє описати методологія IDEF3?
3. Які типи методології IDEF3 існують?
4. Що таке одиниця роботи?
5. Які види відносин використовуються в IDEF3?
6. Які види перехресть бувають?
7. В чому різниця між синхронними та асинхронними перехрестями?
8. Як графічно позначаються об’єкти та роботи в OSTN діаграмах?

IDEF - методології сімейства ICAM (Integrated Computer- Aided Manufacturing) для моделювання складних систем, дозволяє відображати й аналізувати моделі діяльності широкого спектра складних систем у різних розрізах. При використанні методології IDEF широта й глибина обстеження процесів у системі визначається розробником, що дозволяє не перевантажувати створювану модель зайвими даними.

Методолдогії сімества IDEF:

1. **IDEF0** (Function Modeling) - методологія функціонального моделювання. За допомогою наочної графічної мови IDEF0 досліджувана система з'являється перед розробниками й аналітиками у вигляді набору взаємозалежних функцій (функціональних блоків - у термінах IDEF0). Як правило, моделювання засобами IDEF0 є першим етапом вивчення будь-якої системи (SADT (Structured Analysis and Design Technique));
2. **IDEF1** (Information Modeling) - методологія моделювання інформаційних потоків усередині системи, що дозволяє відображати й аналізувати їхню структуру й взаємозв'язку. **IDEF1X** (IDEF1 Extended) - Data Modeling - методологія моделювання баз даних на основі моделі "сутність-зв'язок". Застосовується для побудови інформаційної моделі, яка представляє структуру інформації, необхідної для підтримки функцій виробничої

системи або середовища. Метод IDEF1, розроблений Т. Рэмей (T. Ramey), також заснований на підході П. Чена й дозволяє побудувати модель даних, еквівалентну реляційної моделі в третій нормальній формі.

3. **IDEF2 (**Simulation Model Design) - методологія динамічного моделювання розвитку систем. У зв'язку з досить серйозними складностями аналізу динамічних систем від цього стандарту практично відмовилися, і його розвиток призупинився на самому початковому етапі. У цей час присутні алгоритми і їх комп'ютерні реалізації, що дозволяють перетворювати набір статичних діаграм IDEF0 у динамічні моделі, побудовані на базі "розфарбованих мереж Петри" (CPN - Color Petri Nets);
4. **IDEF3 (**Process Description Capture (Документування технологічних процесів)) - методологія документування процесів, що відбуваються в системі (наприклад, на підприємстві), описуються сценарій і послідовність операцій для кожного процесу. IDEF3 має прямий взаємозв'язок з методологією IDEF0 - кожна функція (функціональний блок) може бути представлена у вигляді окремого процесу засобами IDEF3;
5. **IDEF4** (Object-oriented Design) - методологія побудови об'єктно-орієнтованих систем, дозволяють відображати структуру об'єктів і закладені принципи їх взаємодії, тим самим дозволяючи аналізувати й оптимізувати складні об'єктно-орієнтовані системи.

IDEF3 - це метод, що має основною метою дати можливість аналітикам описати ситуацію, коли процеси виконуються в певній послідовності, а також описати об'єкти, що беруть участь спільно в одному процесі.

Техніка опису набору даних IDEF3 є частиною структурного аналізу. На відміну від деяких методик описів процесів IDEF3 не обмежує аналітика надмірно твердими рамками синтаксису, що може привести до створення неповних або суперечливих моделей.

IDEF3 може бути також використаний як метод створення процесів. IDEF3 доповнює IDEF0 і містить усе необхідне для побудови моделей, які надалі можуть бути використані для імітаційного аналізу.

Існують два типи діаграм у стандарті IDEF3, що представляють опис того

самого сценарію технологічного процесу в різних ракурсах. Діаграми стосовні до першого типу називаються діаграмами **Опису Послідовності Етапів Процесу** (Process Flow Description Diagrams, **PFDD**), а до другого - діаграмами **Стану Об'єкта і його Трансформацій в Процесі** (Object State Transition Network, **OSTN**).

Розглянемо діаграми PFDD.

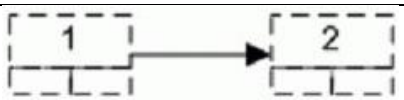
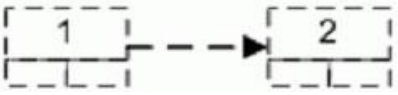
Кожна робота в IDEF3 описує який-небудь сценарій бізнес- процесу й може бути складовою іншої роботи. Оскільки сценарій описує мету й рамки моделі, важливо, щоб роботи йменувалися віддієслівним іменником, що позначають процес дії, або фразою, що містить такий іменник.

Точка зору на модель повинна бути документована. Звичайно це точка зору людину, відповідального за роботу в цілому. Також необхідно документувати мету моделі - ті питання, на які покликано відповісти модель.

Діаграма є основною одиницею опису в IDEF3. Важливо правильно побудувати діаграми, оскільки вони призначені для читання іншими людьми (а не тільки автором).

Одиниці роботи - Unit of Work (UOW) - також називані роботами (activity), є центральними компонентами моделі. В IDEF3 роботи зображуються прямокутниками із прямими кутами й мають ім'я, що позначає процес дії і номер (ідентифікатор); інше ім'я відображає основний вихід (результат) роботи (наприклад, "Виготовлення виробу"). Часте ім'я міняється в процесі моделювання, оскільки модель може уточнюватися й редагуватися. Ідентифікатор роботи привласнюється при створенні й не міняється ніколи. Навіть якщо робота буде вилучена, її ідентифікатор не буде знову використовуватися для інших робіт. Звичайно номер роботи складається з номера батьківської роботи й порядкового номера на поточній діаграмі.

Зв'язки показують взаємини робіт. Усі зв'язки в IDEF3 односпрямовані й можуть бути спрямовані куди завгодно, але звичайно діаграми IDEF3 намагаються побудувати так, щоб зв'язки були спрямований ліворуч праворуч. В IDEF3 розрізняють три типи стрілок, що зображують зв'язки, стиль яких установлюється через меню Edit/Arrow Style:

Назва звя’зку	Вид звя’зку	Смисл звя’зку
Старший зв’язок		Означає, що друга робота почне виконуватись після завершення першої роботи.
Зв’язок відносини		Означає, що друга робота може початися і даже закінчитися до того моменту, як закінчиться виконання першої роботи.
Зв’язок потоки об’єктів		Одночасно означає часову послідовність робіт й матеріальний або інформаційний потік. В даному випадку друга робота почне виконуватися після завершення першої роботи. При чому виходом першої роботи є об’єкт назва якого написана над стрілкою (в даному випадку це документ). Цей зв’язок також означає, що об’єкт породжуємий першою роботою використовується в наступних роботах

Крім наявності декількох типів зв'язків між роботами в стандарті IDEF3 використовуються логічні оператори, які в цьому випадку називаються

перехрестями також діляться на кілька типів: "ЩоВиключає АБО", "И" і "АБО".

Перехрестя "ЩоВиключає АБО" позначає, що після завершення роботи "А" (рисунок 1), починає виконуватися тільки одна із трьох розташованих паралельно робіт B, 3 або D залежно від умов 1, 2 і 3. Перехрестя "И" позначає, що після завершення роботи "А", починають виконуватися одночасно три паралельно розташовані роботи B, C и D. Перехрестя "АБО" позначає, що після завершення роботи "А", може запуститися будь-яка комбінація трьох паралельно розташованих робіт B, C и D. Наприклад може запуститися тільки одна з них, можуть запуститися три роботи, а також можуть запуститися подвійні комбінації B и C, або C і D, або B і D. Перехрестя ", ЩоВиключає АБО" є самим невизначеним, тому що припускає кілька можливих сценаріїв реалізації бізнес- процесу й застосовується для опису слабко формалізованих ситуацій.

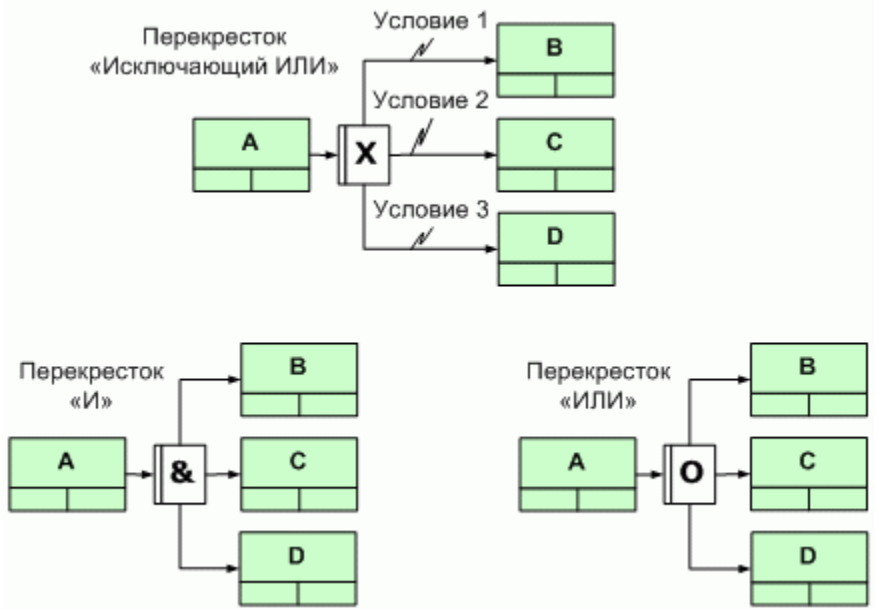


Рисунок 1 – Типи перехресть

Перехрестя "И" і "АБО" підрозділяються ще на два підтипи - синхронні й асинхронні. Перехрестя **синхронного типу** позначають, що роботи B, C и D запускаються одночасно після завершення роботи A. Перехрестя **асинхронного типу** вимог до одночасності не пред'являють.

Наведені на рисунку 2 схеми взаємозв'язку робіт і перехресть називаються схемами **розбіжності**, тому що від перехресть розходяться кілька робіт. Існує й інші схеми взаємозв'язку перехресть і робіт - це так звані схеми **сходження**, коли до перехрестя підходить кілька робіт.

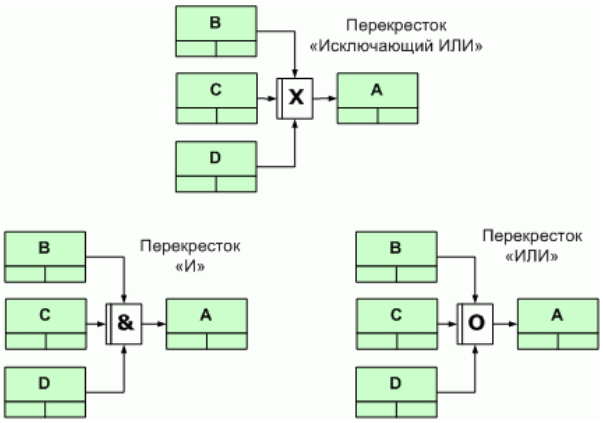


Рисунок 2 – Схеми сходження

Таблиця - Позначення, назви й зміст типів перехресть у схемах сходження й розбіжності

Название перекрестков		Обозначение перекрестков	Смысл перекрестков	
			Схема расхождения	Схема схождения
"Исключающий ИЛИ"		X	Только одна последующая работа запускается	Только одна предшествующая работа должна быть завершена
"И"	Асинхронный	&	Все последующие работы запускаются	Все предшествующие работы должны быть завершены
	Синхронный	&	Все последующие работы запускаются одновременно	Все предшествующие работы должны быть завершены одновременно
"ИЛИ"	Асинхронный	O	Одна или несколько последующих работ запускаются	Одна или несколько предшествующих работ должны быть завершены
	Синхронный	O	Одна или несколько последующих работ запускаются одновременно	Одна или несколько предшествующих работ должны быть завершены одновременно

Приклад:

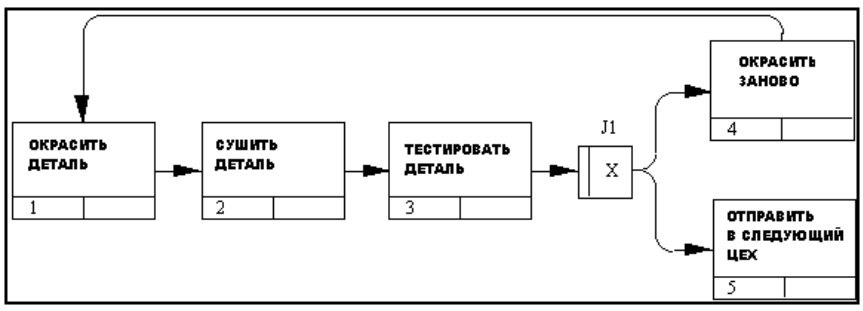


Рисунок 3 – Приклад діаграми PFDD

Сценарій, відображуваний на діаграмі, можна описати в наступному виді:

Деталь надходить у фарбувальний цех, підготовленої до фарбування. У процесі фарбування наноситься один шар емалі при високій температурі. Після цього, проводиться сушіння деталі, після якого починається етап перевірки якості нанесеного шару. Якщо тест підтверджує недостатня якість нанесеного шару (недостатню товщину, неоднорідність і т.д.), то деталь заново пропускається через цех фарбування. Якщо деталь успішно проходить контроль якості, то вона відправляється в наступний цех для подальшої обробки.

Кожний функціональний блок UOB може мати послідовність декомпозицій, і, отже, може бути деталізований з кожної необхідною точністю. Під декомпозицією ми розуміємо виставу кожного UOB за допомогою окремої IDEF3 діаграми. Наприклад, ми можемо декомпозировать UOB "Офарбити Деталь", представивши його окремим процесом і побудувавши для нього свою PFDD діаграму. При цьому ця діаграма буде називатися дочірньої, стосовно зображеної на мал. 1, а та, відповідно до батьківської. Номера UOB дочірніх діаграм мають наскрізну нумерацію, тобто, якщо батьківський UOB має номер "1", те блоки UOB на його декомпозиції будуть відповідно мати номера "1.1", "1.2" і т.д. Застосування принципу декомпозиції в IDEF3 дозволяє структуровано описувати процеси з будь-яким необхідним рівнем деталізації.

Якщо діаграми PFDD технологічний процес "З погляду спостерігача", то інший клас діаграм IDEF3 OSTN дозволяє розглядати той же самий процес "З погляду об'єкта". На рисунку 4.25 представлене відображення процесу фарбування з погляду OSTN діаграми. **Стан об'єкта** (у нашому випадку фарба) і **Зміна станів** є ключовими поняттями OSTN діаграми. Стан об'єкта відображаються окружностями, а їх зміни спрямованими лініями. Кожна лінія має посилання на відповідний функціональний блок UOB, у результаті якого відбулося відображуване їй зміна стану об'єкта.

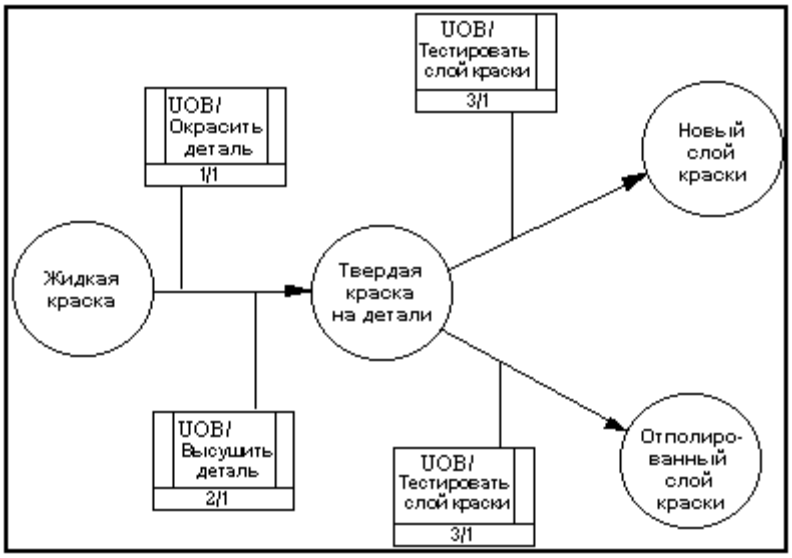


Рисунок 4 – Приклад діаграми OSTN

Основні поняття лекції:

Одиниці роботи - Unit of Work (UOW) - також називані роботами (activity), є центральними компонентами моделі.

Зв'язки показують взаємини робіт. Усі зв'язки в IDEF3 односпрямовані й можуть бути спрямовані куди завгодно.

Старший зв'язок - означає, що друга робота почне виконуватись після завершення першої роботи.

Зв'язок відносин - Означає, що друга робота може початися і даже закінчитися до того моменту, як закінчиться виконання першої роботи.

Зв'язок потоки об’єктів - Одночасно означає часову послідовність робіт й матеріальний або інформаційний потік. В даному випадку друга робота почне виконуватися після завершення першої роботи. При чому виходом першої роботи є об’єкт назва якого написана над стрілкою (в даному випадку це документ). Цей зв’язок також означає, що об’єкт породжуємий першою роботою використовується в наступних роботах.

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.